Domain-Independent Dynamic Programming

by

Ryo Kuroiwa

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy

Graduate Department of Mechanical and Industrial Engineering
University of Toronto

Domain-Independent Dynamic Programming

Ryo Kuroiwa
Doctor of Philosophy
Graduate Department of Mechanical and Industrial Engineering
University of Toronto
2024

Abstract

Dynamic programming (DP) is a framework used in multiple disciplines to solve decision-making problems. In particular, in computer science and operations research (OR), DP algorithms have been developed for combinatorial optimization, a class of problems to make a finite set of decisions to optimize an objective function. In such work, DP algorithms were typically implemented specifically for individual combinatorial optimization problems.

In contrast to problem-specific algorithms, model-based paradigms use general-purpose solvers to solve any problem formulated in a particular form of mathematical model. They aim to decouple modeling and solving a problem: the 'holy grail' of declarative problem-solving. In practice, model-based paradigms such as mixed-integer programming (MIP) and constraint programming (CP) are widely used to solve various combinatorial optimization problems.

We propose domain-independent dynamic programming (DIDP), a novel model-based paradigm for combinatorial optimization based on DP. In DIDP, a user formulates a DP model using a declarative modeling language and then uses a general-purpose DP solver to solve the model. Throughout this dissertation, we develop the modeling language and general-purpose solvers for DIDP.

Our language is based on a state-transition system, inspired by artificial intelligence (AI) planning. However, it is specifically designed for combinatorial optimization: similar to MIP and CP, a user can declaratively include redundant information in a model, which is implied by other parts of the model but may be useful for a solver when made explicit. We demonstrate the modeling capability of our language by formulating eleven combinatorial optimization problems as DP models.

We investigate DIDP solvers using heuristic search, a class of algorithms widely used in the AI community. First, we develop anytime and exact solvers, which improve the solution quality over time and eventually solve the problem optimally. Then, we develop a DIDP solver based on large neighborhood search, which is used to quickly obtain high-quality solutions in MIP and CP. Finally, we develop multi-thread DIDP solvers using parallel heuristic search algorithms. With the developed modeling language and solvers, we demonstrate that DIDP is a promising approach: it empirically outperforms MIP and CP in multiple classes of combinatorial optimization problems.

Acknowledgements

First and foremost, I would like to thank my supervisor, Professor J. Christopher Beck, for his great guidance and support. You always encouraged me to pursue big ideas and interesting research directions while giving me fruitful feedback and timely responses to everything. I learned a lot from you and have been really happy to work with you.

I would like to thank my internal committee members, Professor Sheila McIlraith and Professor Eldan Cohen, for their commitment over time. I would also like to thank my external committee members, Professor Andre Augusto Cire and Professor Laurent Michel.

Thank you to TIDEL members, Alex, Chiara, Margarita, Jasper, Minori, Arnoosh, Jason, Giovanni, Louis, Anton, Litong, Victor, Tan, Oksana, Eugene, Anubhav, Kyle, Man-Qiu, Sonia, Lily, Lea, Nivetha, and Max.

I would like to thank Professor Thorsten Koch and Dr. Yuji Shinano at Zuse Institute Berlin for inviting me to Berlin twice. I also thank my friends met in Zuse Institute Berlin, Mark, Franziska, Thai, Huong, Tu, Hanqiu, Ida, Shanwen, Jaap, Stephanie, Maxwell, Gioni, Tim, and Suresh.

Thank you to my friends Bansho, Naoya, Kotaro, Yoshito, Yutaka, and Yuzuki, who started the undergraduate program together with me and are now working in different fields, for interesting interdisciplinary discussions. Also, thank you to Ro and Shinya for visiting me overseas during my Ph.D.

Lastly, I would like to thank my parents and brother.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Dynamic programming (DP) [27] is a commonly used computational method to solve diverse decision-making problems. In DP, a sequence of decisions is made over multiple stages to optimize an objective function, with the result of the decisions up to the current stage represented by a state. This structure is represented by a Bellman equation [27], a recursive equation where the optimal objective value of a state in the current stage is defined by the optimal objective values of states in the next stage. An optimal solution for a problem can be obtained by solving the Bellman equation.

One important application of DP is combinatorial optimization [259], a class of problems requiring a set of discrete decisions to be made to optimize an objective function. Combinatorial optimization has wide real-world application fields including transportation [422], scheduling [342], and manufacturing [53] and thus has been an active research topic in artificial intelligence (AI) [370, 25, 31], operations research (OR) [209, 259], and discrete mathematics [259]. In previous work, DP methods have been typically implemented as specialized algorithms to solve specific combinatorial optimization problems [29, 28, 198, 318, 104, 2, 109, 46, 358, 152, 357, 355, 153, 17, 187].

Unlike problem-specific algorithms, model-based paradigms are general and declarative problem-solving frameworks. In such paradigms, a user formulates a problem as a mathematical model and then solves it using a general-purpose solver, i.e., a user just needs to define a problem to solve it. Therefore, they represent steps toward the 'holy grail' of declarative problem-solving [145]. Benefitting from this declarative nature, model-based paradigms such as mixed-integer programming (MIP) and constraint programming (CP) have been applied to a wide range of combinatorial optimization problems [222, 49, 151, 350, 287, 363, 54, 248, 307, 317, 277]. Such paradigms employ constraint-based representations: a problem is defined as the optimization of an objective function of decision variables, whose joint assignments are restricted by constraints. In contrast, DP is based on a state-based representation of a problem.

State-based modeling is used in domain-independent AI planning [165], a model-based paradigm to solve planning problems in AI. However, this work has not been designed for combinatorial optimization. In particular, the AI planning community focuses on methodologies more than problems: their purpose is to compare different solving methods using the same input, and thus they describe a model of a planning problem with only the necessary information to define it [308]. In contrast, the OR community focuses on solving particular combinatorial optimization problems and investigates efficient optimization models for them. For example, in MIP, different decision variables and con-

straints can result in different strengths of linear relaxations while sharing equivalent integer feasible regions [248, 197]. In CP, global constraints are specifically designed for common substructures of combinatorial optimization problems so that a solver can exploit them to achieve high performance [432, 280, 391]. Similarly, problem-specific DP methods for combinatorial optimization often exploit problem-specific information to reduce the effort to solve the Bellman equations [318, 104, 109, 46, 358, 152, 357, 355, 153, 17].

Thesis Statement

This dissertation proposes domain-independent dynamic programming (DIDP), a novel model-based paradigm designed for combinatorial optimization based on DP. DIDP is domain independent in the sense that it takes only a declarative mathematical model as input and does not require other information such as problem-specific solving instructions. Thus, we restrict the modeling language to mathematical statements implied by the problem definition, including optional information like dual bounds. In other words, DIDP follows the OR approach that allows a user to investigate efficient optimization models by using different formulations. The central thesis of this dissertation is as follows:

> *DP can be used as a practical model-based paradigm for combinatorial optimization, achieving better performance than existing constraint-based paradigms in multiple problem classes.*

This dissertation approaches the thesis by developing a modeling language and general-purpose solvers for DIDP. In particular, our DIDP solvers meet the standard of general-purpose solvers in existing model-based paradigms for combinatorial optimization: most of them are exact and anytime, and some of them are multi-threaded in addition. To develop the solvers, we use state space search [372], a methodology widely used in AI, which solves a problem by finding a path in an implicitly defined graph. In particular, we employ heuristic search algorithms [335, 116], which use functions called heuristic functions to estimate the path cost. We empirically compare the developed DIDP solvers with commercial solvers for existing constraint-based paradigms using combinatorial optimization problems and demonstrate the advantage of DIDP.

1.1 Dissertation Overview

The dissertation first reviews the literature relevant to DIDP in Chapter 2. Then, Chapter 3 proposes a modeling language for DIDP. The following three chapters (Chapters 4–6) develop DIDP solvers with different characteristics. Finally, Chapter 7 summarizes the contributions and presents future directions.

In Chapter 2, we first introduce DP algorithms used for combinatorial optimization. Then, we describe existing model-based paradigms for combinatorial optimization, discussing their connections to DP. We also review existing model-based DP approaches and argue that they are insufficient to be a practical model-based paradigm for combinatorial optimization.

In Chapter 3, we propose Dynamic Programming Description Language (DyPDL), a modeling formalism for DIDP. Then, we present YAML-DyPDL, a practical modeling language for DyPDL.

We explain YAML-DyPDL with an example of a DP model while presenting the formal syntax in Appendix A.1. We demonstrate the modeling capability of DyPDL and YAML-DyPDL by formulating DP models for eleven classes of combinatorial optimization problems.

In Chapter 4, we develop general-purpose DIDP solvers using heuristic search algorithms. The solvers are anytime and exact: they continuously improve the solution quality and eventually solve a problem optimally. In addition, the solvers provide optimality gaps of intermediate solutions. We empirically compare the developed solvers against commercial MIP and CP solvers and demonstrate that DIDP outperforms them in seven out of the eleven combinatorial optimization problem classes tested. Chapters 3 and 4 are based on a paper currently under review in *Artificial Intelligence* [266], which extends two papers published in the *Proceedings of the International Conference on Automated Planning and Scheduling* [267, 270].

In Chapter 5, we combine DIDP with large neighborhood search (LNS), an algorithmic framework used in MIP and CP to quickly obtain good solutions. We propose large neighborhood beam search (LNBS), an algorithm combining LNS and state space search. We empirically show that the DIDP solver using LNBS achieves a better solution quality than the best DIDP solver developed in Chapter 4 in six out of the eleven problem classes tested. This work is based on a paper published in the *International Conference on Principles and Practice of Constraint Programming* [268].

In Chapter 6, we develop multi-thread DIDP solvers using parallel heuristic search algorithms. We show that the multi-thread DIDP solvers achieve significant speedup up to 32 threads against a single-thread DIDP solver. In addition, we demonstrate that the multi-thread DIDP solvers outperform commercial multi-thread MIP and CP solvers in multiple problem classes. This work is based on a paper published in the *Proceedings of the AAAI Conference on Artificial Intelligence* [269].

Finally, we summarize our contributions in Chapter 7 and discuss future research directions for DIDP.

1.2 Summary of Contributions

The contributions of this dissertation are the development of the modeling language, with which a user can formulate a combinatorial optimization problem as a DP model, and the creation of general-purpose solvers for such models. We formalize the developed language and solvers and empirically evaluate them. In what follows, we describe the contributions of each chapter in detail.

Chapter 3: Modeling Formalism and Language

1. DyPDL, a theoretical formalism to describe a DP model based on a state-transition system. We prove that solving a DyPDL model is undecidable in general while it is decidable under some conditions that are typically satisfied by models of combinatorial optimization problems.

2. YAML-DyPDL, a practical modeling language for DyPDL. It is designed so that a user can incorporate redundant information, which is implied by the other parts of the model but can be useful for a solver, following the standard in OR. We provide an example in the chapter and present the formal syntax in Appendix A.1.

3. DyPDL models (and corresponding YAML-DyPDL encodings in Appendix A.2) for eleven combinatorial optimization problem classes, including routing, scheduling, packing, and manufacturing problems. We demonstrate the modeling capability of DyPDL through these models.

Chapter 4: Heuristic Search Solvers for Domain-Independent Dynamic Programming

1. Theoretical formalization of heuristic search for DyPDL. We propose a generic heuristic search procedure to solve a class of DyPDL models including those formulated in Chapter 3. The procedure can exploit redundant information included in a DyPDL model and provide the optimality gap for an intermediate solution. We prove its theoretical correctness, completeness, and optimality.

2. General-purpose DIDP solvers using heuristic search. We instantiate the generic heuristic search procedure with existing heuristic search algorithms and develop seven DIDP solvers. We implement an open-source software framework to use DIDP solvers with YAML-DyPDL.

3. An empirical evaluation of the DIDP solvers. We evaluate the developed DIDP solvers using the eleven combinatorial optimization problem classes, for which DyPDL models are developed in Chapter 3. We demonstrate that memory consumption is an important issue for DIDP solvers, and complete anytime beam search (CABS), a memory-efficient solver based on a heuristic search algorithm called beam search, achieves the best performance. We also show that CABS outperforms commercial MIP and CP solvers in seven out of the eleven problem classes. In addition, we evaluate the importance of incorporating a dual bound of the objective function, which is redundant information, in a DyPDL model.

Chapter 5: Large Neighborhood Beam Search

1. LNBS, a heuristic search algorithm combining LNS and beam search. It improves a given solution path by removing a partial path and searching for a better one. LNBS is composed of strategies to select the length and starting point of the partial path to remove. We show that an existing LNS algorithm and CABS can be viewed as special cases of LNBS.

2. A DIDP solver using LNBS. We propose concrete strategies for LNBS to develop a DIDP solver. In particular, the strategy to select the length of the partial path uses multi-armed bandits. We prove that with the proposed strategies, LNBS has a guarantee of completeness.

3. An empirical evaluation of LNBS. We demonstrate that the DIDP solver using LNBS achieves a better solution quality than CABS in six out of the eleven combinatorial optimization problem classes and better solution quality than MIP and CP in seven of these classes. However, we also observe that CABS outperforms LNBS in some problem classes. Our experimental analysis suggests that LNBS performs better than CABS when they are used with a less informative heuristic function.

Chapter 6: Parallel Beam Search

1. Shared beam search (SBS) and hash-distributed beam search (HDBS), parallel beam search algorithms. SBS uses a shared data structure, and HDBS uses message passing. While SBS is

a relatively straightforward adaptation of previous work, HDBS combines mechanisms used for different heuristic search algorithms to parallelize beam search. We provide theoretical proofs for their correctness.

2. Multi-thread DIDP solvers. We implement the proposed algorithms in the software framework for DIDP and develop DIDP solvers by combining them with CABS.

3. An empirical evaluation of the multi-thread DIDP solvers. Using 32 threads, we demonstrate that the multi-thread DIDP solvers, particularly those based on HDBS, outperform the single-thread CABS, achieving 9 to 36 times speedup on average. We investigate super linear speedup and slowdown observed in some instances. Our analysis suggests that due to the strong dual bound function used in these instances, small perturbations in search behavior caused by parallelization result in significant differences in performance. We also show that the multi-thread DIDP solvers outperform commercial multi-thread MIP and CP solvers in multiple problem classes where the single-thread CABS is better than the single-thread MIP and CP solvers while also demonstrating larger speedups.

The work present in this dissertation has been published or submitted in the following papers.

- Ryo Kuroiwa and J. Christopher Beck. "Domain-Independent Dynamic Programming: Generic State Space Search for Combinatorial Optimization". In: *Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS)*. 2023, pp. 236–244.

- Ryo Kuroiwa and J. Christopher Beck. "Solving Domain-Independent Dynamic Programming Problems with Anytime Heuristic Search". In: *Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS)*. 2023, pp. 245–253.

- Ryo Kuroiwa and J. Christopher Beck. "Large Neighborhood Beam Search for Domain-Independent Dynamic Programming". In: *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*. 2023, pp. 23:1–23:22.

- Ryo Kuroiwa and J. Christopher Beck. "Parallel Beam Search Algorithms for Domain-Independent Dynamic Programming". In: *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI)*. 2024, pp. 245–253.

- Ryo Kuroiwa and J. Christopher Beck. "Domain-Independent Dynamic Programming". Submitted to *Artificial Intelligence*. 2024.

Chapter 2

Literature Review

In this chapter, we review the background and the related literature to this dissertation. The main topic of this dissertation is domain-independent dynamic programming (DIDP), a model-based paradigm based on dynamic programming (DP). Therefore, we consider the following three topics: methodologies for DP in combinatorial optimization, model-based paradigms, and model-based DP frameworks. We first introduce DP and then focus on each of the three topics.

2.1 Dynamic Programming

Dynamic programming (DP) [27] is a framework to solve decision-making problems. In DP, we sequentially make decisions over multiple (possibly infinite) *stages*. A *state* represents the result of decisions up to the current stage, and the *initial state* is given for the first stage. A *policy* selects a decision given the current state. We want to find an optimal policy that minimizes or maximizes an objective function associated with decisions. DP requires an optimal policy to satisfy the *Principle of Optimality* (Bellman [27], p.83):

> An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

Bellman [27] considered two types of structures for outcomes of decisions: *deterministic* and *stochastic*. While we mainly focus on deterministic DP in this dissertation, we also introduce stochastic DP for completeness.

2.1.1 Deterministic Dynamic Programming

In deterministic DP, a decision and the state in which the decision is made uniquely determine the resulting state. Let S^0 be the initial state, $\mathcal{T}(S)$ be a set of possible decisions in state S, and $S[\![\tau]\!]$ be a state resulting from decision τ in state S. A policy maps a state S to a decision in $\mathcal{T}(S)$. The objective value of a policy π is $V_\pi(S^0)$, which is recursively defined as

$$V_\pi(S) = R(S, \pi(S), V_\pi(S[\![\pi(S)]\!])) \tag{2.1}$$

where R is a function that maps a state, a decision, and the objective value of the resulting state to the objective value. In this chapter, we assume minimization for ease of presentation. With the Principle of Optimality, we assume that the optimal objective value is represented by $V(S^0)$, where V is a *value function* recursively defined as

$$V(S) = \min_{\tau \in \mathcal{T}(S)} R(S, \tau, V(S[\![\tau]\!])). \tag{2.2}$$

Equation (2.2) is called a *Bellman equation*.

2.1.2 Stochastic Dynamic Programming

In the stochastic case, there can be multiple resulting states following a probability distribution. Markov decision processes (MDPs) [27, 219] are widely used to formulate such decision-making problems in the artificial intelligence (AI) community [371, 413]. Following the convention in MDPs, we use the term action instead of decision and denote a set of possible actions in state S by $\mathcal{A}(S)$ instead of $\mathcal{T}(S)$. We also denote the set of states by $\mathcal{S}$. Using action a in state S results in a state $S' \in \mathcal{S}$ with the probability of $p(S'|S, a)$ and yields the reward $r(S, a, S')$. The objective is to find a policy that maximizes the cumulative reward. Given the reward r_t obtained in stage t, the cumulative reward is defined as $\sum_{t=0,\ldots,} \gamma^t r_t$, where $\gamma \in (0, 1]$ is a discount factor. For a finite horizon MDP, which has only a finite number of stages, $\gamma = 1$ can be used, resulting in the sum of the rewards over stages. For an infinite horizon MDP, which has an infinite number of stages, $\gamma < 1$ is used to make the cumulative reward converge to a finite value. The Bellman equation is formulated as

$$V(S) = \max_{a \in \mathcal{A}(S)} \sum_{S' \in \mathcal{S}} p(S'|S, a)(r(S, a, S') + \gamma V(S')). \tag{2.3}$$

When the probability distribution p and the reward function r are unknown, and a policy is learned from observations of states and rewards resulting from using actions, the problem is called reinforcement learning, which is actively studied in AI [414]. While extending DIDP to stochastic DP or reinforcement learning might be possible in the future, we focus on deterministic DP in this dissertation as mentioned above.

2.1.3 Dynamic Programming for Combinatorial Optimization

We focus on *combinatorial optimization* problems [259], where we make a finite set of discrete decisions to minimize or maximize the objective function. Therefore, we mainly focus on deterministic DP such that $\mathcal{T}(S)$ is a finite set for any state S, and the number of stages is finite. For the latter property, we assume that from any state, after making a finite number of decisions, we reach a *base state* S, where no decision is available, i.e., $\mathcal{T}(S) = \emptyset$. A policy π does not define a decision for such a state, and the objective value is non-recursively defined as

$$V_\pi(S) = v_S \tag{2.4}$$

where v_S is a constant. Accordingly, given a base state S, the value function V returns

$$V(S) = v_S. \tag{2.5}$$

With these assumptions, an optimal policy can be represented by a finite sequence of decisions with which we can reach a base state from the initial state.

DP has been used for combinatorial optimization over the past six decades. In 1950's, Jackson [233] proposed a DP algorithm for the simple assembly line balancing problem to minimize the number of stations (SALBP-1). Bellman [29] proposed a DP algorithm to solve the shortest path problem, which is now called the Bellman-Ford algorithm [29, 138, 394]. In 1960's, Bellman [28] proposed a DP formulation for the traveling salesperson problem (TSP). Held and Karp [198] also independently developed an equivalent DP formulation for TSP and proposed DP formulations for single machine scheduling problems and SALBP-1. Since then, DP has been applied to various combinatorial optimization problems including knapsack [318, 348, 408, 62], routing [318, 76, 104, 103, 109, 46, 358, 357, 355, 183, 16, 17, 187, 299, 365, 298, 60, 171, 169, 62, 373, 170, 87, 88, 257], scheduling [1, 2, 153], and manufacturing [152, 348, 408].

2.2 Methodologies for Dynamic Programming

We review existing methodologies to find an optimal policy for DP. We mainly focus on methodologies to solve combinatorial optimization problems using deterministic DP. In what follows, for the sake of simplicity, we assume a minimization problem represented as the Bellman equation in Equation (2.2). In general, the methodologies considered here can be used for maximization as well. First, we introduce four algorithms that directly use a Bellman equation: successive approximation, approximation in policy space, bottom-up computation, and recursion with memoization. Then, we describe mechanisms and algorithms using additional information: dominance detection, branch-and-bound, state space relaxation, decision diagram-based branch-and-bound, and heuristic search.

2.2.1 Successive Approximations

Bellman [27] introduced a method called successive approximations. In this approach, the value function V is approximated by a sequence of functions. First, V_0 is initialized to some function. In each iteration, V_{i+1} is computed as

$$V_{i+1}(S) = \min_{\tau \in \mathcal{T}(S)} R(S, \tau, V_i(S[\![\tau]\!])). \tag{2.6}$$

Bellman proved that V_i eventually converges to the value function V under some conditions. One example of successive approximations is the Bellman-Ford algorithm [29, 138, 394]. Given a directed graph (N, E), where N is the set of nodes, $E \subseteq N \times N$ is the set of edges, and each edge $(j, k) \in E$ has the weight c_{jk}, the Bellman-Ford algorithm computes the shortest path cost from a node j to $t \in N$ using the following equation:

$$V_{i+1}(j) = \begin{cases} 0 & \text{if } j = t \\ \min_{(j,k)\in E} c_{jk} + V_i(k) & \text{if } j \neq t. \end{cases} \tag{2.7}$$

For MDPs, the algorithm called value iteration is based on successive approximations [219, 412].

Algorithm 1 Bottom-up computation for the DP formulation of TSP in Equation (2.9).

Let T be a table mapping a state to a numeric value
for all $i \in N \setminus \{0\}$ **do**
    $T[(\emptyset, i)] \leftarrow c_{i0}$
for $k = 0, ..., n-3$ **do**
    for all (U, j) such that $U \subseteq N \setminus \{0\}, j \in N \setminus (U \cup \{0\})$ and $|U| = k$ **do**
        for all $i \in N \setminus (U \cup \{0, j\})$ **do**
            if $T[(U \cup \{j\}, i)]$ is initialized **then**
                $T[(U \cup \{j\}, i)] \leftarrow \min\{T[(U \cup \{j\}, i)], c_{ij} + T[(U, j)]\}$
            else
                $T[(U \cup \{j\}, i)] \leftarrow c_{ij} + T[(U, j)]$
$T[(N \setminus \{0\}, 0)] \leftarrow \infty$
for all $j \in N \setminus \{0\}$ **do**
    $T[(N \setminus \{0\}, 0)] \leftarrow \min\{T[(N \setminus \{0\}, 0)], c_{0j} + T[(N \setminus \{0, j\}, j)]\}$
return an optimal policy reconstructed from T

2.2.2 Approximation in Policy Space

Bellman [27] also proposed approximation in policy space. In iteration i, this method maintains an approximation of an optimal policy π_i. First, π_0 is initialized to some policy. Then, π_{i+1} is computed from V_{π_i} so that

$$\pi_{i+1}(S) \in \arg\min_{\tau \in \mathcal{T}(S)} R(S, \tau, V_{\pi_i}(S[\![\tau]\!])). \tag{2.8}$$

Bellman proved that π_i eventually converges to an optimal policy under some conditions. This method is called policy iteration in MDPs [219, 412].

2.2.3 Bottom-up Computation

Bottom-up computation iteratively computes the value of V. Unlike successive approximations and approximation in policy space, which compute $V_i(S)$ for each state S over multiple iterations, bottom-up computation computes $V(S)$ once for each state. Bottom-up computation requires ordering states so that $V(S)$ only depends on preceding states in the order. For example, bottom-up computation is used with DP for TSP [29, 198]. In TSP, a set of customers $N = \{0, ..., n-1\}$ is given, and visiting customer j from i incurs the travel time c_{ij}. The objective is to minimize the total travel time of a tour that visits each customer exactly once starting from and returning to the depot 0. In DP for TSP, a decision corresponds to visiting one customer, and a state is represented by the set of unvisited customers U and the current location i. The initial state is $(N \setminus \{0\}, 0)$. The Bellman equation is written as

$$V(U, i) = \begin{cases} \min_{j \in U} c_{ij} + V(U \setminus \{j\}, j) & \text{if } U \neq \emptyset \\ c_{i0} & \text{if } U = \emptyset. \end{cases} \tag{2.9}$$

To solve this Bellman equation, we can use a bottom-up computation algorithm by ordering states according to the number of unvisited customers. Starting from $k = 0$, we iteratively compute the objective values of states where $k + 1$ customers are unvisited using the results with k. We show

Algorithm 2 Recursion with memoization for the DP formulation in Equations (2.2) and (2.5).

function EVALUATE(S, T)
  if $T[S]$ is initialized **then return** $T[S]$
  if S is a base state **then return** v_S
  $v \leftarrow \infty$
  for all $\tau \in \mathcal{T}(S)$ **do**
    $v \leftarrow \min\{v, R(S, \tau, \text{EVALUATE}(S[\![\tau]\!], T))\}$
  $T[S] \leftarrow v$
  return v
Let T be a table mapping a state to a numeric value
EVALUATE(S^0, T)
return an optimal policy reconstructed from T

pseudo-code for this algorithm in Algorithm 1. The algorithm computes $V(S)$ and stores it in a table T.

2.2.4 Recursion with Memoization

While bottom-up computation evaluates each state only once, it requires an ordering of states, which is specific to the problem. An alternative approach is to perform recursion following the Bellman equation with memoization [312], which stores the evaluation results of function calls in memory. Unlike bottom-up computation, the order in which states are evaluated is automatically determined by recursion. With memoization, each state is evaluated only once. We present pseudo-code of the algorithm for the generic Bellman equation in Equations (2.2) and (2.5). Recursion with memoization was used for combinatorial optimization problems such as knapsack [348], the minimization of open stacks problem (MOSP) [152, 348, 408], and talent scheduling [153].

2.2.5 Dominance Detection

Bottom-up computation and recursion with memoization avoid evaluating the same state multiple times by storing it in memory. Dominance detection is a generalization of this strategy: it avoids evaluating a state if it is known to be equal to or worse than an already evaluated state. DP algorithms with dominance detection have been actively studied for the shortest path problems with resource constraints (SPPRC) [232]. In SPPRC, a directed graph (N, E) and m resources $1, ..., m$ are given, and each node $i \in N$ is associated with the resource window $[a_{ik}, b_{ik}]$ for each resource k. Each edge $(i, j) \in E$ is associated with the weight c_{ij} and the resource consumption d_{ijk} for each resource k. Given cumulative resource consumptions $r_1, ..., r_m$ at node i, a feasible path can use edge (i, j) only if $r_k + d_{ijk} \leq b_{jk}$ and updates the consumption of resource k to $\max\{a_{jk}, r_k + d_{ijk}\}$ when reaching node j from node i. The objective is to find a shortest feasible path from a node $s \in N$ to a node $t \in N$ starting with the cumulative resource consumption $r_k = 0$ at s for each resource k. SPPRC can be solved by DP algorithms called labeling algorithms [232], which are generalizations of shortest path algorithms such as the Bellman-Ford algorithm and Dijkstra's algorithm [105].

Labeling algorithms for SPPRC exploit dominance between different paths. Suppose that the cumulative resource consumptions of a path p from s to i are $r_1, ..., r_m$, the cumulative resource consumptions of another path q from s to i are $r'_1, ..., r'_m$ with $r_k \leq r'_k$ for each resource k, and p

Algorithm 3 Labeling algorithm for SPPRC in a directed graph (N, E) with m resources. A path is represented by a sequence of nodes.

$O \leftarrow \{(\langle s\rangle, 0, ..., 0, 0)\}$
$G \leftarrow \emptyset$
while $O \neq \emptyset$ **do**
    Let $(\langle s, ..., i\rangle, r_1, ..., r_m, g) \in O$
    $O \leftarrow O \setminus \{(\langle s, ..., i\rangle, r_1, ..., r_m, g)\}$
    for all $(i, j) \in E$ such that $r_k + d_{ijk} \leq b_{jk}$ for $k = 1, ..., m$ **do**
        $O \leftarrow O \cup \{(\langle s, ..., i, j\rangle, \max\{a_{j1}, r_1 + d_{ij1}\}, ..., \max\{a_{im}, r_m + d_{ijm}\}, g + c_{ij})\}$
    $G \leftarrow G \cup \{(\langle s, ..., i\rangle, r_1, ..., r_m, g)\}$
    Remove (possibly a subset of) labels in O dominated by another label in $O \cup G$
Let $(p, r'_1, ..., r'_m, g') \in \arg\min_{(\langle s,...,t\rangle, r_1, ..., r_m, g) \in G} g$
return p

has an equal or smaller path cost than q. To find a shortest feasible path from s to t, we can ignore extensions of q as long as we consider extensions of p, i.e., p dominates q. In labeling algorithms, instead of a state, a label $(p, r_1, ..., r_m, g)$ is used, where p is a path, $r_1, ..., r_m$ are the cumulative resource consumptions, and g is the current path cost. Instead of checking if a state is already stored in memory, a labeling algorithm checks if a label is dominated by another label stored in memory.

We present pseudo-code for a generic labeling algorithm, following Irnich and Desaulniers [232], in Algorithm 3. In the pseudo-code, we represent a path by a sequence of nodes, e.g., a path from s to i is $\langle s, ..., i\rangle$. The algorithm stores the set of labels to process in O, initialized with $O = \{(\langle s\rangle, 0, ..., 0, 0)\}$, and the set of processed labels in G, initialized with $G = \emptyset$. In each iteration, the algorithm removes one label from O, inserts new labels extending the path associated with the label into O, and inserts the processed label into G. How to select the label to process from O depends on the concrete labeling algorithm. For example, the algorithm proposed by Desrochers and Soumis [104] selects the label based on a lexicographic order of the cumulative resource consumptions, which guarantees, under some conditions, that the processed label will not be dominated by labels generated later. Algorithms with such a guarantee are called label setting algorithms, and other algorithms are called label correcting algorithms [232]. After inserting the generated labels into O, the algorithm removes dominated labels from O. How to detect and remove dominated labels also depends on the concrete algorithm. Since exactly detecting dominance between all labels in $O \cup G$ can be computationally expensive, some labeling algorithms check only a subset of labels in $O \cup G$ for each label [299, 373].

While Algorithm 3 extends forward paths from s toward t, bidirectional labeling algorithms, which search from both forward and backward directions, were also proposed [358, 60, 373]. Such algorithms extend backward paths from t in addition to forward paths from s and check if forward and backward paths collide.

2.2.6 Branch-and-Bound

Bottom-up computation and recursion with memoization enumerate all possible states that can be reached from the initial state. Even with dominance detection, we still need to enumerate all states that are not dominated. In contrast, Morin and Marsten [318] used branch-and-bound [279] for DP, which can avoid enumerating some states by using bounds on the objective function. Branch-and-bound is a general algorithmic framework to solve optimization problems. A branch-and-bound

Algorithm 4 Bottom-up branch-and-bound for the DP formulation of TSP in Equation (2.9). A primal bound u and a dual bound function l satisfying Equation (2.11) are given as input.

Let T be a table mapping a state to a numeric value
for all $i \in N \setminus \{0\}$ **do**
    if $l(\emptyset, i) + c_{i0} < u$ **then**
        $T[(\emptyset, i)] \leftarrow c_{i0}$
for $k = 0, ..., n-3$ **do**
    for all (U, j) such that $T[(U, j)]$ is initialized and $|U| = k$ **do**
        for all $i \in N \setminus (U \cup \{0, j\})$ such that $l(U \cup \{j\}, i) + c_{ij} + T[(U, j)] < u$ **do**
            if $T[(U \cup \{j\}, i)]$ is initialized **then**
                $T[(U \cup \{j\}, i)] \leftarrow \min\{T[(U \cup \{j\}, i)], c_{ij} + T[(U, j)]\}$
            else
                $T[(U \cup \{j\}, i)] \leftarrow c_{ij} + T[(U, j)]$
$T[(N \setminus \{0\}, 0)] \leftarrow u$
for all $j \in N \setminus \{0\}$ such that $T[(N \setminus \{0, j\}, j)]$ is initialized **do**
    $T[(N \setminus \{0\}, 0)] \leftarrow \min\{T[(N \setminus \{0\}, 0)], c_{0j} + T[(N \setminus \{0, j\}, j)]\}$
return an optimal policy reconstructed from T

algorithm divides the original problem into subproblems so that the optimal solution for the problem is computed from the optimal solutions for the subproblems. This procedure is called branching. The algorithm solves the subproblems by recursively branching until reaching a trivial subproblem, which can be solved without branching. In doing so, the algorithm may avoid solving a subproblem by using bounds. In minimization, the algorithm maintains an upper bound (primal bound) on the optimal objective value for the original problem and computes a lower bound (dual bound) on the optimal objective value of each subproblem. Suppose that the optimal solution cost for the original problem is exactly the same as the optimal solution cost for its best subproblem, which minimizes the objective function among the subproblems. Then, if the dual bound of a subproblem is greater than or equal to the primal bound, the algorithm does not need to solve that subproblem since a better solution will not be obtained from it.

Morin and Marsten [318] combined branch-and-bound and bottom-up computation for DP. Their algorithm requires the objective function to be additive,[1] i.e., the Bellman equation is defined as

$$V(S) = \begin{cases} v_S & \text{if } S \text{ is a base state} \\ \min_{\tau \in \mathcal{T}(S)} w_\tau(S) + V(S[\![\tau]\!]) & \text{otherwise} \end{cases} \tag{2.10}$$

where w_τ returns a constant given a state. Given a state S, the algorithm computes the dual bound to reach S from the initial state S^0. For minimization, given any policy π, the dual bound $l(S)$ satisfies

$$l(S) \leq \sum_{i=0}^{m-1} w_{\pi(S^i)}(S^i) \tag{2.11}$$

where $S^{i+1} = S^i[\![\pi(S^i)]\!]$ and $S^m = S$. Given a primal bound u, if $l(S) + V(S) \geq u$, we can ignore S. The primal bound can be the objective value of any policy. Morin and Marsten applied this algorithm to TSP and nonlinear knapsack. We show pseudo-code for the branch-and-bound

[1] Morin and Marsten [318] mentioned that their algorithm can be extended to a cost defined by any commutative isotonic binary operator such as multiplication and taking the maximum or minimum.

Algorithm 5 Recursion with memoization and local bounding for the DP formulation in Equations (2.10). A dual bound function l satisfying Equation (2.12) is given as input.

```
1: function EVALUATE(S, T)
2:     if T[S] is initialized then return v
3:     if S is a base state then return v_S
4:     v ← ∞
5:     T̂ ← T(S)
6:     while T̂ ≠ ∅ do
7:         Let τ ∈ arg min_{τ'∈T̂} w_τ'(S) + l(S⟦τ'⟧)
8:         T̂ ← T̂ \ {τ}
9:         if w_τ(S) + l(S⟦τ⟧) ≥ v then
10:            break
11:        v ← min{v, w_τ(S)+ EVALUATE(S⟦τ⟧, T))}
12:    T[S] ← v
13:    return v
14: Let T be a table mapping a state to a numeric value
15: EVALUATE(S^0, T)
16: return an optimal policy reconstructed from T
```

algorithm in Algorithm 4 using the DP formulation of TSP in Equation (2.9). In TSP, given a state (U, i), where U is the set of unvisited customers, and i the current location, the cost to reach the state from the target state is the travel time of a route from 0 to i visiting all customers in $N \setminus U$. A dual bound can be obtained by computing a lower bound on the travel time, e.g., by solving the minimum spanning tree problem [199].

Branch-and-bound was also combined with recursion with memoization [152, 348, 153]. For example, Puchinger and Stuckey [348] proposed two strategies: local bounding and argument bounding. We explain their algorithm using the Bellman equation in Equation (2.10).[2] Unlike bottom-up computation, both local bounding and argument bounding use a dual bound function l that satisfies

$$l(S) \leq V(S). \tag{2.12}$$

In local bounding, when computing $V(S) = \min_{\tau \in \mathcal{T}(S)} w_\tau(S) + V(S[\![\tau]\!])$, once we obtain $w_\tau(S) + V(S[\![\tau]\!])$ for some τ (a primal bound), we can ignore τ' without computing $V(S[\![\tau']\!])$ if $w_{\tau'}(S) + l(S[\![\tau']\!]) \geq w_\tau(S) + V(S[\![\tau]\!])$. To obtain a good primal bound earlier, the algorithm evaluates decisions in the ascending order of $w_\tau(S) + l(S[\![\tau]\!])$. We show pseudo-code for recursion with memoization and local bounding in Algorithm 5.

Local bounding still requires to compute $V(S[\![\tau]\!])$ for at least one τ. In contrast, argument bounding uses a primal bound given as input, similar to branch-and-bound with bottom-up computation. We show pseudo-code for recursion with memoization and argument bounding in Algorithm 6. The recursive function takes a value u in addition to a state S and the table T and returns $V(S)$ if $V(S) < u$. If $V(S) > u$, it returns a value less than or equal to $V(S)$ and greater than or equal to u; such a value is sufficient for the caller (line 9) to decide if $V(S) < u$ or not since $w_\tau(S) + V(S[\![\tau]\!]) \geq u$ if $\textsc{Evaluate}(S[\![\tau]\!], u - w_\tau(S), T) \geq u - w_\tau(S)$. In the table T, the algorithm stores a value $v \leq V(S)$ and a flag is_optimal indicating if $v = V(S)$. Given a state S stored in T, if is_optimal $= \top$, the function returns $v = V(S)$. If is_optimal $= \bot$ and $v \geq u$, the function returns v, which satisfies the

[2] The original algorithm is defined for more general cost structures [348].

Algorithm 6 Recursion with memoization and argument bounding for the DP formulation in Equations (2.10). A primal bound u and a dual bound function l satisfying Equation (2.12) are given as input.

```
1: function EVALUATE(S, u, T)
2:     if T[S] is initialized then
3:         (v, is_optimal) ← T[S]
4:         if is_optimal or v ≥ u then return v
5:     if l(S) ≥ u then return l(S)
6:     if S is a base state then return v_S
7:     v ← ∞
8:     while τ ∈ T(S) do
9:         v ← min{v, w_τ(S) + EVALUATE(S⟦τ⟧, u − w_τ(S), T))}
10:    T[S] ← (v, v < u)
11:    return v
12: Let T be a table mapping a state to a numeric value and a binary flag
13: EVALUATE(S^0, u, T)
14: return an optimal policy reconstructed from T
```

requirement of the function since $v \geq u$ is a dual bound on $V(S)$, and $v = V(S)$ if $V(S) = u$. If is_optimal $= \perp$ and $v < u$, the function needs to compute a new v such that $v = V(S)$ if $V(S) < u$ and $u \leq v \leq V(S)$ if $V(S) \geq u$ using recursion. When storing S in T in line 10, $v < u$ indicates if $v = V(S)$.

In addition to bottom-up computation and recursion with memoization, branch-and-bound was also combined with labeling algorithms [358, 299, 60]. Contrary to introducing bounding in DP algorithms, methods that use mechanisms similar to DP, which store subproblems in memory to avoid solving them again, in branch-and-bound algorithms were also studied. Such approaches are called by various names, such as test for dominance [255], subproblem dominance [77], branch, bound, and remember [239, 388, 319], branch and memorize [389], caching [403], and nogood recording [79].

2.2.7 State Space Relaxation

Christofides, Mingozzi, and Toth [76] proposed state space relaxation, which uses DP to obtain a dual bound. The idea is to solve a relaxation of the problem, which ignores some aspects of the original problem and thus is easier to solve. For example, for the DP formulation for TSP in Equation (2.9), they proposed the n-path relaxation, which replaces the set of unvisited customers $U \subseteq N = \{0, ..., n-1\}$ with its cardinality k and reduces the number of states from $O(n2^n)$ to $O(n^2)$. The n-path relaxation considers the optimal cost to visit $n-1$ customers in $N \setminus \{0\}$ and return to the depot 0, allowing visiting the same customer multiple times. The optimal cost for the n-path relaxation is a lower bound on the optimal cost of the original problem. The Bellman equation for the n-path relaxation is

$$V(k,i) = \begin{cases} \min_{j \in N \setminus \{0,i\}} c_{ij} + V(k-1,j) & \text{if } k \geq 1 \\ c_{i0} & \text{if } k = 0. \end{cases} \tag{2.13}$$

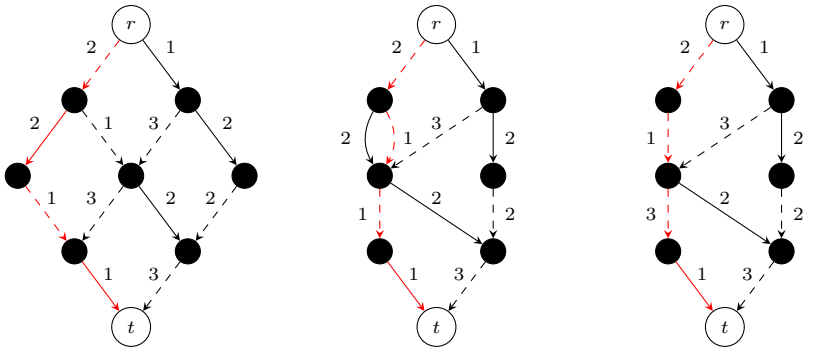
This idea was also applied to variants of TSP [183, 17, 365], scheduling problems [1], and a facility location problem [144].

Boland, Dethridge, and Dumitrescu [46] and Righini and Salani [357, 355] independently proposed decremental state space relaxation, which iteratively solves relaxations while refining them, to solve routing problems. Their approaches are based on the n-path relaxation, but each customer in $C \subseteq N$, called a critical vertex set by Righini and Salani, can be visited at most once. The algorithm starts from $C = \emptyset$, which is the n-path relaxation. In each iteration, the relaxed problem is solved by a DP algorithm. If the optimal solution visits the same customer only once, it is also an optimal solution for the original problem. Otherwise, some of the customers that are visited multiple times are added to C, and the procedure is repeated.

2.2.8 Decision Diagram-Based Branch-and-Bound

Bergman et al. [35] proposed a branch-and-bound algorithm that uses decision diagrams (DDs) to obtain primal and dual bounds, based on the connection between DP and DDs pointed by Hooker [217]. This algorithm can be viewed as a combination of branch-and-bound and state space relaxation. A DD is a directed acyclic graph partitioned into layers $l = 0, ..., n$. Layer 0 contains only one node called the root. An edge (i, j, d) connects node i in layer l and node j in layer $l + 1$ with label d, and nodes in layer n do not have outgoing edges. There can be multiple edges connecting the same nodes with different labels. Bergman et al. [35] focused on DDs that have only one node, called the terminal, in layer n for their purpose. A DP formulation is represented as a DD, where each node in layer l corresponds to a DP state in stage l, and edges are labeled with decisions: an edge $(S, S[\![\tau]\!], \tau)$ exists if $\tau \in \mathcal{T}(S)$. Here, for simplicity, we denote a DD node by its corresponding DP state assuming that there is a one-to-one mapping between them. This assumption is valid in the DP formulation for TSP in Equation (2.9), for example. However, in general, if the same state appears in different stages, multiple DD nodes in different layers correspond to that state. Bergman et al. [35] assume that the DP formulation has the additive cost structure (Equation (2.10)), and there is only one base state S with $V(S) = v_S = 0$ in stage n. With these assumptions, edge $(S, S[\![\tau]\!], \tau)$ is associated with the weight $w_\tau(S)$, and the optimal objective value is the shortest path cost (or the longest path cost for maximization) from the root to the terminal in the DD. This DD is called the exact DD. Constructing the exact DD corresponds to enumerating all DP states, which can be computationally expensive. Bergman et al. [35] addressed this issue by obtaining dual and primal bounds from relaxed and restricted DDs, which are smaller than the exact DD. We present examples of exact, relaxed, and restricted DDs in Figure 2.1.

A dual bound, which is a lower bound on the shortest path cost, is obtained from a relaxed DD [7, 36]. The concept of relaxed DDs is similar to state space relaxation: it has fewer DD nodes and the shortest path cost is a lower bound on that of the exact DD. More formally, for each path from the root to the terminal in the exact DD, a relaxed DD must have a path from the root to the terminal with the same edge labels and a less or equal cost. Bergman et al. [35] construct relaxed DDs with at most b nodes in each layer, where b is a parameter. In their algorithm, a relaxed DD is constructed layer by layer from the root. For each node S in the current layer and each decision $\tau \in \mathcal{T}(S)$, a node $S[\![\tau]\!]$ in the next layer is created. After generating all nodes in the next layer, if the number of the nodes exceeds the limit b, a subset of nodes M are selected and merged into one node, and a corresponding DP state S_M is created. Each edge (S', S, τ) with $S \in M$ is updated to (S', S_M, τ), and its weight is updated so that the resulting DD satisfies the requirement of a relaxed DD. These procedures are performed by a problem-specific merging operator, which defines a map



(a) Exact DD. (b) Relaxed DD. (c) Restricted DD.

Figure 2.1: Examples of decision diagrams (DDs), where r is the root and t is the terminal. There are two labels for edges, represented by dashed and solid lines. The numbers presented are the edge weights, and the shortest paths are colored in red.

from a set of states M to a state S_M and how to update the edge weights.

A primal bound is obtained from a restricted DD [37]. For each path from the root to the terminal in a restricted DD, the exact DD must have a path from the root to the terminal with the same edge labels and a less or equal cost. Bergman et al. [35] construct a restricted DD that has at most b nodes in each layer. In their algorithm, when the number of nodes in a layer exceeds b, a subset of nodes are removed from the layer.

DD-based branch-and-bound tries to find a shortest path in the exact DD representing the original problem through repeatedly constructing relaxed and restricted DDs [35]. We present pseudocode for DD-based branch-and-bound in Algorithm 7. The algorithm maintains states in the set O. For each state in $S \in O$, the algorithm also records $g(S)$, the shortest path cost from S^0 to S in the exact DD of the original problem and the corresponding shortest path $\sigma(S)$. In each iteration, the algorithm removes one state S from O and constructs a restricted DD $\overline{B}$ whose root is S. Then, a shortest path σ^t from S to the terminal in $\overline{B}$ is extracted. The path $\langle \sigma(S); \sigma^t \rangle$, which concatenates $\sigma(S)$ and σ^t, is a path from the root to the terminal in the exact DD of the original problem. Therefore, its cost $g(S) + u(S)$ is an upper bound on the optimal objective value. If $\overline{B}$ is actually exact, we have obtained the shortest path cost from S^0 to the terminal via S in the exact DD of the original problem, so we no longer need to consider paths extending $\sigma(S)$. If $\overline{B}$ is not exact, the algorithm creates a relaxed DD $\underline{B}$ and obtains $l(S)$, the shortest path cost from S to the terminal in $\underline{B}$. If $g(S) + l(S) \geq u$, paths extending $\sigma(S)$ will not have better costs than the current solution cost, so we ignore S. Otherwise, we need to investigate paths extending $\sigma(S)$.

Bergman et al. [35] proved that it is sufficient to consider a set of nodes called an exact cut set. A node in $\underline{B}$ is exact if all sequences of decisions that correspond to paths from S to that node in $\underline{B}$ transform S to the same state. An exact cut set is a set of exact nodes such that all paths from S to the terminal pass at least one node in it. Since there can be multiple exact cut sets in $\underline{B}$, the algorithm selects one using some strategy. Then, it extends $\sigma(S)$ to each node S' in the set using a shortest path from S to S' in $\underline{B}$ and inserts S' into O. The algorithm terminates when O becomes empty.

DD-based branch-and-bound was originally proposed for binary DDs (BDDs) [283], where each node has at most two outgoing edges [35]. Gillard, Schaus, and Coppé [171] generalized the algorithm

Algorithm 7 DD-based branch-and-bound for the DP formulation in Equation (2.10).

$O \leftarrow \{S^0\}$
$g(S^0) \leftarrow 0$, $\sigma(S^0) \leftarrow \langle\rangle$, $u \leftarrow \infty$, $\overline{\sigma} \leftarrow$ NULL
while $O \neq \emptyset$ **do**
  Let $S \in O$
  $O \leftarrow O \setminus \{S\}$
  Create a restricted DD $\overline{B}$ whose root is S
  Let σ^t be a shortest path from S to the terminal in $\overline{B}$ and its cost be $u(S)$
  if $g(S) + u(S) < u$ **then**
    $u \leftarrow g(S) + u(S)$, $\overline{\sigma} \leftarrow \langle \sigma(S); \sigma^t \rangle$
  if $\overline{B}$ is not exact **then**
    Create a relaxed DD $\underline{B}$ whose root is S
    Let the shortest path cost from S to the terminal in $\underline{B}$ be $l(S)$
    if $g(S) + l(S) < u$ **then**
      for all S' in an exact cut set of $\underline{B}$ **do**
        Let $\sigma(S, S')$ be a shortest path from S to S' in $\underline{B}$ and $g(S, S')$ be its cost
        $g(S') \leftarrow g(S) + g(S, S')$, $\sigma(S') \leftarrow \langle \sigma(S); \sigma(S, S') \rangle$
        $O \leftarrow O \cup \{S'\}$
return $\overline{\sigma}$

to multi-valued DDs (MDDs) [406], where a node may have more than two outgoing edges. DD-based branch-and-bound has been improved by using dual bounds to remove unnecessary nodes from relaxed and restricted DDs [169, 87] and dominance detection [88, 89].

2.2.9 Heuristic Search

DD-based branch-and-bound finds a shortest path in the exact DD to solve the problem. This approach can be viewed as state space search in AI, which solves a problem by finding a path in the state transition graph, an implicitly defined graph by the problem [372]. Similarly to DP, in state space search, a problem is defined by states and actions, and a state is transformed into another by applying an action. A solution is a sequence of actions that transforms the initial state into a goal state. In our DP notation, decisions correspond to actions, and a base state corresponds to a goal state. The state transition graph is a directed graph where nodes are states and edges are actions. A solution corresponds to a path from the initial state to a goal state in the state transition graph. Since state transition graphs may contain cycles and are not necessarily partitioned into layers in general, DDs are special cases of state transition graphs.

Heuristic search is a widely used approach for state space search [335, 116, 372]. Similarly to labeling algorithms, a heuristic search algorithm tries to find a path from the initial state to a goal state by iteratively extending a path. In doing so, it uses a heuristic function h, which estimates the path cost from a state S to a goal state by the heuristic value $h(S)$. Assuming the additive cost structure (Equation (2.10)) and $v_S = 0$ for a base state (goal state), we present generic pseudo-code for heuristic search in Algorithm 8. The algorithm maintains a set of states to search in the open list O, initialized with $\{S^0\}$. In addition, to prevent searching already searched states, the algorithm maintains another set G, initialized with $\{S^0\}$. The algorithm also records the g-value $g(S)$, the cost of the shortest path from S^0 to S found so far. In each step, the algorithm expands one state S from O: remove S from O and inserts state $S[\![\tau]\!]$ into O and G for each $\tau \in \mathcal{T}(S)$ if it has not been

Algorithm 8 Heuristic search for the DP formulation in Equation (2.10).

$O, G \leftarrow \{S^0\}$
while $O \neq \emptyset$ **do**
  Let $S \in O$
  $O \leftarrow O \setminus \{S\}$
  if S is a goal state **then return** the path from S^0 to S
  for all $\tau \in \mathcal{T}(S)$ **do**
    if $S[\![\tau]\!] \notin G$ or $g(S) + w_\tau(S) < g(S[\![\tau]\!])$ **then**
      $g(S[\![\tau]\!]) \leftarrow g(S) + w_\tau(S)$
      Compute $h(S[\![\tau]\!])$
      $O \leftarrow O \cup \{S[\![\tau]\!]\}$, $G \leftarrow G \cup \{S[\![\tau]\!]\}$
return no solution

generated with an equal or better g-value. The algorithm selects a state to expand based on the heuristic values, and the strategy depends on the concrete heuristic search algorithm. For example, A* [190] selects a state S that minimizes the f-value, $f(S) = g(S) + h(S)$, in O. The algorithm terminates when it finds a goal state or O becomes empty.

A heuristic function is typically obtained by ignoring some aspects of the original problem and solving the resulting problem, which is called relaxation or abstraction [115, 372]. This procedure is similar to state space relaxation in DP as discussed by Holte and Fan [214]. A heuristic function h is admissible if $h(S)$ always returns a lower bound on the optimal path cost from S to a goal state. With an admissible heuristic function, if the edge weights are nonnegative, the solution found by A* is guaranteed to be a shortest path [190]. Nau, Kumar, and Kanal [325] pointed out that A* with an admissible heuristic function can be considered a branch-and-bound algorithm: the f-value $f(S) = g(S) + h(S)$ is a lower bound on the shortest path cost from the initial state to a goal state via state S, and A* is guaranteed to never expand S if $f(S)$ is greater than the optimal solution cost. Ibaraki [225] and Kumar and Kanal [263] also proposed theoretical frameworks unifying heuristic search, branch-and-bound, and DP.

The idea to use heuristic search for DP dates back to the work by Martelli and Montanari [304]. They applied heuristic search to DP with more general cost structures than the additive cost structure by considering functionally weighted graphs. In such a graph, the weight of an edge is defined as a monotonically non-decreasing function instead of a constant, and the cost of a path is computed by recursively applying the functions along the path. They focused on theoretical analysis and did not use the resulting algorithm to solve problems in practice. Theoretical connections between DP and finding a path in a graph were also investigated by subsequent work [67, 175].

2.3 Model-Based Paradigms

In a model-based paradigm, a user formulates a problem as a mathematical model and then solves it using a general-purpose solver. We review three model-based paradigms by which DIDP is inspired: mathematical programming, constraint programming, and domain-independent AI planning. For each paradigm, we present the problem formulation, introduce general-purpose solvers and their algorithms, review modeling frameworks, and discuss the connections to DP.

2.3.1 Mathematical Programming

In mathematical programming, a problem is formulated as the minimization or maximization of an objective function of decision variables under constraints. A value assignment to the decision variables is a feasible solution if it satisfies the constraints. Assuming minimization, we represent a mathematical programming model as

$$\min\ f(x) \tag{2.14}$$

$$\text{s.t. } g_i(x) \leq 0 \qquad i = 1, ..., m \tag{2.15}$$

$$x \in \mathbb{R}^n \tag{2.16}$$

where x is an n-dimensional vector of the decision variables, f is the objective function, and g_i for $i = 1, ..., m$ are functions representing the constraints. We particularly focus on mixed-integer programming (MIP), where the domains of the decision variables are continuous values or integers, and the objective function and constraints are linear [440]. We represent a MIP model as

$$\min\ c'x \tag{2.17}$$

$$\text{s.t. } Ax \leq b \tag{2.18}$$

$$x \in \mathbb{R}^{n-k} \times \mathbb{Z}^k \tag{2.19}$$

where $c, b \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, c' is the transposition of c, and k is the number of integer decision variables. If all decision variables are continuous values, i.e., $k = 0$, the model is called a linear programming (LP) model. While solving MIP is NP-hard [240], LP can be solved in polynomial time [251].

Modeling Languages for Mathematical Programming

To formulate mathematical programming models, declarative modeling languages and libraries in programming languages such as C++, Java, Python, and Julia are used. While general-purpose mathematical programming solvers typically provide specific modeling interfaces, they can also be used with solver-independent modeling interfaces. With such interfaces, a user can easily use different solvers with the same model. For example, AIMMS [43], MPL [261], AMPL [139], GAMS [56], and CMPL[3] are solver-independent declarative modeling languages specifically designed for mathematical programming. PuLP,[4] Python-MIP,[5] and Pyomo [191, 59] in Python and JuMP [301] in Julia are solver-independent modeling libraries. MPS [230] is a solver-independent data format for LP and MIP, which is used in MIPLIP 2017, a benchmark library for MIP [173].

Mathematical Programming Solvers

General-purpose mathematical programming solvers are being actively developed. For example, commercial mathematical programming solvers such as Gurobi [188], CPLEX,[6] Xpress,[7] COPT

[3] https://github.com/coin-or/Cmpl
[4] https://coin-or.github.io/pulp/
[5] https://www.python-mip.com/
[6] https://www.ibm.com/products/ilog-cplex-optimization-studio
[7] https://www.fico.com/en/products/fico-xpress-optimization

[158], and MindOPT,[8] and open-source solvers such as SCIP [3], CBC,[9] GLPK,[10] and HIGHS [221] can be used for mathematical programming models including MIP. These solvers use branch-and-bound tree search [279] as a basic algorithm to solve MIP.[11] Starting from the original model, the algorithm solves the linear relaxation of a MIP model, which is the LP model created by relaxing the domain of the integer decision variables to continuous values. If the solution for the linear relaxation is feasible for the MIP model, the problem is solved. Otherwise, the algorithm branches, i.e., it creates subproblems by introducing new constraints. For example, if a decision variable $x_i \in \mathbb{Z}$ takes a continuous value in $(0, 1)$ in the solution for the linear relaxation, we can create one MIP model with an additional constraint $x_i \leq 0$ and another with $x_i \geq 1$. Thus, the algorithm constructs a search tree, where nodes are MIP models, the root is the original model, each node is connected to its subproblems, and leaves are models optimally solved without branching. The algorithm uses bounds to avoid solving subproblems. The optimal objective value of a leaf node can be used as a primal bound. For each node, the optimal objective value of the linear relaxation is used as a dual bound: if it meets or exceeds the primal bound, we do not need to solve that subproblem. The algorithm terminates when there are no subproblems to solve.

While the optimal objective values of different MIP models for the same problem should be the same, the optimal objective values of their linear relaxations can be different. For example, valid inequalities can be used to obtain better dual bounds [439]. A valid inequality is a constraint that is satisfied by all feasible solutions of the MIP model but may not be satisfied by some feasible solutions of the linear relaxation. Such constraints are redundant, unnecessary to model and solve the problem in theory, but may improve the solving performance.

Connections Between Mathematical Programming and Dynamic Programming

We review three connections between mathematical programming and DP: DP-based MIP formulations, column generation, and hybridization of DD-based branch-and-bound and MIP.

DP-Based MIP Formulations In DP-based MIP formulations, a problem or its substructure is formulated as DP, and such a formulation is embedded in a MIP model. Thus, DP is conceptually used to capture the structure of the problem, but DP algorithms are not used. Eppen and Martin [124] proposed a technique to reformulate a MIP model for a lot-sizing problem by encoding particular subsets of constraints as DP to find a shortest path in a directed acyclic graph. Bergman and Cire [32] took a similar approach, but they combined MIP, DP, and DDs. In their framework, the exact BDD for the DP formulation of a problem is decomposed into multiple BDDs, and the MIP model is formulated as finding a set of paths in the decomposed BDDs. Similarly, Lozano and Smith [300] decomposed a MIP model for a stochastic programming problem, where each subproblem corresponds to the shortest path problem in a BDD. Bergman and Cire [33] proposed solving a nonlinear optimization problem by formulating it as DP and then converting it into a MIP model. They also obtained objective bounds by aggregating states using an approach similar to a relaxed DD.

[8] `https://opt.alibabacloud.com/doc/latest/en/html/index.html`
[9] `https://www.coin-or.org/Cbc/cbcuserguide.html`
[10] `https://www.gnu.org/software/glpk/glpk.html`
[11] While the source codes of the commercial solvers are unavailable, their user manuals mention that branch-and-bound is used.

Column Generation DP algorithms are used in column generation, where an LP model is divided into a master problem and pricing subproblems [101]. In each iteration, a restricted master problem, which contains only a subset of decision variables in the master problem, is solved. Then, the pricing subproblems are solved to check the optimality of the solution for the original problem and generate decision variables to be added to the restricted master problem if the solution is not optimal. In some routing problems, where the pricing subproblems are formulated as variants of SPPRC, labeling algorithms with state space relaxation and/or bounding were used to solve the subproblems [103, 16, 299, 298, 373].

Hybridization of DD-Based Branch-and-Bound and MIP González et al. [182] hybridized DD-based branch-and-bound with MIP. In their algorithm, when an exact cut set is obtained from a relaxed DD, a node in the set is processed either by DD or MIP. In the former case, restricted and relaxed DDs will be constructed just as in the original DD-based branch-and-bound (see Section 2.2.8). In the latter case, a MIP model for the subproblem represented by the DD node is solved by a MIP solver to directly obtain an optimal solution. Which methods, DD or MIP, to use for a given node is decided by a strategy learned with supervised machine learning.

2.3.2 Constraint Programming

Constraint programming (CP) [369] is a model-based paradigm to solve constraint satisfaction problems (CSPs) and constraint optimization problems (COPs). In a CSP, decision variables $X = \{x_1, ..., x_n\}$, their domains $D = \{D_1, ..., D_n\}$, and constraints $C = \{C_1, ..., C_m\}$ are given. Each decision variable x_i is associated with its domain D_i and must be assigned a value in D_i. We assume that each domain is a finite set of values.[12] A constraint $C_j \subseteq D_{j1} \times ... \times D_{jl}$ is a relation defined over a subset of decision variables $Y_j = \{y_{j1}, ..., y_{jl}\} \subseteq X$, where D_{jk} is the domain of y_{jk}. A solution for the CSP is a value assignment $(d_1, ..., d_n)$ to the decision variables satisfying the constraints, i.e., $d_i \in D_i$ for each $x_i \in X$ and $(d_{j1}, ..., d_{jl}) \in C_j$ for each $C_j \in C$, where d_{jk} is the value of y_{jk}. In a COP, an objective function of the decision variables f is defined, and an optimal solution minimizes or maximizes $f(x)$. CSPs and COPs are NP-hard in general since the NP-complete boolean satisfiability problem (SAT) [240] is a special case of CSPs.

In CP, a CSP or COP is formulated as a CP model. While constraints are relations in the CSP definition, they are not necessarily explicitly defined in a CP model. Constraints can be described by expressions, such as equations and inequalities, using decision variables. For example, given two decision variables x_i and x_j with $D_i = D_j = \{0, 1\}$, an inequality $x_i \neq x_j$ can be used as a constraint instead of a binary relation $\{(0, 1), (1, 0)\}$. Moreover, global constraints [432] are used to represent specific substructures. For example, the AllDifferent global constraint [280] takes an arbitrary number of variables, and AllDifferent$(y_1, ..., y_l)$ represents that all values of $y_1, ..., y_l$ must be different, i.e., $y_i \neq y_j$ for $i \neq j$. Using global constraints can make a CP model simpler and easier to understand. Furthermore, CP solvers may exploit the specific combinatorial substructure explicitly represented by global constraints.

[12] CSPs with infinite domains are also studied in the literature, e.g., by Bodirsky [45].

Modeling Languages for Constraint Programming

While some CP solvers provide modeling libraries in programming languages, there are modeling languages designed for CP such as MiniZinc [326], XCSP^3 [51], ECLiPSe [8], Essence [147], OPL [431], and Comet [431]. There are also solver-independent modeling libraries in Python such as Numberjack [196] and PyCSP^3 [282]. With solver-independent modeling languages, the CP community has been organizing competitions to evaluate general-purpose CP solvers using the same CP models. The MiniZinc Challenge,[13] which uses MiniZinc, has been annually held since 2008. XCSP competitions,[14] which use XCSP^3, have also been held since 2017.

Constraint Programming Solvers

There are a number of open-source CP solvers such as CP-SAT [337], Choco-solver [346], Gecode,[15] Chuffed,[16] MiniCP [311], OscaR [330], Minion [161], JaCoP [262], Mistral [195], ACE [281], Yuck [44], and SeaPearl [69]. There are also commercial CP solvers such IBM ILOG CP Optimizer [277] and iZplus.[17] In addition, SCIP [3] is designed to solve constraint integer programming, a hybridization of CP and MIP. In CP solvers, constraint propagation and tree search are commonly used techniques [41, 429]. After introducing them, we also mention constraint-based local search [431], a framework closely related to CP and integrated into some CP solvers such as OscaR and Yuck.

Constraint Propagation In constraint propagation [41], algorithms called constraint propagators remove values from the domains of the decision variables based on constraints. For example, a constraint propagator may enforce a pair of decision variables to satisfy arc consistency [302]. Given two decision variables x_i and x_j and a set of constraints $C' \subseteq C$ defined over them, i.e., $C_k \subseteq D_i \times D_j$ for $C_k \in C'$, x_i is arc consistent with x_j if for each $d_i \in D_i$, there exists $d_j \in D_j$ such that (d_i, d_j) satisfies all constraints in C', i.e., $(d_i, d_j) \in \bigcap_{C_k \in C'} C_k$. If x_i is arc inconsistent with x_j due to $d_i \in D_i$, we can remove d_i from D_i. Once d_i is removed from D_i, another decision variable may become arc inconsistent with x_i, and some values may be removed from the domain. Therefore, constraint propagation repeatedly performs the procedure after removing values from the domains possibly using multiple constraint propagators. In practice, implementations of constraint propagators depend on CP solvers. Typically, specific constraint propagators are designed for global constraints to exploit the combinatorial substructures represented by them [432]. Although constraint propagation by itself may find a solution or prove that there is no solution, there is no guarantee in general. In CP solvers, constraint propagation is usually combined with tree search, which has the guarantee of completeness.

Tree Search In tree search for CP, each node corresponds to a CSP, and branching creates subproblems by introducing additional constraints [429]. Typically, one decision variable is selected by a mechanism called a variable ordering heuristic, and constraints on the variable are added. For example, a tree search algorithm may create one subproblem with $x_i = d$ and another with $x_i \neq d$

[13] https://www.minizinc.org/challenge.html
[14] https://www.xcsp.org/competitions/
[15] https://www.gecode.org/
[16] https://github.com/chuffed/chuffed
[17] http://www.constraint.org/en/izc_download/

or multiple subproblems where x_i is assigned different values in D_i. In these branching strategies, the values assigned to the variable are selected by a mechanism called a value ordering heuristic. For each subproblem, the domains can be reduced by constraint propagation. When there is no value to assign a decision variable, the subproblem does not have a solution, so the algorithm needs to search different nodes. When all decision variables are assigned values satisfying the constraints, a solution for the original CSP is found. A COP can be solved by repeatedly solving CSPs: when a solution is found with the objective value u, the algorithm solves a new CSP with an additional constraint that the solution cost is better than u [429]. Commonly, the search does not restart but simply continues with a new bound on the objective value.

Constraint-Based Local Search While each node in tree search corresponds to a partial value assignment, constraint-based local search (CBLS) [431] uses local search [218], which searches in the space of complete value assignments. Local search does not have a guarantee of completeness, but it is usually efficient in solving large-scale problems. To solve a CSP, starting from a complete value assignment that does not satisfy the constraints, local search investigates a set of value assignments called the neighborhood. For example, the neighborhood can be a set of value assignments created by swapping two values in the current assignment. If a solution is not found, local search repeats this procedure from a different value assignment such as one in the neighborhood. In CBLS, how to create the neighborhood is specified with the CP model by a user.

Connections Between Constraint Programming and Dynamic Programming

We review four types of connections between CP and DP: DP-based CP formulations, constraint propagation with DP, caching subproblems in tree search, and bucket elimination.

DP-Based CP Formulations Cappart et al. [62] used DP as a unified modeling framework for CP and reinforcement learning. In their approach, a CP model based on a DP formulation is used, where each decision variable in the CP model corresponds to each stage in the DP formulation, and the value assigned to the variable corresponds to the decision in the stage. The DP formulation is also interpreted as an MDP, though there is no stochasticity, and a policy for the MDP is learned by reinforcement learning. The learned policy, which ranks decisions given a state, is used as a value ordering heuristic in tree search. While this framework was originally implemented with an existing solver, a dedicated CP solver, SeaPearl [69], was developed later. In contrast to requiring a user to explicitly formulate a DP, Lin, Meng, and Li [293] proposed a method to automatically reformulate a given CP model to a DP-based model by detecting specific substructures.

Constraint Propagation with DP Trick [424] proposed a constraint propagator using DP for a global constraint based on the knapsack problem. Constraint propagators based on MDDs were also developed in the CP community [7, 210, 81]. In these approaches, constraint propagators use relaxed MDDs to represent a set of value assignments for a CSP. Cire and van Hoeve [81] associated such relaxed MDDs with state space relaxation in DP. Gentzel, Michel, and van Hoeve [162] proposed HADDOCK, a state-based modeling language for MDDs, with which a user can define MDDs in a CP model to use MDD-based constraint propagators.

Caching Subproblems in Tree Search Smith [403] proposed caching, which stores subproblem in memory during tree search and avoids searching the same subproblem multiple times, similarly to DP. Chu, Garcia de la Banda, and Stuckey [77] extended this approach to dominance detection.

Bucket Elimination Dechter [97] proposed bucket elimination, an algorithmic framework generalizing DP, as a unified approach to different problems including CSPs, SAT, and probabilistic reasoning. A bucket elimination algorithm processes functions defined over variables. It uses a bucket for each variable. The buckets are ordered, and each bucket stores functions defined over variables associated with that bucket or later buckets. The algorithm eliminates a bucket one by one: from the functions in the bucket, it generates new functions that no longer depend on the variable associated with the bucket. Such functions are inserted into the remaining buckets and will be processed later without considering the variable of the eliminated bucket. Thus, the algorithm can be viewed as reusing previously computed results as in DP algorithms.

In the bucket elimination algorithm for a CSP by Dechter [97], each decision variable is associated with a bucket, and constraints defined over variables associated with the bucket or later buckets are stored in the bucket. In each step, the algorithm eliminates the bucket associated with a decision variable x_i. The algorithm takes the join of all constraints in the bucket: given a constraint C_j defined over a subset of decision variables $Y_j \subseteq X$ and another constraint C_k over $Y_k \subseteq X$, the join of C_j and C_k is a set of value assignments to $Y_j \cup Y_k$ satisfying C_j and C_k. The algorithm projects out the variable x_i associated with the bucket from the join: creating a set of value assignment R to $(Y_k \cup Y_j) \setminus \{x_i\}$ by removing the value of x_i from each value assignment in the join and eliminating duplicates. Then, R is inserted into the next bucket as a constraint. In subsequent buckets, the variable x_i is no longer considered since some value can be assigned to x_i as long as R is satisfied. If constraints are not satisfied in a bucket, there is no solution. Otherwise, after eliminating all buckets, a solution can be constructed by selecting one value for each variable from the values remaining after the join operation performed when the associated bucket was eliminated. Hooker [216] also proposed a similar DP algorithm to solve a CSP.

2.3.3 Domain-Independent AI Planning

The objective of AI planning is to find a sequence of actions to achieve a goal from a given initial state [165]. While there are multiple classes of AI planning problems, in this section, we mainly focus on classical planning, where a state is represented by discrete variables, and an action deterministically transforms one state into another. The STRIPS formalism of classical planning [137] is represented by a tuple $\langle V_P, \mathit{Init}, \mathit{Goal}, \mathit{Ops}\rangle$, where V_P is a set of state variables, *Init* is the initial state, *Goal* is a set of goal conditions, and *Ops* is a set of operators (or equivalently, actions).[18] A state maps each state variable to a boolean value in $\{\top, \bot\}$. An operator is represented by $\langle \mathit{Pre}, \mathit{Eff}\rangle$, where *Pre* is a set of preconditions, and *Eff* is a set of effects. A precondition in *Pre* or a goal condition in *Goal* is written as $v = \top$ for some $v \in V_p$ and is satisfied by a state if it maps v to $\top$. An effect in *Eff* assigns a value to a variable, written as $v \leftarrow \top$ or $v \leftarrow \bot$. An operator can be used in a state if all of its preconditions are satisfied. When an operator is used, a state is transformed into another, where state variables mentioned in the effects are changed accordingly, and other state variables are

[18] Here, we use 'operator' to make our terminology consistent with the definition of numeric planning by Helmert [202], which will be introduced in Chapter 3.

not changed. A solution for the STRIPS planning problem is to find a plan, a sequence of operators that transforms *Init* to a goal state, which satisfies all conditions in *Goal*. The problem of finding a plan is called satisficing planning. The problem of optimizing the cost of a plan is called optimal planning, where each operator has a nonnegative cost, and the cost of a plan is the sum of the operator costs. Satisficing classical planning is PSPACE-complete [58], and thus optimal classical planning is PSPACE-hard.

STRIPS is not the only formalism for classical planning. For example, SAS+ [13] is a classical planning formalism where the domain of a state variable is a finite set, whose cardinality can be more than two. Therefore, SAS+ can explicitly represent a set of conditions that do not simultaneously hold. Some classical planners transform a STRIPS representation of a planning problem to SAS+ as preprocessing [203].

More general classes of AI planning problems than classical planning have also been studied. For example, in numeric planning [140], a state is represented by numeric state variables, whose domain is rational numbers. A precondition of an action or a goal condition can be a condition on numeric state variables, e.g., inequalities over numeric state variables. In temporal planning [140], each action is associated with time duration, and multiple actions can possibly be executed in parallel. In probabilistic planning [444, 375], the effects of an action are stochastic, which can be represented by MDPs. There are yet more classes such as non-deterministic planning [92, 80], hierarchical planning [224], and multi-agent planning [420].

Modeling Languages for Domain-Independent AI Planning

In domain-independent AI planning, planning problems are represented by the same modeling language, and domain-independent AI planners solve any problem formulated in the language. The standard modeling language for AI planning is Planning Domain Definition Language (PDDL) [164]. Since our modeling language for DIDP is inspired by PDDL, we explain it with an example focusing on a subset of features used for STRIPS. We also discuss the design philosophy of PDDL comparing it with MIP and CP. Finally, we will mention extensions of PDDL to non-classical planning.

Planning Domain Definition Language (PDDL) PDDL is a declarative language having a similar syntax to LISP. PDDL uses a domain file and problem file to specify one planning problem instance. A domain file contains definitions that can be used by multiple instances of the planning problem. A problem file contains information specific to the particular problem instance. We show example domain and problem files for TSP in Figure 2.2. The model is equivalent to the DP formulation in Equation (2.9), but we do not model the travel time for the sake of simplicity. In practice, we can model the travel time as the cost of an action in PDDL. In a domain file, object types, predicates, and actions are defined. Predicates (at ?loc - customer), (visited ?loc - customer), and (unvisited ?loc - customer) take an object with the type customer as an argument. Concrete objects for the object types are defined in a problem file. In our example, four customers, depot, c1, c2, and c3 are defined.

A state in classical planning is represented as a set of propositions, predicates whose arguments are assigned concrete objects of the object types. In our example, the initial state is explicitly given as a set of propositions (at depot), (visited depot), (unvisited c1), (unvisited c2), and (unvisited c3) in the problem file. The propositions included in a state correspond to state variables whose values

```
(define (domain tsp)
    (:requirements :strips :typing)
    (:types customer)
    (:predicates (at ?loc - customer) (visited ?loc - customer) (unvisited ?loc - customer))
    (:action visit
        :parameters (?from ?to - customer)
        :precondition (and (at ?from) (unvisited ?to))
        :effect (and (at ?to) (visited ?to) (not (unvisited ?to)) (not (at ?from)))
    )
)
```

(a) Domain file.

```
(define (problem tsp-example)
    (:domain tsp)
    (:objects depot c1 c2 c3 - customer)
    (:init (at depot) (visited depot) (unvisited c1) (unvisited c2) (unvisited c3))
    (:goal (and (visited c1) (visited c2) (visited c3)))
)
```

(b) Problem file.

Figure 2.2: Example PDDL domain and problem files for TSP.

are $\top$ in the STRIPS formalism. An action visit is defined with two parameters, ?from and ?to, which have the object type customer. Similarly to predicates, one grounded action is defined for each combination of concrete objects. In this way, we need only one definition for multiple actions having the same structure and can reuse the domain file for different instances with possibly different numbers of objects. The preconditions of the action are predicates that must be included in a state, e.g., (at ?from) and (unvisited ?to). The effects of actions are predicates added to or removed from a state. In our example, (at ?to) and (visited ?to) are added and (unvisited ?to) and (at ?from) are removed. The goal is a set of propositions that must be included in a state, defined in the problem file.

While PDDL is a declarative modeling language, modeling libraries for AI planning such as Tarski [143] and the Unified-Planning Library[19] have been also developed. With these libraries, a user can formulate a planning problem in Python and convert it to PDDL or related languages.

Design Philosophy of PDDL PDDL was originally developed for the First International Planning Competition (IPC) held in 1998 [308] to compare different AI planners using the same input. The design philosophy of PDDL is 'physics, not advice' (McDermott [308], p.37):

> every piece of a representation would be a necessary part of the specification of what actions were possible and what their effects are. All traces of "hints" to a planning system would be eliminated.

In contrast, in MIP and CP, a user has more freedom in modeling. In these paradigms, a user can include information unnecessary to define a problem, such as valid inequalities in MIP, which potentially improves the solving performance. Moreover, in CP, a user can select different representations to describe the same constraints, e.g., defining a set of inequalities or using a global constraint. Nevertheless, these approaches are still declarative model-based paradigms: redundant information or structured representations of constraints are declarative, and a user does not program particular solving algorithms in the model. The design philosophy of PDDL is justified for evaluating the performance of domain-independent AI planners to solve planning problems. However, when used

[19] `https://github.com/aiplan4eu/unified-planning`

as declarative problem-solving technologies for combinatorial optimization, the approaches of MIP and CP, where a user can investigate different models to improve the solving performance, can have a significant advantage.

We note that there were some attempts to increase the expressive power of PDDL such as Functional STRIPS [159, 142] and Planning Modulo Theories [186]. Closely related to AI planning, Hernádvölgyi, Holte, and Walsh [207] proposed PSVN, a language to define a state space search problem. PSVN automatically generates a heuristic function for heuristic search based on its state representation using a fixed length vector of discrete values. However, none of these approaches were designed for combinatorial optimization.

Extensions of PDDL and Related Langauges Multiple extensions of PDDL or modeling languages inspired by PDDL were developed for different classes of AI planning problems, most of which were motivated by IPCs.[20] The original PDDL is for classical planning. PDDL2.1 [140] is an extension of PDDL for numeric and temporal planning, developed for the third IPC in 2002. PDDL2.1 was later extended to PDDL2.2 [113] and PDDL3 [163] for the fourth and fifth IPCs in 2004 and 2006. PPDDL [444] is an extension of PDDL2.1 for probabilistic planning developed for the fourth IPC. There are other extensions of PDDL such as HDDL for hierarchical planning [224], MA-PDDL for multi-agent planning [420], and PDDL+ to handle external processes and events. In contrast to extensions of PDDL, Relational Dynamic Influence Diagram Language (RDDL) [375] is designed for MDPs from scratch and has been used for the probabilistic planning tracks of the IPCs since 2011.

Domain-Independent AI Planners

We describe two approaches for domain-independent AI planners: heuristic search and compilation-based planners.

Heuristic Search Planners Many planners that participated in the classical planning tracks of the past IPCs are based on heuristic search [48, 213, 354, 295, 154, 428, 419]. In these planners, heuristic search solves a planning problem by finding a path from the initial state to a goal state in the state transition graph, where nodes correspond to states, and edges correspond to actions. The planners automatically obtain heuristic functions from PDDL representations. For example, the FF heuristic [213] uses the length of a plan for the delete relaxation [48] of the planning problem as a heuristic value. In the delete relaxation, delete effects, i.e., effects that assign $\bot$ to state variables in STRIPS, are ignored. The delete relaxation was extended to more general classes of planning problems such as numeric planning [212, 379, 380] and temporal planning [86, 85]. Abstraction is another approach to obtain heuristic functions, typically used in optimal planning [112, 193, 242, 204, 386]. An abstraction heuristic creates a relaxed problem by mapping multiple states to the same abstract state and obtains the heuristic value by solving the relaxed problem. For example, a pattern database heuristic [112, 193] maps a state to an abstract state by ignoring a subset of state variables. Then, it computes the optimal plan cost from each abstract state to an abstract goal state and stores the results in a table. During search, the heuristic maps a state to an abstract state and uses the optimal plan cost as the heuristic value, which is obtained by a table lookup.

[20] `https://www.icaps-conference.org/competitions/`

In terms of software, the Fast Downward planning system [203], which is based on heuristic search, was used by many classical planners that participated in IPCs [154, 428, 419]. In addition, it was extended to numeric planning [6] and temporal planning [128]. The Lightweight Automated Planning Toolkit (LAPKT) [351], another AI planning framework based on heuristic search, was also used by a winning classical planner in IPC 2018 [295]. All of these systems are open-source.

Compilation-Based Planners A competitive approach to heuristic search is to compile a planning problem into another problem and use an off-the-shelf solver. For example, SAT was used for classical planning [243, 247, 244, 245, 362, 366, 73, 220, 360, 223], and satisfiability modulo theories (SMT) was used for numeric planning [381, 286, 63] and temporal planning [361, 333]. From mathematical programming, MIP was used for classical planning [246, 438, 430] and numeric planning [340, 265], and mixed-integer nonlinear programming was used for probabilistic planning [172]. CP was used for classical planning [91, 106, 297, 20, 185, 166, 12] and temporal planning [435, 156].

Connections Between Domain-Independent AI Planning and DP

There is a clear connection between AI planning and DP: both approaches are based on state-based representations. Indeed, heuristic search can be used for both AI planning and DP as we described in Section 2.2.9. In this section, we discuss two additional topics: DP in heuristic functions and DP for probabilistic planning.

DP in Heuristic Functions Some heuristic functions for AI planning are related to DP. For example, the max heuristic [48], the additive heuristic [48], and the h^m heuristic [194] for classical planning are based on Bellman-equations to compute approximated costs to achieve goal conditions. These heuristic functions are computed by algorithms based on the Bellman-Ford algorithm or Dijkstra's algorithm [48, 296, 250]. As mentioned in Section 2.2.9, Holte and Fan [214] discussed the similarity between abstraction heuristic functions in AI planning [193, 204, 386] and state space relaxation in DP [76, 1, 144, 46, 357, 355, 183, 16, 17, 365]. Similar to abstraction, Castro et al. [66] developed heuristic functions using relaxed DDs.

DP for Probabilistic Planning Since probabilistic planning is formulated as an MDP, DP algorithms such as value iteration and policy iteration can be used as mentioned in Sections 2.2.1 and 2.2.2. Indeed, planners based on DP algorithms such as real-time dynamic programming (RTDP) [21], stochastic planning using decision diagrams (SPUDD) [211], and symbolic dynamic programming (SDP) [52, 376] participated in the probabilistic planning tracks of IPCs. In particular, SDP is specifically designed for first-order MDPs, where states and actions are defined by predicates and objects as in PDDL. The number of states in such an MDP increases as the number of objects increases, which makes computing the value function impractical. Instead of considering each state using grounded predicates and actions, SDP aggregates multiple states that satisfy the same set of conditions. Then, SDP learns the value function of aggregated states using ungrounded predicates and actions.

2.4 Model-Based Dynamic Programming

While some prior studies used DP as a model-based framework, they are insufficient to be a practical model-based DP paradigm for combinatorial optimization in some aspects. We review four research directions studied by previous work: theoretical formalisms for DP, DP with logic programming, model-based DP software, and DD solvers.

2.4.1 Theoretical Formalisms for Dynamic Programming

Some problem-independent formalisms for DP have been developed, but they were studied mainly for theoretical purposes and were not actual modeling languages. In their seminal work, Karp and Held [241] introduced sequential decision process (sdp), a problem-independent formalism for DP based on a finite state automaton. However, their main purpose is to use DP algorithms without DP modeling: while they used sdp to develop DP algorithms problem-independently, they noted that "In many applications, however, the state-transition representation is not the most natural form of problem statement" and thus introduced discrete decision process, a formalism to describe a combinatorial optimization problem, from which sdp is derived. This line of research was further investigated by subsequent work [201, 228, 227, 229, 226, 225, 263, 305]. In particular, Kumar and Kanal [263] proposed the composite decision process, a theoretical formalism based on context-free grammar, as a unified framework for DP, heuristic search, and branch-and-bound.

2.4.2 Dynamic Programming with Logic Programming

A technique called tabling [42, 415] in logic programming [260] can be viewed as declarative DP modeling and solving, but it is not designed for combinatorial optimization. In logic programming, a program describes specifications of computational results using logical formulas and rules. For example, a program can specify that a predicate reachable(X, Y) is satisfied by X and Y if there exists Z such that reachable(X, Z) and reachable(Z, Y). If logical formulas reachable(a, b) and reachable(b, c) are defined in the program, the program can reason that reachable(a, c) holds. Since such reasoning requires recursively evaluating the predicate, possibly multiple times with the same arguments, tabling stores the evaluation results in memory and reuses them to speed up reasoning. In particular, Dyna is a declarative programming language for DP based on logic programming and tabling, designed for natural language processing and machine learning [121, 437]. In functional programming, similarly to tabling, Norvig [327] developed a technique to automate memoization in LISP. While these approaches were not for optimization, some researchers combined tabling with branch-and-bound (Section 2.2.6). For example, Puchinger and Stuckey [348] extended Prolog so that a user can define bounds on the objective function with a recursive DP formulation. Picat [449], a logic programming language hybridized with MIP, CP, AI planning, SAT, and SMT, implements algorithms for AI planning by combining tabling and branch-and-bound. However, all these approaches still cannot model dominance (Section 2.2.5) since tabling can reuse the evaluation result only when given arguments exactly match previously evaluated arguments.

2.4.3 Model-Based Dynamic Programming Software

Software frameworks for model-based DP were developed by previous work, but they were impractical for combinatorial optimization or designed for different application fields. DP2PNSolver [288] takes a DP model coded in a Java-style language, gDPS, as input and compiles it to program code, e.g., Java code. It first transforms the DP model into a graphical representation to identify the topological order of states and then generates a program to perform bottom-up computation (Section 2.2.3) following the order. DP2PNSolver does not use dominance nor bounding, which can be inefficient for combinatorial optimization. Algebraic dynamic programming (ADP) [167] is a software framework to formulate a DP model using context-free grammar. It was designed for bioinformatics and was originally limited to DP models where a state is described by a string. While ADP has been extended to describe a state using data structures such as a set, it is still focused on bioinformatics applications [454]. For optimal control, general-purpose DP solvers were developed as libraries in MATLAB [411, 314].

2.4.4 Decision Diagram Solvers

In existing work, the approach closest to a general-purpose DP solver for combinatorial optimization is ddo [171], a general-purpose solver using DD-based branch-and-bound (Section 2.2.8). However, modeling in ddo is specific to DD-based branch-and-bound, and thus it is a DD solver rather than a DP solver. Ddo provides modeling interfaces as libraries in Rust and Python. In either of the modeling libraries, the user formulates a DP model by defining classes (or structs in Rust) satisfying particular interfaces defined in the library. In addition, ddo requires a user to provide a merging operator to construct relaxed DDs and a ranking operator to select states to merge in relaxed DDs and states to remove in restricted DDs. These operators are specific to DD-based branch-and-bound. In contrast, existing model-based paradigms such as MIP and CP do not require information more than necessary to define a problem, while their general-purpose solvers can exploit it if provided. Thus, ddo is closer to the approach of CBLS, which requires a user to provide information specific to the solving algorithm.

2.5 Summary

We have reviewed the literature related to domain-independent dynamic programming (DIDP) from three perspectives: methodologies for dynamic programming (DP), model-based paradigms, and model-based DP. We introduced DP algorithms to solve problems defined with a Bellman equation. While basic algorithms only depend on the Bellman equation, some algorithms speed up the solving process by exploiting additional information: dominance, bounds, merging operators, and heuristic functions. We also discussed existing model-based paradigms, focusing on mixed-integer programming (MIP), constraint programming (CP), and domain-independent artificial intelligence (AI) planning. Modeling in MIP and CP is based on decision variables and constraints, which is fundamentally different from DP, while there are some connections between these paradigms and DP. In contrast, domain-independent AI planning is similar to DP in its state-based modeling, but unlike MIP and CP, its modeling language is not designed to include redundant information, such as dominance and bounds. We reviewed existing model-based DP approaches, but they are not

designed for combinatorial optimization or have restrictions in their modeling capabilities. Overall, existing approaches are insufficient to be a practical model-based DP paradigm for combinatorial optimization.

Chapter 3

Modeling Formalism and Language

In this chapter, we introduce Dynamic Programming Description Language (DyPDL), a modeling formalism for domain-independent dynamic programming (DIDP). For software implementations, we propose YAML-DyPDL, a practical modeling language for DyPDL. DyPDL is based on a state transition system, inspired by artificial intelligence (AI) planning formalisms such as STRIPS [137] and SAS+ [13], but we also follow the operations research (OR) approach of incorporating implied and redundant information in a model. In particular, differently from AI planning languages such as Planning Domain Definition Language (PDDL) [164], YAML-DyPDL is designed to allow a user to incorporate such information. This design is motivated by existing problem-specific DP algorithms in OR, which exploit redundant information to achieve better performance [109, 152, 153, 357, 355]. YAML-DyPDL provides problem- and solver-independent features to declaratively incorporate such knowledge into a DyPDL model. We demonstrate that dynamic programming (DP) models including redundant information can be formulated in DyPDL in practice using eleven combinatorial optimization problem classes.

In Section 3.1, we define DyPDL and present its theoretical properties including its computational complexity. In Section 3.2, we introduce YAML-DyPDL. In Section 3.3, we show DyPDL models for the combinatorial optimization problems studied. Finally, Section 3.4 summarizes the contributions of this chapter.

The work in this chapter is based on a paper currently under review in *Artificial Intelligence* [266], which extends two papers published in the *Proceedings of the International Conference on Automated Planning and Scheduling* [267, 270].

3.1 Dynamic Programming Description Language (DyPDL)

DyPDL is a solver-independent formalism to define a DP model. In DyPDL, a problem is described by states and transitions between states, and a solution corresponds to a sequence of transitions satisfying particular conditions. A state is a complete assignment to state variables.

Definition 1. A *state variable* is either an *element, set, or numeric variable.* An element variable v has domain $D_v = \mathbb{Z}_0^+$ (nonnegative integers). A set variable v has domain $D_v = 2^{\mathbb{Z}_0^+}$ (sets of nonnegative integers). A numeric variable v has domain $D_v = \mathbb{Q}$ (rational numbers).

Definition 2. Given a set of state variables $\mathcal{V} = \{v_1, ..., v_n\}$, a *state* is a tuple of values $S = (d_1, ..., d_n)$ where $d_i \in D_{v_i}$ for $i = 1, ..., n$. We denote the value d_i of variable v_i in state S by $S[v_i]$.

A state can be transformed into another state by changing the values of the state variables. To describe such changes, we define *expressions*: functions returning a value given a state.

Definition 3. An *element expression* e is a function that maps a state S to a nonnegative integer $e(S) \in \mathbb{Z}_0^+$. A *set expression* e is a function that maps a state S to a set $e(S) \in 2^{\mathbb{Z}_0^+}$. A *numeric expression* e is a function that maps a state S to a numeric value $e(S) \in \mathbb{Q}$. A *condition* c is a function that maps a state S to a boolean value $c(S) \in \{\bot, \top\}$. We denote $c(S) = \top$ by $S \models c$ and $c(S) = \bot$ by $S \not\models c$. For a set of conditions C, we denote $\forall c \in C, S \models c$ by $S \models C$ and $\exists c \in C, S \not\models c$ by $S \not\models C$.

With the above expressions, we formally define transitions, which transform one state into another.

Definition 4. A *transition* τ is a 4-tuple $\langle \mathsf{eff}_\tau, \mathsf{cost}_\tau, \mathsf{pre}_\tau, \mathsf{forced}_\tau \rangle$ where eff_τ is the *effect*, cost_τ is the *cost expression*, pre_τ is the set of *preconditions*, and $\mathsf{forced}_\tau \in \{\bot, \top\}$.

- The effect eff_τ maps a state variable v to an expression $\mathsf{eff}_\tau[v]$. An element, set, or numeric variable must be mapped to an element, set, or numeric expression, respectively.
- The cost expression cost_τ is a function that maps a numeric value $x \in \mathbb{Q}$ and a state S to a numeric value $\mathsf{cost}_\tau(x, S) \in \mathbb{Q}$.
- A precondition in pre_τ is a condition.
- If $\mathsf{forced}_\tau = \top$, τ is a *forced transition*.

The preconditions of a transition define when we can use it. A forced transition is a transition that must be used if its preconditions are satisfied, ignoring other transitions. Forced transitions must be totally ordered so that only one is used when a state satisfies the preconditions of multiple forced transitions. We give a formal definition of the applicability of transitions using preconditions and forced transitions.

Definition 5 (Applicability)**.** Given a set of transitions $\mathcal{T}$ and a total order $\preceq_t$ over forced transitions in $\mathcal{T}$, the set of *applicable* transitions $\mathcal{T}(S)$ in a state S is defined as follows:

$$\mathcal{T}(S) = \begin{cases} \{\tau \in \mathcal{T}_f(S) \mid \forall \tau' \in \mathcal{T}_f(S), \tau \preceq_t \tau'\} & \text{if } \mathcal{T}_f(S) \neq \emptyset \\ \{\tau \in \mathcal{T} \mid S \models \mathsf{pre}_\tau\} & \text{if } \mathcal{T}_f(S) = \emptyset. \end{cases}$$

where $\mathcal{T}_f(S) = \{\tau \in \mathcal{T} \mid S \models \mathsf{pre}_\tau \wedge \mathsf{forced}_\tau\}$.

With forced transitions, a user may formulate a more efficient model for a solver through, for example, symmetry breaking [153]. As an intuitive example for symmetry breaking using forced transitions, we use a DyPDL model for bin packing [306], which is formally defined later in Section 3.3.5. In this problem, we minimize the number of bins to pack a set of weighted items, where all bins have the same capacity. In the DyPDL model, a transition packs one item into the current bin or opens a new bin when no item can be packed. We can pack any item as the first item in a

new bin without loss of optimality, so it is sufficient to consider only one case where an arbitrary item is packed. To model this symmetry breaking, for each item, we define a forced transition that opens a new bin and packs the item with minimum index, by specifying forced transitions ordered in the ascending order of the indices of the items.

We also define the changes in the state variables using the effect of a transition.

Definition 6 (State transition)**.** Given a state S and a transition τ, the successor state $S[\![\tau]\!]$, which results from applying τ to S, is defined as $S[\![\tau]\!][v] = \mathsf{eff}_\tau[v](S)$ for each variable v. For a sequence of transitions $\sigma = \langle \sigma_1, ..., \sigma_m \rangle$, the state $S[\![\sigma]\!]$, which results from applying σ to S, is defined as $S[\![\sigma]\!] = S[\![\sigma_1]\!][\![\sigma_2]\!]...[\![\sigma_m]\!]$. If σ is an empty sequence, i.e., $\sigma = \langle\rangle$, $S[\![\sigma]\!] = S$.

We introduce notation for a sequence of transitions $\sigma = \langle \sigma_1, ..., \sigma_m \rangle$ here.

- We denote the length of σ by $|\sigma| = m$.
- We denote the i-th transition of σ by σ_i.
- We denote the first i transitions of σ by $\sigma_{:i} = \langle \sigma_1, ..., \sigma_i \rangle$ where $\sigma_{:0} = \langle\rangle$.
- We denote the last $m - i + 1$ transitions by $\sigma_{i:} = \langle \sigma_i, ..., \sigma_m \rangle$ where $\sigma_{m+1:} = \langle\rangle$.
- We denote the i-th to j-th transitions of σ by $\sigma_{i:j} = \langle \sigma_i, ..., \sigma_j \rangle$ where $1 \leq i \leq j \leq m$.
- Given a transition τ, we denote the concatenation of σ and τ by $\langle \sigma; \tau \rangle = \langle \sigma_1, ..., \sigma_m, \tau \rangle$. Similarly, $\langle \tau; \sigma \rangle = \langle \tau, \sigma_1, ..., \sigma_m \rangle$.
- Given a sequence of transitions $\sigma' = \langle \sigma'_1, ..., \sigma'_{m'} \rangle$, we denote the concatenation of σ and σ' by $\langle \sigma; \sigma' \rangle = \langle \sigma_1, ..., \sigma_m, \sigma'_1, ..., \sigma'_{m'} \rangle$. We denote the concatenation of multiple sequences σ^j for $j = 1, ..., n$ by $\langle \sigma^1; ...; \sigma^n \rangle$. We also denote the concatenation of multiple sequences with a transition τ added after the j-th sequence by $\langle \sigma^1; ...; \sigma^j; \tau; \sigma^{j+1}; ...; \sigma^n \rangle$.

While Definitions 5 and 6 are related to preconditions, forced transitions, and effects, another component, the cost expression, has not been used thus far. Since it is used to define the cost of a solution for a DyPDL model, we will use it once we define a DyPDL model in Definition 7 and an optimization problem in Definition 10.

Definition 7. A *DyPDL* model is a tuple $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$, where $\mathcal{V}$ is the set of state variables, S^0 is a state called the *target state*, $\mathcal{T}$ is the set of transitions, $\mathcal{B}$ is the set of *base cases*, and $\mathcal{C}$ is the set of *state constraints*.

- Forced transitions in $\mathcal{T}$ are totally ordered.
- A base case $B \in \mathcal{B}$ is a tuple $\langle C_B, \mathsf{cost}_B \rangle$, where C_B is a set of conditions and cost_B is a numeric expression.
- A state constraint in $\mathcal{C}$ is a condition.

A state S is a *base state* if $\exists B \in \mathcal{B}, S \models C_B \cup \mathcal{C}$.

Intuitively, the target state is the start of the state transition system, and a base state is a goal. State constraints are constraints that must be satisfied by all states. We are interested in the set of states reachable from the target state by applying transitions in the state transition system.

Definition 8 (Reachability)**.** Given a DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$, a state S, and a sequence of transitions σ with $m = |\sigma| \geq 1$, the state $S[\![\sigma]\!]$ is reachable from S with σ if the following conditions are satisfied:

- All transitions are applicable, i.e., $\sigma_{i+1} \in \mathcal{T}(S[\![\sigma_{:i}]\!])$ for $i = 0, ..., m-1$.
- All intermediate states do not satisfy a base case, i.e., $S[\![\sigma_{:i}]\!] \not\models C_B$ for $i = 0, ..., m-1$ for any base case $B \in \mathcal{B}$.
- All intermediate states satisfy the state constraints, i.e., $S[\![\sigma_{:i}]\!] \models \mathcal{C}$ for $i = 0, ..., m-1$.

Similarly, we say that $S[\![\tau]\!]$ is reachable from S with a transition τ if τ is applicable in S, S is not a base state, and S satisfies the state constraints. For states S and S', if there exists σ such that S' is reachable from S with σ, we say that S' is reachable from S. If a state S is reachable from the target state S^0, we say that S is reachable or S is a reachable state. In addition, we define that S^0 is reachable.

In the above definition, intermediate states $S[\![\sigma_{:i}]\!]$ for $i = 1, ..., m-1$ need to satisfy state constraints, but a reachable state $S[\![\sigma]\!]$ itself may violate them. However, such a reachable state is a dead end, i.e., no state is reachable from it. The reachability relation is transitive, as shown in Lemma 1.

Lemma 1 (Transitivity of reachability)**.** *Given a DyPDL model, if state S' is reachable from state S with σ, and a state S'' is reachable from S' with σ', then S'' is reachable from S with $\langle \sigma; \sigma' \rangle$.*

Proof. Since S'' is reachable from S', S' is not a base state and satisfies the state constraints. Since $S[\![\sigma]\!] = S'$, $\langle \sigma; \sigma' \rangle$ makes S transition to S'' while satisfying the conditions in Definition 8. Thus, the lemma is proved. □

With the reachability, we define the notion of a solution for a DyPDL model. Intuitively, a solution is a sequence of transitions that transforms the target state into a base state.

Definition 9. Given a DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$, let σ be a sequence of transitions. Then, σ is an *S-solution* if $S[\![\sigma]\!]$ is a base state and reachable from S with σ. For a base state S, we define an empty sequence $\langle\rangle$ to be an S-solution. An S^0-solution, i.e., a solution starting from the target state, is a *solution* for the DyPDL model. If a model has a solution, the model is *feasible*. Otherwise, the model is *infeasible*.

We consider finding a solution that maximizes or minimizes the cost over all solutions, computed from base cases and cost expressions.

Definition 10 (Optimization problems with DyPDL)**.** Given a DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$, a *minimization (maximization) problem* is to find an optimal solution for the model. An *optimal S-solution* for minimization (maximization) is an S-solution with its cost less (greater) than or equal to the cost of any S-solution. For minimization, the cost of an S-solution σ is defined recursively as

$$\mathsf{cost}_\sigma(S) = \begin{cases} \min\limits_{B \in \mathcal{B}: S \models C_B} \mathsf{cost}_B(S) & \text{if } \sigma = \langle\rangle \\ \mathsf{cost}_{\sigma_1}(\mathsf{cost}_{\sigma_{2:}}(S[\![\sigma_1]\!]), S) & \text{else.} \end{cases}$$

For maximization, we define the cost of the S-solution by replacing min with max. An *optimal solution* for the minimization (maximization) problem is an optimal S^0-solution for minimization (maximization).

Intuitively, the term x in the cost expression $\mathsf{cost}_\tau(x, S)$ represents the cost of the successor state. The cost of the solution depends on whether the problem is maximization or minimization; we take the best one according to the optimization direction when multiple base cases are satisfied. Note that an optimal solution does not necessarily exist. For example, if the set of reachable states is infinite in a maximization problem, we may construct a better solution than any given solution by adding transitions.

3.1.1 Complexity

We have defined expressions as functions of states and have not specified further details. Therefore, the complexity of an optimization problem with a DyPDL model depends on the complexity of evaluating expressions. In addition, for example, if we have an infinite number of preconditions, evaluating the applicability of a single transition may not terminate in finite time. Given these facts, we consider the complexity of a model whose definition is finite.

Definition 11. A DyPDL model is *finitely defined* if the following conditions are satisfied:

- The numbers of the state variables, transitions, base cases, and state constraints are finite.
- Each transition has a finite number of preconditions.
- Each base case has a finite number of conditions.
- All the effects, the cost expression, the preconditions of the transitions, the conditions and the costs of the base cases, and the state constraints can be evaluated in finite time.

Even with this restriction, finding a solution for a DyPDL model is an undecidable problem. We show this by reducing one of the AI planning formalisms, which is undecidable, into a DyPDL model. We define a numeric planning task and its solution in Definitions 12 and 13 following Helmert [202].

Definition 12. A *numeric planning task* is a tuple $\langle V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops}\rangle$ where V_P is a finite set of *propositional variables*, V_N is a finite set of *numeric variables*, *Init* is a state called the *initial state*, *Goal* is a finite set of *propositional and numeric conditions*, and *Ops* is a finite set of *operators*.

- A *state* is defined by a pair of functions (α, β), where $\alpha : V_P \to \{\bot, \top\}$ and $\beta : V_N \to \mathbb{Q}$.
- A propositional condition is written as $v = \top$ where $v \in V_P$. A state (α, β) satisfies it if $\alpha(v) = \top$.
- A numeric condition is written as $f(v_1, ..., v_n)$ **relop** 0 where $v_1, ..., v_n \in V_N$, f maps n numeric variables to a rational number, and **relop** $\in \{=, \neq, <, \leq, \geq, >\}$. A state (α, β) satisfies it if $f(\beta(v_1), ..., \beta(v_n))$ **relop** 0.

An operator in *Ops* is a pair $\langle \mathit{Pre}, \mathit{Eff}\rangle$, where *Pre* is a finite set of conditions (*preconditions*), and *Eff* is a finite set of *propositional and numeric effects*.

- A propositional effect is written as $v \leftarrow t$ where $v \in V_P$ and $t \in \{\bot, \top\}$.
- A numeric effect is written as $v \leftarrow f(v_1, ..., v_n)$ where $v, v_1, ..., v_n \in V_N$ and f maps n numeric variables to a rational number.

All functions that appear in numeric conditions and numeric effects are restricted to functions represented by arithmetic operators $\{+, -, \cdot, /\}$, but the divisor must be a non-zero constant.

The restriction to arithmetic operators $\{+, -, \cdot, /\}$ is motivated by PDDL 2.1 [141], where functions are described by these operators. Indeed, Definition 12 is a superset of commonly studied formalisms such as simple numeric planning [380] and linear numeric planning [212] used in the International Planning Competition 2023.[1]

Definition 13. Given a numeric planning task $\langle V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops}\rangle$, the *state transition graph* is a directed graph where nodes are states and there is an edge $((\alpha, \beta), (\alpha', \beta'))$ if there exists an operator $\langle \mathit{Pre}, \mathit{Eff}\rangle \in \mathit{Ops}$ satisfying the following conditions.

- (α, β) satisfies all conditions in *Pre*.
- $\alpha'(v) = t$ if $v \leftarrow t \in \mathit{Eff}$ and $\alpha'(v) = \alpha(v)$ otherwise.
- $\beta'(v) = f(\beta(v_1), ..., \beta(v_n))$ if $v \leftarrow f(v_1, ..., v_n) \in \mathit{Eff}$ and $\beta'(v) = \beta(v)$ otherwise.

A *solution* for the numeric planning task is a path from the initial state to a state that satisfies all goal conditions in *Goal* in the state transition graph.

Helmert [202] showed that finding a solution for the above-defined numeric planning task is undecidable. To show the undecidability of DyPDL, we reduce a numeric planning task into a DyPDL model by replacing propositional variables with a single set variable.

Theorem 1. *Finding a solution for a finitely defined DyPDL model is undecidable.*

Proof. Let $\langle V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops}\rangle$ be a numeric planning task. We compose a DyPDL model as follows:

- If $V_P \neq \emptyset$, we introduce a set variable P' in the DyPDL model. For each numeric variable $v \in V_N$ in the numeric planning task, we introduce a numeric variable v' in the DyPDL model.
- Let $(\alpha^0, \beta^0) = \mathit{Init}$. We index propositional variables in V_P using $i = 0, ..., |V_P| - 1$ and denote the i-th variable by u_i. In the target state S^0 of the DyPDL model, $S^0[P'] = \{i \in \{0, ..., |V_P| - 1\} \mid \alpha^0(u_i) = \top\}$ and $S^0[v'] = \beta^0(v)$ for each numeric variable $v \in V_N$.
- We introduce a base case $B = \langle C_B, 0\rangle$ in the DyPDL model. For each propositional condition $u_i = \top$ in *Goal*, we introduce a condition $i \in S[P']$ in C_B. For each numeric condition $f(v_1, ..., v_n)$ **relop** 0 in *Goal*, we introduce a condition $f(S[v_1], ..., S[v_n])$ **relop** 0 in C_B.
- For each operator $o = \langle \mathit{Pre}, \mathit{Eff}\rangle$, we introduce a transition $o' = \langle \mathsf{eff}_{o'}, \mathsf{cost}_{o'}, \mathsf{pre}_{o'}, \mathsf{forced}_{o'}\rangle$ with $\mathsf{cost}_{o'}(x, S) = x + 1$ and $\mathsf{forced}_{o'} = \bot$.

[1] `https://ipc2023-numeric.github.io/`

- For each propositional condition $u_i = \top$ in *Pre*, we introduce $i \in S[P']$ in $\mathsf{pre}_{o'}$. For each numeric condition $f(v_1, ..., v_n)$ **relop** 0 in *Pre*, we introduce $f(S[v'_1], ..., S[v'_n])$ **relop** 0 in $\mathsf{pre}_{o'}$.
- Let $Add = \{i \in \{0, ..., |V_P| - 1\} \mid u_i \leftarrow \top \in \mathit{Eff}\}$ and $Del = \{i \in \{0, ..., |V_P| - 1\} \mid u_i \leftarrow \bot \in \mathit{Eff}\}$. We have $\mathsf{eff}_{o'}[P'](S) = (S[P'] \setminus Del) \cup Add$. We have $\mathsf{eff}_{o'}[v'](S) = f(S[v'_1], ..., S[v'_n])$ if $v \leftarrow f(v_1, ..., v_n) \in \mathit{Eff}$ and $\mathsf{eff}_{o'}[v'](S) = S[v']$ otherwise.
- The set of state constraints is empty.

The construction of the DyPDL model is done in finite time. The numbers of propositional variables, numeric variables, goal conditions, transitions, preconditions, and effects are finite. Therefore, the DyPDL model is finitely defined.

Let $\sigma = \langle o'_1, ..., o'_m \rangle$ be a solution for the DyPDL model. Let $S^j = S^0[\![\sigma_{:j}]\!]$ for $j = 1, ..., m$. Let (α^j, β^j) be a numeric planning state such that $\alpha^j(u_i) = \top$ if $i \in S^j[P']$, $\alpha^j(u_j) = \bot$ if $i \notin S^j[P']$, and $\beta^j(v) = S^j[v']$. Note that $(\alpha^0, \beta^0) = \mathit{Init}$ satisfies the above condition by construction. We prove that the state transition graph for the numeric planning task has edge $((\alpha^j, \beta^j), (\alpha^{j+1}, \beta^{j+1}))$ for $j = 0, ..., m-1$, and (α^m, β^m) satisfies all conditions in *Goal*.

Let $o_j = (\mathit{Pre}_j, \mathit{Eff}_j)$. Since o'_j is applicable in S^j, for each propositional condition $u_i = \top$ in Pre_j, the set variable satisfies $i \in S^j[P']$. For each numeric condition $f(v_1, ..., v_n)$ **relop** 0 in Pre_j, the numeric variables satisfy $f(S^j[v'_1], ..., S^j[v'_n])$ **relop** 0. By construction, $\alpha^j(u_i) = \top$ for $i \in S^j[P']$ and $f(\beta^j(v_1), ..., \beta^j(v_n))$ **relop** 0. Therefore, (α^j, β^j) satisfies all conditions in Pre_j. Similarly, (α^m, β^m) satisfies all conditions in *Goal* since S^m satisfies all base cases.

Let $Add_j = \{i \in \{0, ..., |V_P| - 1\} \mid u_i \leftarrow \top \in \mathit{Eff}_j\}$ and $Del_j = \{i \in \{0, ..., |V_P| - 1\} \mid u_i \leftarrow \bot \in \mathit{Eff}_j\}$. By construction, $S^{j+1}[P'] = (S^j[P'] \setminus Del_j) \cup Add_j$. Therefore, for i with $u_i \leftarrow \bot \in \mathit{Eff}_j$, we have $i \notin S^{j+1}[P']$, which implies $\alpha^{j+1}(u_i) = \bot$. For i with $u_i \leftarrow \top \in \mathit{Eff}_j$, we have $i \in S^{j+1}[P']$, which implies $\alpha^{j+1}(u_i) = \top$. For other i, $i \in S^{j+1}[P']$ if $i \in S^j[P']$ and $i \notin S^{j+1}[P']$ if $i \notin S^j[P']$, which imply $\alpha^{j+1}(u_i) = \alpha^j(u_i)$. For v with $v \leftarrow f(v_1, ..., v_n) \in \mathit{Eff}_j$, we have $S^{j+1}[v'] = f(S^j[v'_1], ..., S^j[v'_n]) = f(\beta^j(v_1), ..., \beta^j(v_n))$, which implies $\beta^{j+1}(v) = f(\beta^j(v_1), ..., \beta^j(v_n))$. For other v, we have $S^{j+1}[v'] = S^j[v'] = \beta^j(v)$, which implies $\beta^{j+1}(v) = \beta^j(v)$. Therefore, edge $((\alpha^j, \beta^j), (\alpha^{j+1}, \beta^{j+1}))$ exists in the state transition graph.

Thus, by solving the DyPDL model, we can find a solution for the numeric planning task. Since the numeric planning task is undecidable, finding a solution for a DyPDL model is also undecidable. □

While we used the numeric planning formalism in Definition 12, Helmert [202] and subsequent work [174] proved undecidability for more restricted numeric planning formalisms such as subclasses of a restricted numeric planning task, where functions in numeric conditions are linear, and numeric effects increase or decrease a numeric variable only by a constant. We expect that these results can be easily applied to DyPDL since our reduction is straightforward. Previous work also investigated conditions with which a numeric planning task becomes more tractable, e.g., decidable or PSPACE-complete [202, 399, 168]. We also expect that we can generalize such conditions to DyPDL. However, in this dissertation, we consider typical properties in DP models for combinatorial optimization problems.

Definition 14. A DyPDL model is *finite* if it is finitely defined, and the set of reachable states is finite.

Definition 15. A DyPDL model is *acyclic* if any reachable state is not reachable from itself.

If a model is finite, we can enumerate all reachable states and check if there is a base state in finite time. If there is a reachable base state, and the model is acyclic, then there are a finite number of solutions, and each solution has a finite number of transitions. Therefore, by enumerating all sequences with which a state is reachable, identifying solutions, and computing their costs, we can find an optimal solution in finite time.

Theorem 2. *Given a finite and acyclic DyPDL model, the minimization or maximization problem has an optimal solution, or the model is infeasible. A problem to decide if a solution whose cost is less (greater) than a given rational number exists for minimization (maximization) is decidable.*

3.1.2 Redundant Information

In AI planning, a model typically includes only information necessary to define a problem [164, 308]. In contrast, in OR, an optimization model often includes information implied by the other parts of the model, e.g., valid inequalities in mixed-integer programming. Such information is redundant but can be useful for a solver. While DyPDL is inspired by AI planning, we also value the OR approach to solving application problems. With this motivation, we defined a forced transition in DyPDL, which can be used to break symmetries. In addition, we consider redundant information commonly used in problem-specific DP methods for combinatorial optimization problems. While such information is typically exploited algorithmically in problem-specific DP methods, in DyPDL, a user can declaratively provide it as a part of a model, and thus the domain-independence of DIDP is not broken; a user needs to define only a mathematical model of a problem.

First, we define the notion of dominance. Intuitively, one state S dominates another state S' if S always leads to an as good or a better solution, and thus S' can be ignored once we consider S. In addition to this intuition, we require that S leads to a shorter solution; otherwise, we may discard an S-solution that goes through S' if S' is dominated by S.

Definition 16 (Dominance)**.** For the minimization problem with a DyPDL model, a state S *dominates* another state S', denoted by $S' \preceq S$, iff, for any S'-solution σ', there exists an S-solution σ such that $\mathsf{cost}_\sigma(S) \leq \mathsf{cost}_{\sigma'}(S')$ and $|\sigma| \leq |\sigma'|$. For maximization, we replace $\leq$ with $\geq$ in the first inequality.

Our definition of dominance is inspired by simulation-based dominance in AI planning [418]. In that paradigm, S dominates S' only if for each applicable transition τ' in S', there exists an applicable transition τ in S such that $S[\![\tau]\!]$ dominates $S'[\![\tau']\!]$. Also, if S dominates a base state S' (a goal state in their terminology), S is also a base state. In addition to the original transitions, simulation-based dominance adds a NOOP transition, which stays in the current state. In simulation-based dominance, if we have an S'-solution, we also have S-solution with an equal number of transitions (or possibly fewer transitions if we exclude NOOP). Therefore, intuitively, Definition 16 is a generalization of simulation-based dominance; a formal discussion is out of the scope of this dissertation.

Theorem 3. *For the minimization or maximization problem with a DyPDL model, the dominance relation is a preorder, i.e., the following conditions hold.*

- $S \preceq S$ *for a state* S *(reflexivity).*
- $S'' \preceq S' \wedge S' \preceq S \to S'' \preceq S$ *for states* S*,* S'*, and* S'' *(transitivity).*

Proof. The first condition holds by Definition 16. For the second condition, for any S''-solution σ'', there exists an equal or better S'-solution σ' with $|\sigma'| \leq |\sigma''|$. There exists an equal or better S-solution σ for σ' with $|\sigma| \leq |\sigma'|$. Therefore, the cost of σ is equal to or better than σ'' with $|\sigma| \leq |\sigma''|$. □

In practice, it may be difficult to always detect if one state dominates another or not, and thus an algorithm may focus on dominance that can be easily detected. We define an approximate dominance relation to represent such a strategy.

Definition 17. For the minimization problem with a DyPDL model, an *approximate dominance relation* $\preceq_a$ is a preorder over two states such that $S' \preceq_a S \to S' \preceq S$ for reachable states S and S'.

An approximate dominance relation is sound but not complete: it always detects the dominance if two states are the same and may produce a false negative but never a false positive otherwise.

We also define a dual bound function, which underestimates (overestimates) the cost of a solution in minimization (maximization). We assume that $-\infty < x < \infty$ for any $x \in \mathbb{Q}$.

Definition 18. For the minimization problem with a DyPDL model, a function η that maps a state S to $\eta(S) \in \mathbb{Q} \cup \{\infty, -\infty\}$ is a *dual bound function* iff, for any reachable state S and any S-solution σ, $\eta(S)$ is a *dual bound* on $\mathsf{cost}_\sigma(S)$, i.e., $\eta(S) \leq \mathsf{cost}_\sigma(S)$. For maximization, we replace $\leq$ with $\geq$.

A function that always returns $-\infty$ (∞) for minimization (maximization) is trivially a dual bound function. If there exists an S-solution σ for minimization, $\eta(S) \leq \mathsf{cost}_\sigma(S) < \infty$. Otherwise, $\eta(S)$ can be any value, including ∞. Thus, if a dual bound function can detect that an S-solution does not exist, the function should return ∞ to tell a solver that there is no S-solution. For maximization, a dual bound function should return $-\infty$ in such a case.

3.1.3 The Bellman Equation for DyPDL

Typically, a DP model is succinctly represented by a recursive equation called a Bellman equation [27], and we can solve the model by solving the equation. Since DyPDL is a formalism for DP, we explicitly make a connection to a Bellman equation. For DyPDL, we can use a Bellman equation under certain conditions. First, the model should be finite and acyclic to avoid infinite recursion.[2] Second, the cost expressions need to satisfy the Principle of Optimality [27]: if σ is an optimal S-solution, $\sigma_{2:}$ is an optimal $S[\![\sigma_1]\!]$-solution.

Definition 19. Given a DyPDL model, consider any pair of a reachable state S and its successor state $S[\![\tau]\!]$ with $\tau \in \mathcal{T}(S)$ such that there exists an S-solution with $\sigma_1 = \tau$. A DyPDL model satisfies the *Principle of Optimality* for minimization if for any pair of $S[\![\tau]\!]$-solutions σ^1 and σ^2, it holds that $\mathsf{cost}_{\sigma^1}(S[\![\tau]\!]) \leq \mathsf{cost}_{\sigma^2}(S[\![\tau]\!]) \to \mathsf{cost}_{\langle \tau; \sigma^1 \rangle}(S) \leq \mathsf{cost}_{\langle \tau; \sigma^2 \rangle}(S)$. For maximization, we replace $\leq$ with $\geq$.

[2] Even if a model contains cycles, we may still use a Bellman equation by considering a fixed point of the recursive equation. However, we do not discuss it further in this dissertation.

With the Principle of Optimality, we give the Bellman equation for a DyPDL model, defining the value function V that maps a state to the optimal S-solution cost or ∞ ($-\infty$) if an S-solution does not exist in minimization (maximization).

Theorem 4 (Bellman equation)**.** *Consider the minimization problem with a finite and acyclic DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$ satisfying the Principle of Optimality. For each reachable state S, there exists an optimal S-solution with a finite number of transitions, or there does not exist an S-solution. Let V be a function of a state that returns ∞ if there does not exist an S-solution or the cost of an optimal S-solution otherwise. Then, V satisfies the following equation:*

$$V(S) = \begin{cases} \infty & \text{if } S \not\models \mathcal{C} \\ \min\limits_{B \in \mathcal{B}: S \models C_B} \mathsf{cost}_B(S) & \text{else if } \exists B \in \mathcal{B}, S \models C_B \\ \min\limits_{\tau \in \mathcal{T}(S)} \mathsf{cost}_\tau(V(S[\![\tau]\!]), S) & \text{else if } \exists \tau \in \mathcal{T}(S), V(S[\![\tau]\!]) < \infty \\ \infty & \text{else.} \end{cases} \tag{3.1}$$

For maximization, we replace ∞ with $-\infty$ and min *with* max*.*

Proof. Since the model is acyclic, we can define a partial order over reachable states where S precedes its successor state $S[\![\tau]\!]$ if $\tau \in \mathcal{T}(S)$. We can sort reachable states topologically according to this order. Since the set of reachable states is finite, there exists a state that does not precede any reachable state. Let S be such a state. Then, one of the following holds: $S \not\models \mathcal{C}$, S is a base state, or $\mathcal{T}(S) = \emptyset$ by Definition 8. If $S \not\models \mathcal{C}$, there does not exist an S-solution and $V(S) = \infty$, which is consistent with the first line of Equation (3.1). If S satisfies a base case, since the only S-solution is an empty sequence by Definition 9, $V(S) = \min_{B \in \mathcal{B}: S \models C_B} \mathsf{cost}_B(S)$, which is consistent with the second line of Equation (3.1). If S is not a base state and $\mathcal{T}(S) = \emptyset$, then there does not exist an S-solution and $V(S) = \infty$, which is consistent with the fourth line of Equation (3.1).

Assume that for each reachable state $S[\![\tau]\!]$ preceded by a reachable state S in the topological order, one of the following conditions holds:

1. There does not exist an $S[\![\tau]\!]$-solution, i.e., $V(S[\![\tau]\!]) = \infty$.
2. There exists an optimal $S[\![\tau]\!]$-solution with a finite number of transitions with cost $V(S[\![\tau]\!]) < \infty$.

If the first case holds for each $\tau \in \mathcal{T}(S)$, there does not exist an S-solution, and $V(S) = \infty$. Since $V(S[\![\tau]\!]) = \infty$ for each τ, $V(S) = \infty$ is consistent with the fourth line of Equation (3.1). The first case of the assumption also holds for S. If the second case holds for some τ, there exists an optimal $S[\![\tau]\!]$-solution with a finite number of transitions and the cost $V(S[\![\tau]\!]) < \infty$. For any S-solution σ starting with $\tau = \sigma_1$, by Definition 19,

$$\mathsf{cost}_\sigma(S) = \mathsf{cost}_\tau(\mathsf{cost}_{\sigma_{2:}}(S[\![\tau]\!]), S) \geq \mathsf{cost}_\tau(V(S[\![\tau]\!]), S),$$

and we can construct a better or equal S-solution than σ by concatenating τ and an optimal $S[\![\tau]\!]$-solution with a finite number of transitions. By considering all possible τ,

$$V(S) = \min_{\tau \in \mathcal{T}(S)} \mathsf{cost}_\tau(V(S[\![\tau]\!]), S),$$

which is consistent with the third line of Equation (3.1). The second case of the assumption also holds for S. We can prove the theorem by mathematical induction. The proof for maximization is similar. □

We can represent the redundant information, i.e., the dominance relation and a dual bound function, using the value function of the Bellman equation.

Theorem 5. *Given a finite and acyclic DyPDL model satisfying the Principle of Optimality, let V be the value function of the Bellman equation for minimization. Given an approximate dominance relation $\preceq_a$, for reachable states S and S',*

$$V(S) \leq V(S') \text{ if } S' \preceq_a S. \tag{3.2}$$

For maximization, we replace $\leq$ with $\geq$.

Proof. For reachable states S and S' with $S' \preceq_a S$, assume that there exist S- and S'-solutions. Then, $V(S)$ ($V(S')$) is the cost of an optimal S-solution (S'-solution). By Definition 16, for minimization, an optimal S-solution has a smaller cost than any S'-solution, so $V(S) \leq V(S')$. If there does not exist an S-solution, by Definition 16, there does not exist an S'-solution, so $V(S) = V(S') = \infty$. If there does not exist an S'-solution, $V(S) \leq V(S') = \infty$. The proof for the maximization problem is similar. □

Theorem 6. *Given a finite and acyclic DyPDL model satisfying the Principle of Optimality, let V be the value function of the Bellman equation for minimization. Given a dual bound function η, for a reachable state S,*

$$V(S) \geq \eta(S). \tag{3.3}$$

For maximization, we replace $\geq$ with $\leq$.

Proof. For a reachable state S, if there exists an S-solution, the cost of an optimal S-solution is $V(S)$. By Definition 18, $\eta(S)$ is a lower bound of the cost of any S-solution, so $\eta(S) \leq V(S)$. Otherwise, $\eta(S) \leq V(S) = \infty$. The proof for maximization is similar. □

3.2 YAML-DyPDL: A Practical Modeling Language

As a practical modeling language for DyPDL, we propose YAML-DyPDL on top of a data serialization language, YAML 1.2.[3] YAML-DyPDL is inspired by PDDL in AI planning [164]. However, in PDDL, while a model typically contains only information necessary to define a problem, YAML-DyPDL allows a user to explicitly model redundant information, i.e., implications of the definition. Such is the standard convention in OR and is commonly exploited in problem-specific DP algorithms for combinatorial optimization (e.g., Dumas et al. [109]). In particular, in YAML-DyPDL, a user can explicitly define an approximate dominance relation (Definition 17) and dual bound functions (Definition 18).

In the DyPDL formalism, expressions and conditions are defined as functions. In a practical implementation, the kinds of functions that can be used as expressions are defined by the syntax

[3] `https://yaml.org/`

of a modeling language. In YAML-DyPDL, for example, arithmetic operations (e.g., addition, subtraction, multiplication, and division) and set operations (e.g., adding an element, removing an element, union, intersection, and difference) using state variables can be used. We give an example of YAML-DyPDL here and present a detailed description of the syntax in Appendix A.1.

3.2.1 Example

As a running example of a combinatorial optimization problem, we use the traveling salesperson problem with time windows (TSPTW) [377]. In TSPTW, a set of customers $N = \{0, ..., n-1\}$ is given. A solution is a tour starting from the depot (index 0), visiting each customer exactly once, and returning to the depot. Visiting customer j from i incurs the travel time $c_{ij} \geq 0$. In the beginning, $t = 0$. The visit to customer i must be within a time window $[a_i, b_i]$. Upon earlier arrival, waiting until a_i is required. The objective is to minimize the total travel time. TSPTW is a generalization of the traveling salesperson problem (TSP), which does not consider time windows. Since TSP is strongly NP-hard [334], TSPTW is also strongly NP-hard. Moreover, finding a feasible solution for TSPTW is known to be NP-complete [377].

First, we present the Bellman equation of a DP model for TSPTW based on Dumas et al. [109]. In this model, a state is a tuple of variables (U, i, t), where U is the set of unvisited customers, i is the current location, and t is the current time. The value function V maps a state to the optimal cost of visiting all customers in U and returning to the depot starting from i at time t. At each step, we consider visiting one of the unvisited customers. We use c^*_{ij} as the shortest travel time from i to j ignoring time window constraints, which can be replaced with c_{ij} when the triangle inequality holds or precomputed otherwise. Also, we use the minimum travel time to customer j, $c^{\text{in}}_j = \min_{k \in N \setminus \{j\}} c_{kj}$, and the minimum travel time from j, $c^{\text{out}}_j = \min_{k \in N \setminus \{j\}} c_{jk}$. The DP model is represented by the following Bellman equation:

$$\text{compute } V(N \setminus \{0\}, 0, 0) \tag{3.4}$$

$$V(U, i, t) = \begin{cases} \infty & \text{if } \exists j \in U, t + c^*_{ij} > b_j \\ c_{i0} & \text{else if } U = \emptyset \\ \min\limits_{j \in U : t + c_{ij} \leq b_j} c_{ij} + V(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\}) & \text{else if } \exists j \in U, t + c_{ij} \leq b_j \\ \infty & \text{else} \end{cases} \tag{3.5}$$

$$V(U, i, t) \leq V(U, i, t') \qquad \text{if } t \leq t' \tag{3.6}$$

$$V(U, i, t) \geq \max \left\{ \sum_{j \in U \cup \{0\}} c^{\text{in}}_j, \sum_{j \in U \cup \{i\}} c^{\text{out}}_j \right\}. \tag{3.7}$$

Objective (3.4) declares that the optimal cost is $V(N \setminus \{0\}, 0, 0)$, the cost to visit all customers starting from the depot with $t = 0$. In Equation (3.5), the second line describes a base case of the recursion, where all customers are visited. In such a case, the cost is defined to be the travel time to return to the depot. The third line corresponds to visiting customer j from i. If j is visited, j is removed from U, the current location i is updated to j, and t is updated to the maximum of $t + c_{ij}$ and the beginning of the time window a_j. If no customer can be visited before the deadline, the state does not lead to a solution, represented by $V(U, i, t) = \infty$ in the fourth line. The first line of

Table 3.1: DyPDL representation of the DP model for TSPTW.

State variable in $\mathcal{V}$	Type	Target state S^0		
U	set	$N \setminus \{0\}$		
i	element	0		
t	numeric	0		
Transition τ in $\mathcal{T}$	eff_τ	$\mathsf{cost}_\tau(x, S)$	pre_τ	forced_τ
Visit j	$\mathsf{eff}_\tau[U](S) = S[U] \setminus \{j\}$ $\mathsf{eff}_\tau[i](S) = j$ $\mathsf{eff}_\tau[t](S) = \max\{S[t] + c_{S[i],j}, a_j\}$	$c_{S[i],j} + x$	$j \in S[U]$ $S[t] + c_{S[i],j} \leq b_j$	$\bot$
Base cases $\mathcal{B}$	$\{\langle \{S[U] = \emptyset\}, c_{S[i],0} \rangle\}$			
State constraints $\mathcal{C}$	$\{\forall j \in S[U], S[t] + c^*_{S[i],j} \leq b_j\}$			
Dominance	$S' \preceq S$ if $S[t] \leq S'[t]$			
Dual bound η	$\max \left\{ \sum_{j \in S[U] \cup \{0\}} c^{\text{in}}_j, \sum_{j \in S[U] \cup \{S[i]\}} c^{\text{out}}_j \right\}$			

Equation (3.5) states that if there exists a customer j that cannot be visited by the deadline b_j even if we use the shortest path with travel time c^*_{ij}, then the state does not lead to a solution. This equation is actually implied by the other lines. However, explicitly defining it can help a solver to detect infeasibility earlier. Inequality (3.6) defines an approximate dominance relation between two states: if the set of unvisited customers U and the current location i are the same in two states, a state having smaller t leads to a better solution. This inequality is also implied by Equation (3.5) but can be useful for a solver. Indeed, Dumas et al. [109] exploited the first line of Equation (3.5) and Inequality (3.6) in their DP method in algorithmic ways. Here, we show them declaratively with the Bellman equation. Note that the dominance implied by Inequality (3.6) satisfies Definition 16 since we need to visit the same number of customers, $|U|$, in both states (U, i, t) and (U, i, t'). Inequality (3.7) is a dual bound function of V. The cost of a state can be underestimated by the sum of the minimum travel time to visit each unvisited customer and the depot (the first term). Similarly, the cost can be underestimated by the sum of the minimum travel time from each unvisited customer and the current location (the second term). This dual bound function was not used by Dumas et al. [109].

In a DyPDL model, Objective (3.4) is defined by the target state. The first line of Equation (3.5) is defined by a state constraint, the second line is defined by a base case, and the third line is defined by transitions. In addition, the approximate dominance relation is defined by resource variables, and the dual bound function is also explicitly defined in the model. We show the description of the DP model in DyPDL in Table 3.1.

Now, we present how the DyPDL model in Table 3.1 is described by YAML-DyPDL. Following PDDL, we require two files, a domain file and a problem file, to define a DyPDL model. A domain file describes a class of problems by declaring state variables and constants and defining transitions, base cases, and dual bound functions using expressions. In contrast, a problem file describes one problem instance by defining information specific to that instance, e.g., the target state and the values of constants.

Figure 3.1 shows the domain file for the DyPDL model of TSPTW. The domain file is a map in YAML, which associates keys with values. In YAML, a key and a value are split by :. Keys and values can be maps, lists of values, strings, integers, and floating-point numbers. A list is described by multiple lines starting with -, and each value after - is an element of the list. In YAML, we can

```
cost_type: integer
objects:
  - customer
state_variables:
  - name: U
    type: set
    object: customer
  - name: i
    type: element
    object: customer
  - name: t
    type: integer
    preference: less
tables:
  - name: a
    type: integer
    args:
      - customer
  - name: b
    type: integer
    args:
      - customer
  - name: c
    type: integer
    args:
      - customer
      - customer
  - name: cstar
    type: integer
    args:
      - customer
      - customer
  - name: cin
    type: integer
    args:
      - customer
  - name: cout
    type: integer
    args:
      - customer
```

```
transitions:
  - name: visit
    parameters:
      - name: j
        object: U
    effect:
      U: (remove j U)
      i: j
      t: (max (+ t (c i j)) (a j))
    cost: (+ (c i j) cost)
    preconditions:
      - (<= (+ t (c i j)) (b j))
constraints:
  - condition: (<= (+ t (cstar i j)) (b j))
    forall:
      - name: j
        object: U
base_cases:
  - conditions:
      - (is_empty U)
    cost: (c i 0)
dual_bounds:
  - (+ (sum cin U) (cin 0))
  - (+ (sum cout U) (cout i))
reduce: min
```

Figure 3.1: YAML-DyPDL domain file for TSPTW.

also use a JSON-like syntax,[4] where a map is described as { key_1: value_1, ..., key_n: value_n }, and a list is described as [value_1, ..., value_n].

The first line defines key cost_type and its value integer, meaning that the cost of the DyPDL model is computed in integers. While the DyPDL formalism considers numeric expressions that return a rational number, in a software implementation, it is beneficial to differentiate integer and continuous values. In YAML-DyPDL, we explicitly divide numeric expressions into integer and continuous expressions.

The key objects, whose value is a list of strings, defines *object types*. In the example, the list only contains one value, customer. An object type is associated with a set of n nonnegative integers $\{0, ..., n-1\}$, where n is defined in a problem file. The customer object type represents a set of customers $N = \{0, ..., n-1\}$ in TSPTW. The object type is used later to define a set variable and constants.

The key state_variables defines state variables. The value is a list of maps describing a state variable. For each state variable, we have key name defining the name and key type defining the type, which is either element, set, integer, or continuous.

The variable U is the set variable U representing the set of unvisited customers. YAML-DyPDL requires associating a set variable with an object type. The variable U is associated with the object

[4] https://www.json.org/json-en.html

type, customer, by object: customer. Then, the domain of U is restricted to 2^N. This requirement arises from practical implementations of set variables; we want to know the maximum cardinality of a set variable to efficiently represent it in a computer program (e.g., using a fixed length of a bit vector).

The variable i is the element variable i representing the current location. YAML-DyPDL also requires associating an element variable with an object type for readability; by associating an element variable with an object type, it is easier to understand the meaning of the variable. However, the domain of the element variable is not restricted by the number of objects, n; while objects are indexed from 0 to $n-1$, a user may want to use n to represent none of them.

The variable t is the numeric variable t representing the current time. For this variable, the preference is defined by preference: less, which means that a state having smaller t dominates another state if U and i are the same. Such a variable is called a *resource variable*.

The value of the key tables is a list of maps declaring tables of constants. A table maps a tuple of objects to a constant. The table a represents the beginning of the time window a_j at customer j, so the values in the table are integers (type: integer). The concrete values are given in a problem file. The key args defines the object types associated with a table using a list. For a, one customer j is associated with the value a_j, so the list contains only one string customer. The tables b, cin, and cout are defined for the deadline b_j, the minimum travel time to a customer c_j^{in}, and the minimum travel time from a customer c_j^{out}, respectively. The table c is for c_{kj}, the travel time from customer k to j. This table maps a pair of customers to an integer value, so the value of args is a list equivalent to [customer, customer]. Similarly, the shortest travel time c_{kj}^* is represented by the table cstar.

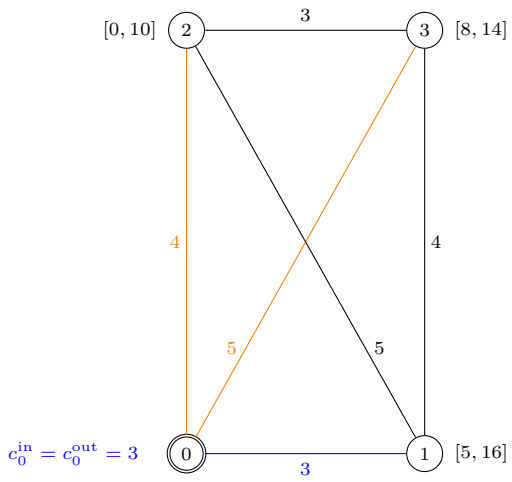
The value of the key transitions is a list of maps defining transitions. Using parameters, we can define multiple transitions in the same scheme but associated with different objects. The key name defines the name of the parameter, j, and object defines the object type. Basically, the value of the key object should be the name of the object type, e.g., customer. However, we can also use the name of a set variable. In the example, by using object: U, we state that the transition is defined for each object $j \in N$ with a precondition $j \in U$.

The key preconditions defines preconditions by using a list of conditions. In YAML-DyPDL, conditions and expressions are described by arithmetic operations in a LISP-like syntax. In the precondition of the transition in our example, (c i j) corresponds to c_{ij}, so (<= (+ t (c i j)) (b j)) corresponds to $t + c_{ij} \leq b_j$. The key effect defines the effect by using a map, whose keys are names of the state variables. For set variable U, the value is a set expression (remove j U), corresponding to $U \setminus \{j\}$. For element variable i, the value is an element expression j, corresponding to j. For integer variable t, the value is an integer expression (max (+ t (c i j)) (a j)), corresponding to $\max\{t+c_{ij}, a_j\}$. The key cost defines the cost expression (+ (c i j) cost), corresponding to $c_{ij}+x$. In the example, the cost expression must be an integer expression since the cost_type is integer. In the cost expression, we can use cost to represent the cost of the successor state (x). We can also have a key forced, whose value is boolean, indicating whether the transition is forced. We do not have it in the example, which means the transition is not forced.

The value of the key constraints is a list of state constraints. In the DyPDL model, we have $\forall j \in U, t+c_{ij}^* \leq b_j$. Similarly to the definition of transitions, we can define multiple state constraints with the same scheme associated with different objects using forall. The value of the key forall is a map defining the name of the parameter and the associated object type or set variable. The value

```
object_numbers:
  customer: 4
target:
  U: [1, 2, 3]
  i: 0
  t: 0
table_values:
  a: { 1: 5, 2: 0, 3: 8 }
  b: { 1: 16, 2: 10, 3: 14 }
  c:
    {
      [0, 1]: 3, [0, 2]: 4, [0, 3]: 5,
      [1, 0]: 3, [1, 2]: 5, [1, 3]: 4,
      [2, 0]: 4, [2, 1]: 5, [2, 3]: 3,
      [3, 0]: 5, [3, 1]: 4, [3, 2]: 3,
    }
  cstar:
    {
      [0, 1]: 3, [0, 2]: 4, [0, 3]: 5,
      [1, 0]: 3, [1, 2]: 5, [1, 3]: 4,
      [2, 0]: 4, [2, 1]: 5, [2, 3]: 3,
      [3, 0]: 5, [3, 1]: 4, [3, 2]: 3,
    }
  cin: { 0: 3, 1: 3, 2: 3, 3: 3 }
  cout: { 0: 3, 1: 3, 2: 3, 3: 3 }
```

(a) YAML-DyPDL problem file.



(b) Visualization of the instance. Each circle represents a customer, the time window is presented beside each customer, and the travel time from one customer to another is presented beside the edge between them. We use undirected edges since the travel time is symmetric. To visualize the computation of $c_0^{\text{in}} = c_0^{\text{out}}$, among the edges connected to the depot (0), we highlight the one with the minimum travel time in blue and others in orange.

Figure 3.2: Example TSPTW instance.

of the key condition is a string describing the condition, (<= (+ t (cstar i j)) (b j)), which uses the parameter j.

The value of the key base_cases is a list of maps defining base cases. Each map has two keys, conditions and cost. The value of the key conditions is a list of conditions, and the value of the key cost is a numeric expression (must be an integer expression in the example since cost_type is integer). The condition (is_empty U) corresponds to $U = \emptyset$, and the cost (c i 0) corresponds to c_{i0}.

The value of the key dual_bounds is a list of numeric expressions describing dual bound functions. In the example, we use (+ (sum cin U) (cin 0)) and (+ (sum cout U) (cout i)) corresponding to $\sum_{j \in U} c_j^{\text{in}} + c_0^{\text{in}} = \sum_{j \in U \cup \{0\}} c_j^{\text{in}}$ and $\sum_{j \in U} c_j^{\text{out}} + c_i^{\text{out}} = \sum_{j \in U \cup \{i\}} c_j^{\text{out}}$, respectively. Since cost_type is integer, they are integer expressions. The value of the key reduce is min, which means that we want to solve the minimization problem with the model.

In Figure 3.2, we present an example TSPTW instance with a YAML-DyPDL problem file with a visualization. In the problem file, the value of object_numbers is a map defining the number of objects for each object type. The value of target is a map defining the values of the state variables in the target state. For the set variable U, a list of nonnegative integers is used to define a set of elements in the set. The value of table_values is a map defining the values of the constants in the tables. For a, b, cin, and cout, a key is the index of an object, and a value is an integer. For c and cstar, a key is a list of the indices of objects.

3.2.2 Complexity

In Section 3.1, we showed that finding a solution for a DyPDL model is undecidable in general by reducing a numeric planning task to a DyPDL model. YAML-DyPDL has several restrictions compared to Definition 7. A set variable is associated with an object type, restricting its domain to a subset of a given finite set. In addition, expressions are limited by the syntax in Figure A.1 of Appendix A.1. However, these restrictions do not prevent the reduction.

Theorem 7. *Finding a solution for a finitely defined DyPDL model is undecidable even with the following restrictions.*

- *The domain of each set variable v is restricted to 2^{N_v} where $N_v = \{0, ..., n_v - 1\}$, and n_v is a positive integer.*
- *Numeric expressions and element expressions are functions represented by arithmetic operations $\{+, -, \cdot, /\}$.*
- *Set expressions are functions constructed by a set of constants, set variables, and the intersection, union, and difference of two set expressions.*
- *A condition compares two numeric expressions, compares two element expressions, or checks if a set expression is a subset of another set expression.*

Proof. We can follow the proof of Theorem 1 even with the restrictions. Since the number of propositional variables in the set V_P in a numeric planning task is finite, we can use $n_{P'} = |V_P|$ for the set variable P' representing propositional variables. Arithmetic operations $\{+, -, \cdot, /\}$ are sufficient for numeric expressions by Definition 12. Similarly, if we consider a condition $i \in S[P']$, which checks if i is included in a set variable P', as $\{i\} \subseteq S[P']$, the last two restrictions do not prevent the compilation of the numeric planning task to the DyPDL model. □

With the above reduction, we can use a system to solve a YAML-DyPDL model for the numeric planning formalism in Theorem 1.

3.2.3 DIDPPy: A Python Interface for DyPDL

In addition to YAML-DyPDL, we also provide a library for DyPDL that can be used from a programming language. Such a library is beneficial because a user can use the syntax of the programming language, and it can be easily integrated with other software in that language. We develop DIDPPy,[5] a Python library that has the same features as YAML-DyPDL: tables of constants can be defined and accessed by using element and set expressions as indices; set and element variables are associated with object types; resource variables and dual bound functions can be defined. DIDPPy also includes interfaces of solvers developed in the later chapters, so a user can define a model and give it to a solver in a Python program. We present an example Python program code defining the DyPDL model for TSPTW, which is equivalent to the one described in Table 3.1 and Figures 3.1 and 3.2a.

[5] `https://didppy.rtfd.io`

```python
import didppy as dp

model = dp.Model(maximize=False)

n = 4
a = [0, 5, 0, 8]
b = [0, 16, 10, 14]
c = model.add_int_table([[0, 3, 4, 5], [3, 0, 5, 4], [4, 5, 0, 3], [5, 4, 3, 0]])
cstar = model.add_int_table([[0, 3, 4, 5], [3, 0, 5, 4], [4, 5, 0, 3], [5, 4, 3, 0]])
cin = model.add_int_table([3, 3, 3, 3])
cout = model.add_int_table([3, 3, 3, 3])

customer = model.add_object_type(number=n)
u = model.add_set_var(object_type=customer, target=list(range(n)))
i = model.add_element_var(object_type=customer, target=0)
t = model.add_int_resource_var(target=0, less_is_better=True)

for j in range(1, n):
    visit = dp.Transition(
        name="visit {}".format(j),
        effects=[(u, u.remove(j)), (i, j), (t, dp.max(t + c[i, j], a[j]))],
        cost=c[i, j] + dp.IntExpr.state_cost(),
        preconditions=[u.contains(j), t + c[i, j] <= b[j]],
    )
    model.add_transition(visit)

for j in range(1, n):
    model.add_state_constr(~(u.contains(j)) | (t + cstar[i, j] <= b[j]))

model.add_base_case([u.is_empty()], cost=c[i, 0])

model.add_dual_bound(cin[u] + cin[0])
model.add_dual_bound(cout[u] + cout[0])
```

Figure 3.3: Python program with DIDPPy for TSPTW.

3.3 DyPDL Models for Combinatorial Optimization

To demonstrate the modeling capability, we formulate eleven combinatorial optimization problems as DyPDL models. All models are finite and acyclic and satisfy the Principle of Optimality in Definition 19. For each problem, we first present the Bellman equation to explain the model and then present the DyPDL representation as in Table 3.1 to clearly show how the model is formulated in DyPDL. The YAML-DyPDL files for the models are presented in Appendix A.2.

3.3.1 Capacitated Vehicle Routing Problem (CVRP)

In the capacitated vehicle routing problem (CVRP) [95], customers $N = \{0, ..., n-1\}$, where 0 is the depot, are given, and each customer $i \in N \setminus \{0\}$ has the demand $d_i \geq 0$. A solution is to visit each customer in $N \setminus \{0\}$ exactly once using m vehicles, which start from and return to the depot. The sum of demands of customers visited by a single vehicle must be less than or equal to the capacity q. We assume $d_i \leq q$ for each $i \in N$. Visiting customer j from i incurs the travel time $c_{ij} \geq 0$, and the objective is to minimize the total travel time. CVRP is strongly NP-hard since it is a generalization of TSP [421].

We formulate the DyPDL model based on the giant-tour representation [187]. We sequentially construct tours for the m vehicles. Let U be a set variable representing unvisited customers, i be an element variable representing the current location, l be a numeric variable representing the current load, and k be a numeric variable representing the number of used vehicles. Both l and k are resource variables where less is preferred. At each step, one customer j is visited by the current vehicle or a new vehicle. When a new vehicle is used, j is visited via the depot, l is reset, and k is increased.

Similar to TSPTW, let $c_j^{\text{in}} = \min_{k \in N \setminus \{j\}} c_{kj}$ and $c_j^{\text{out}} = \min_{k \in N \setminus \{j\}} c_{jk}$.

$$\text{compute } V(N \setminus \{0\}, 0, 0, 1) \tag{3.8}$$

$$V(U,i,l,k) = \begin{cases} \infty & \text{if } (m-k+1)q < l + \sum_{j \in U} d_j \\ c_{i0} & \text{else if } U = \emptyset \\ \min \begin{cases} \min_{j \in U : l + d_j \le q} c_{ij} + V(U \setminus \{j\}, j, l + d_j, k) \\ \min_{j \in U} c_{i0} + c_{0j} + V(U \setminus \{j\}, j, d_j, k+1) \end{cases} & \text{else if } \exists j \in U, l + d_j \le q \wedge k < m \\ \min_{j \in U : l + d_j \le q} c_{ij} + V(U \setminus \{j\}, j, l + d_j, k) & \text{else if } \exists j \in U, l + d_j \le q \\ \min_{j \in U} c_{i0} + c_{0j} + V(U \setminus \{j\}, j, d_j, k+1) & \text{else if } k < m \\ \infty & \text{else} \end{cases} \tag{3.9}$$

$$V(U,i,l,k) \le V(U,i,l',k') \qquad \text{if } l \le l' \wedge k \le k' \tag{3.10}$$

$$V(U,i,l,k) \ge \max \left\{ \sum_{j \in U \cup \{0\}} c_j^{\text{in}}, \sum_{j \in U \cup \{i\}} c_j^{\text{out}} \right\}. \tag{3.11}$$

The first line of Equation (3.9) represents a state constraint: in a state, if the sum of capacities of the remaining vehicles ($(m-k+1)q$) is less than the sum of the current load (l) and the demands of the unvisited customers ($\sum_{j \in U} d_j$), it does not lead to a solution. The second line is a base case where all customers are visited. The model has two types of transitions: directly visiting customer j, which is applicable when the current vehicle has sufficient space ($l + d_j \le q$), and visiting j with a new vehicle from the depot, which is applicable when there is an unused vehicle ($k < m$). The third line is active when both of them are possible, and the fourth and fifth lines are active when only one of them is possible. Recall that a state S dominates another state S' iff for any S'-solution, there exists an equal or better S-solution with an equal or shorter length in Definition 16. If $l \le l'$ and $k \le k'$, any (U,i,l',k')-solution is also a (U,i,l,k)-solution, so the dominance implied by Inequality (3.10) satisfies this condition. Inequality (3.11) is a dual bound function defined in the same way as Inequality (3.7) of the DyPDL model for TSPTW. We present the DyPDL representation of the model in Table 3.2.

3.3.2 Multi-Commodity Pickup and Delivery TSP (m-PDTSP)

A one-to-one multi-commodity pickup and delivery traveling salesperson problem (m-PDTSP) [208] is to pick up and deliver commodities using a single vehicle. In this problem, customers $N = \{0, ..., n-1\}$, edges $A \subseteq N \times N$, and commodities $M = \{0, ..., m-1\}$ are given. The vehicle can visit customer j directly from customer i with the travel time $c_{ij} \ge 0$ if $(i,j) \in A$. Each commodity $k \in M$ is picked up at customer $p_k \in N$ and delivered to customer $d_k \in N$. The load increases (decreases) by w_k at p_k (d_k) and must not exceed the capacity q. The vehicle starts from 0, visits each customer once, and stops at $n-1$. The objective is to minimize the total travel time. Similar to CVRP, m-PDTSP is a generalization of TSP [208], so it is strongly NP-hard.

We propose a DyPDL model based on the 1-PDTSP reduction [184] and the DP model by Castro,

Table 3.2: DyPDL representation of the DP model for CVRP.

State variable in $\mathcal{V}$	Type	Target state S^0		
U	set	$N \setminus \{0\}$		
i	element	0		
l	numeric	0		
k	numeric	1		
Transition τ in $\mathcal{T}$	eff_τ	$\mathsf{cost}_\tau(x, S)$	pre_τ	forced_τ
Visit j	$\mathsf{eff}_\tau[U](S) = S[U] \setminus \{j\}$ $\mathsf{eff}_\tau[i](S) = j$ $\mathsf{eff}_\tau[l](S) = S[l] + d_j$ $\mathsf{eff}_\tau[k](S) = S[k]$	$c_{S[i],j} + x$	$j \in S[U]$ $S[l] + d_j \leq q$	$\bot$
Visit j via the depot	$\mathsf{eff}_\tau[U](S) = S[U] \setminus \{j\}$ $\mathsf{eff}_\tau[i](S) = j$ $\mathsf{eff}_\tau[l](S) = d_j$ $\mathsf{eff}_\tau[k](S) = S[k] + 1$	$c_{S[i],0} + c_{0j} + x$	$j \in S[U]$ $S[k] < m$	$\bot$
Base cases $\mathcal{B}$	$\{\langle \{S[U] = \emptyset\}, c_{S[i],0} \rangle\}$			
State constraints $\mathcal{C}$	$\left\{(m - S[k] + 1)q \geq S[l] + \sum_{j \in S[U]} d_j\right\}$			
Dominance	$S' \preceq S$ if $S[l] \leq S'[l]$ and $S[k] \leq S'[k]$			
Dual bound η	$\max\left\{\sum_{j \in S[U] \cup \{0\}} c_j^{\text{in}}, \sum_{j \in S[U] \cup \{S[i]\}} c_j^{\text{out}}\right\}$			

Cire, and Beck [65]. In a state, a set variable U represents the set of unvisited customers, an element variable i represents the current location, and a numeric resource variable l represents the current load. The net change of the load at customer j is represented by $\delta_j = \sum_{k \in M: p_k = j} w_k - \sum_{k \in M: d_k = j} w_k$, and the customers that must be visited before j is represented by $P_j = \{p_k \mid k \in M : d_k = j\}$, both of which can be precomputed. The set of customers that can be visited next is $X(U, i, l) = \{j \in U \mid (i, j) \in A \wedge l + \delta_j \leq q \wedge P_j \cap U = \emptyset\}$. Let $c_j^{\text{in}} = \min_{k \in N: (k,j) \in A} c_{kj}$ and $c_j^{\text{out}} = \min_{k \in N: (j,k) \in A} c_{jk}$.

$$\text{compute } V(N \setminus \{0, n-1\}, 0, 0) \tag{3.12}$$

$$V(U, i, l) = \begin{cases} c_{i,n-1} & \text{if } U = \emptyset \wedge (i, n-1) \in A \\ \min_{j \in X(U,i,l)} c_{ij} + V(U \setminus \{j\}, j, l + \delta_j) & \text{else if } X(U, i, l) \neq \emptyset \\ \infty & \text{else} \end{cases} \tag{3.13}$$

$$V(U, i, l) \leq V(U, i, l') \qquad \text{if } l \leq l' \tag{3.14}$$

$$V(U, i, l) \geq \max\left\{\sum_{j \in U \cup \{n-1\}} c_j^{\text{in}}, \sum_{j \in U \cup \{i\}} c_j^{\text{out}}\right\}. \tag{3.15}$$

Similarly to CVRP, Inequalities (3.14) and (3.15) represent resource variables and a dual bound function. We present the DyPDL representation of the model in Table 3.3.

3.3.3 Orienteering Problem with Time Windows (OPTW)

In the orienteering problem with time windows (OPTW) [238], customers $N = \{0, ..., n-1\}$ are given, where 0 is the depot. Visiting customer j from i incurs the travel time $c_{ij} > 0$ while producing the integer profit $p_j \geq 0$. Each customer j can be visited only in the time window $[a_j, b_j]$, and the vehicle needs to wait until a_j upon earlier arrival. The objective is to maximize the total profit while starting from the depot at time $t = 0$ and returning to the depot by b_0. OPTW is NP-hard since it is a generalization of the orienteering problem, which is NP-hard [238, 176].

Table 3.3: DyPDL representation of the DP model for m-PDTSP.

State variable in $\mathcal{V}$	Type	Target state S^0		
U	set	$N \setminus \{0, n-1\}$		
i	element	0		
l	numeric	0		
Transition τ in $\mathcal{T}$	eff_τ	$\mathsf{cost}_\tau(x, S)$	pre_τ	forced_τ
Visit j	$\mathsf{eff}_\tau[U](S) = S[U] \setminus \{j\}$ $\mathsf{eff}_\tau[i](S) = j$ $\mathsf{eff}_\tau[l](S) = S[l] + \delta_j$	$c_{S[i],j} + x$	$j \in S[U]$ $(S[i], j) \in A$ $S[l] + \delta_j \leq q$ $P_j \cap S[U] = \emptyset$	$\bot$
Base cases $\mathcal{B}$	$\{\langle \{S[U] = \emptyset, (S[i], n-1) \in A\}, c_{S[i],n-1} \rangle\}$			
State constraints $\mathcal{C}$	$\emptyset$			
Dominance	$S' \preceq S$ if $S[l] \leq S'[l]$			
Dual bound η	$\max \left\{ \sum_{j \in S[U] \cup \{n-1\}} c_j^{\text{in}}, \sum_{j \in S[U] \cup \{S[i]\}} c_j^{\text{out}} \right\}$			

Our DyPDL model is similar to the DP model by Righini and Salani [355] but designed for DIDP with forced transitions and a dual bound function. A set variable U represents the set of customers to visit, an element variable i represents the current location, and a numeric resource variable t represents the current time, where less is preferred. We visit customers one by one using transitions. Customer j can be visited next if it can be visited and the depot can be reached by the deadline after visiting j. Let c^*_{ij} be the shortest travel time from i to j. Then, the set of customers that can be visited next is $X(U, i, t) = \{j \in U \mid t + c_{ij} \leq b_j \wedge t + c_{ij} + c^*_{j0} \leq b_0\}$. In addition, we remove a customer that can no longer be visited using a forced transition. If $t + c^*_{ij} > b_j$, then we can no longer visit customer j. If $t + c^*_{ij} + c^*_{j0} > b_0$, then we can no longer return to the depot after visiting j. Thus, the set of unvisited customers that can no longer be visited is represented by $Y(U, i, t) = \{j \in U \mid t + c^*_{ij} > b_j \vee t + c^*_{ij} + c^*_{j0} > b_0\}$. The set $Y(U, i, t)$ is not necessarily equivalent to $U \setminus X(U, i, t)$ since it is possible that j cannot be visited directly from i but can be visited via another customer when the triangle inequality does not hold.

If we take the sum of profits over $U \setminus Y(U, i, t)$, we can compute an upper bound on the value of the current state. In addition, we use another upper bound considering the remaining time limit $b_0 - t$. We consider a relaxed problem, where the travel time to customer j is always $c_j^{\text{in}} = \min_{k \in N \setminus \{j\}} c_{kj}$. This problem can be viewed as the well-known 0-1 knapsack problem [306, 249], which is to maximize the total profit of items included in a knapsack such that the total weight of the included items does not exceed the capacity of the knapsack. Each customer $j \in U \setminus Y(U, i, t)$ is an item with the profit p_j and the weight c_j^{in}, and the capacity of the knapsack is $b_0 - t - c_0^{\text{in}}$ since we need to return to the depot. Then, we can use the Dantzig upper bound [96], which sorts items in the descending order of the efficiency $e_j^{\text{in}} = p_j / c_j^{\text{in}}$ and includes as many items as possible. When an item k exceeds the remaining capacity q, then it is included fractionally, i.e., the profit is increased by $\lfloor q e_k^{\text{in}} \rfloor$. This procedural upper bound is difficult to represent efficiently with the current YAML-DyPDL due to its declarative nature. Therefore, we further relax the problem by using $\max_{j \in U \setminus Y(U,i,t)} e_j^{\text{in}}$ as the efficiencies of all items, i.e., we use $\lfloor (b_0 - t - c_0^{\text{in}}) \max_{j \in U \setminus Y(U,i,t)} e_j^{\text{in}} \rfloor$ as an upper bound. Similarly, based on $c_j^{\text{out}} = \min_{k \in N \setminus \{j\}} c_{jk}$, the minimum travel time from j, we

also use $\lfloor (b_0 - t - c_i^{\text{out}}) \max_{j \in U \setminus Y(U,i,t)} e_j^{\text{out}} \rfloor$ where $e_j^{\text{out}} = p_j / c_j^{\text{out}}$.

$$\text{compute } V(N \setminus \{0\}, 0, 0) \tag{3.16}$$

$$V(U,i,t) = \begin{cases} 0 & t + c_{i0} \leq b_0 \wedge U = \emptyset \\ V(U \setminus \{j\}, i, t) & \text{else if } \exists j \in Y(U,i,t) \\ V(U \setminus \{j\}, i, t) & \text{else if } X(U,i,t) = \emptyset \wedge \exists j \in U \\ \max\limits_{j \in X(U,i,t)} p_j + V(U \setminus \{j\}, j, \max\{t + c_{ij}, a_j\}) & \text{else if } X(U,i,t) \neq \emptyset \\ -\infty & \text{else} \end{cases} \tag{3.17}$$

$$V(U,i,t) \geq V(U,i,t') \qquad \text{if } t \leq t' \tag{3.18}$$

$$V(U,i,t) \leq \min \left\{ \sum_{j \in U \setminus Y(U,i,t)} p_j, \left\lfloor (b_0 - t - c_0^{\text{in}}) \max_{j \in U \setminus Y(U,i,t)} e_j^{\text{in}} \right\rfloor, \left\lfloor (b_0 - t - c_i^{\text{out}}) \max_{j \in U \setminus Y(U,i,t)} e_j^{\text{out}} \right\rfloor \right\}. \tag{3.19}$$

The second line of Equation (3.17) removes an arbitrary customer j in $Y(U,i,t)$, which is implemented by a forced transition. In practice, the one having the smallest index is removed. The third line also defines a forced transition to remove a customer j in U when no customer can be visited directly ($X(U,i,t) = \emptyset$); in such a case, even if $j \in U \setminus Y(U,i,t)$, i.e., $j + c_{ij}^* \leq b_j$, the shortest path to customer j is not available. The base case (the first line of Equation (3.17)) becomes active when all customers are visited or removed. This condition forces the vehicle to visit as many customers as possible. Since each transition removes one customer from U, and all customers must be removed in a base state, all (U,i,t)- and (U,i,t')-solutions have the same length. If $t \leq t'$, more customers can potentially be visited, so (U,i,t) leads to an equal or better solution than (U,i,t'). Thus, the dominance implied by Inequality (3.18) satisfies Definition 16. We present the DyPDL representation of the model in Table 3.4.

Table 3.4: DyPDL representation of the DP model for OPTW.

State variable in $\mathcal{V}$	Type	Target state S^0		
U	set	$N \setminus \{0\}$		
i	element	0		
t	numeric	0		
Transition τ in $\mathcal{T}$	eff_τ	$\mathsf{cost}_\tau(x,S)$	pre_τ	forced_τ
Remove j by time	$\mathsf{eff}_\tau[U](S) = S[U] \setminus \{j\}$ $\mathsf{eff}_\tau[i](S) = S[i]$ $\mathsf{eff}_\tau[t](S) = S[t]$	x	$j \in S[U]$ $\begin{pmatrix} S[t] + c^*_{S[i],j} > b_j \\ \vee S[t] + c^*_{S[i],j} + c^*_{j0} > b_0 \end{pmatrix}$	$\top$
Remove j by paths	$\mathsf{eff}_\tau[U](S) = S[U] \setminus \{j\}$ $\mathsf{eff}_\tau[i](S) = S[i]$ $\mathsf{eff}_\tau[t](S) = S[t]$	x	$j \in S[U]$ $\begin{pmatrix} \forall k \in S[U], \\ S[t] + c_{S[i],k} > b_k \\ \vee S[t] + c_{S[i],k} + c^*_{k0} > b_0 \end{pmatrix}$	$\top$
Visit j	$\mathsf{eff}_\tau[U](S) = S[U] \setminus \{j\}$ $\mathsf{eff}_\tau[i](S) = j$ $\mathsf{eff}_\tau[t](S) = \max\{S[t] + c_{S[i],j}, a_j\}$	$p_j + x$	$j \in S[U]$ $S[t] + c_{S[i],j} \leq b_j$ $S[t] + c_{S[i],j} + c^*_{j0} \leq b_0$	$\bot$
Base cases $\mathcal{B}$	$\{\langle \{S[t] + c_{S[i],0} \leq b_0, S[U] = \emptyset\}, 0 \rangle\}$			
State constraints $\mathcal{C}$	$\emptyset$			
Dominance	$S' \preceq S$ if $S[t] \leq S'[t]$			
Dual bound η	RHS of Inequality (3.19)			

3.3.4 Multi-Dimensional Knapsack Problem (MDKP)

The multi-dimensional knapsack problem (MDKP) [306, 249] is a generalization of the 0-1 knapsack problem. In this problem, each item $i \in \{0, ..., n-1\}$ has the integer profit $p_i \geq 0$ and m-dimensional nonnegative weights $(w_{i,0}, ..., w_{i,m-1})$, and the knapsack has the m-dimensional capacities $(q_0, ..., q_{m-1})$. In each dimension, the total weight of items included in the knapsack must not exceed the capacity. The objective is to maximize the total profit. MDKP is strongly NP-hard [155, 61].

In our DyPDL model, we decide whether to include an item one by one. An element variable i represents the index of the item considered currently, and a numeric variable r_j represents the remaining space in the j-th dimension. We can use the total profit of the remaining items as a dual bound function. In addition, we consider an upper bound similar to that of OPTW by ignoring dimensions other than j. Let $e_{kj} = p_k/w_{kj}$ be the efficiency of item k in dimension j. Then, $\lfloor r_j \max_{k=i,...,n-1} e_{kj} \rfloor$ is an upper bound on the cost of a (i, r_j)-solution. If $w_{kj} = 0$, we define $e_{kj} = \sum_{k=i,...,n-1} p_k$, i.e., the maximum additional profit achieved from (i, r_j). In such a case, $\max\{r_j, 1\} \cdot \max_{k=1,...,n-1} e_{ij}$ is still a valid upper bound.

$$\text{compute } V(0, q_0, ..., q_{m-1}) \tag{3.20}$$

$$V(i, r_0, ..., r_{m-1}) = \begin{cases} 0 & \text{if } i = n \\ \max \begin{cases} p_i + V(i+1, r_0 - w_{i,0}, ..., r_{m-1} - w_{i,m-1}) \\ V(i+1, r_0, ..., r_{m-1}) \end{cases} & \text{else if } \forall j \in M, w_{ij} \leq r_{ij} \\ V(i+1, r_0, ..., r_{m-1}) & \text{else} \end{cases} \tag{3.21}$$

$$V(i, r_0, ..., r_{m-1}) \leq \min \left\{ \sum_{k=i,...,n-1} p_k, \min_{j \in M} \left\lfloor \max\{r_j, 1\} \cdot \max_{k=i,...,n-1} e_{kj} \right\rfloor \right\} \tag{3.22}$$

where $M = \{0, ..., m-1\}$. We present the DyPDL representation of the model in Table 3.5.

Table 3.5: DyPDL representation of the DP model for MDKP.

State variable in $\mathcal{V}$	Type	Target state S^0		
i	element	0		
$\forall j \in M, r_j$	numeric	q_j		
Transition τ in $\mathcal{T}$	eff_τ	$\mathsf{cost}_\tau(x, S)$	pre_τ	forced_τ
Pack	$\mathsf{eff}_\tau[i](S) = i + 1$ $\mathsf{eff}_\tau[r_j](S) = S[r_j] - w_{S[i],j}$	$p_{S[i]} + x$	$\forall j \in M, w_{S[i],j} \leq r_{S[i],j}$	$\bot$
Ignore	$\mathsf{eff}_\tau[i](S) = i + 1$ $\mathsf{eff}_\tau[r_j](S) = S[r_j]$	x		$\bot$
Base cases $\mathcal{B}$	$\{\langle \{S[i] = n\}, 0 \rangle\}$			
State constraints $\mathcal{C}$	$\emptyset$			
Dominance	-			
Dual bound η	$\min \left\{ \begin{array}{l} \sum_{k=S[i],...,n-1} p_k \\ \min_{j \in M} \left\lfloor \max\{S[r_j], 1\} \cdot \max_{k=S[i],...,n-1} e_{kj} \right\rfloor \end{array} \right.$			

3.3.5 Bin Packing

In a bin packing problem [306], items $N = \{0, ..., n-1\}$ are given, and each item i has weight w_i. The objective is to pack items in bins with the capacity q while minimizing the number of bins. We assume $q \geq w_i$ for each $i \in N$. Bin packing is strongly NP-hard [155].

In our DyPDL model, we pack items one by one. A set variable U represents the set of unpacked items, and a numeric resource variable r represents the remaining space in the current bin, where more is preferred. In addition, we use an element resource variable k representing the number of used bins, where less is preferred. The model breaks symmetry by packing item i in the i-th or an earlier bin. Thus, $X(U, r, k) = \{i \in U \mid r \geq w_i \wedge i + 1 \geq k\}$ represents items that can be packed in the current bin. When $\forall j \in U, r < w_j$, then a new bin is opened, and any item in $Y(U, k) = \{i \in U \mid i \geq k\}$ can be packed. Here, we break symmetry by selecting an item having the minimum index using forced transitions.

For a dual bound function, we use lower bounds, LB1, LB2, and LB3, used by Johnson [236]. The first lower bound, LB1, is defined as $\lceil (\sum_{i \in U} w_i - r)/q \rceil$, which relaxes the problem by allowing splitting an item across multiple bins. The second lower bound, LB2, only considers items in $\{i \in U \mid w_i \geq q/2\}$. If $w_i > q/2$, item i cannot be packed with other items considered. If $w_i = q/2$, at most one additional item j with $w_j = q/2$ can be packed. Let $Z(U, (a, b]) = \{i \in U \mid a < w_i \leq b\}$, $Z(U, (a, b)) = \{i \in U \mid a < w_i < b\}$, and $Z(U, a) = \{i \in U \mid w_i = a\}$. The number of bins is lower bounded by $\left|Z\left(U, \left(\frac{q}{2}, q\right]\right)\right| + \left\lceil \frac{|Z(U, \frac{q}{2})|}{2} \right\rceil - \mathbb{1}\left(r \geq \frac{q}{2}\right)$, where $\mathbb{1}$ is an indicator function that returns 1 if the given condition is true and 0 otherwise. The last term considers the case when an item can be packed in the current bin. Similarly, LB3 only considers items in $Z\left(U, \left(\frac{q}{3}, q\right]\right)$, and the number of bins is lower bounded by $\left\lceil \frac{|Z(U, \frac{q}{3})|}{3} + \frac{|Z(U, (\frac{q}{3}, \frac{2q}{3}))|}{2} + \frac{2|Z(U, \frac{2q}{3})|}{3} + \left|Z\left(U, \left(\frac{2q}{3}, q\right]\right)\right| \right\rceil - \mathbb{1}\left(r \geq \frac{q}{3}\right)$.

$$\text{compute } V(N, 0, 0) \tag{3.23}$$

$$V(U, r, k) = \begin{cases} 0 & \text{if } U = \emptyset \\ 1 + V(U \setminus \{i\}, q - w_i, k + 1) & \text{else if } \exists i \in Y(U, k) \wedge \forall j \in U, r < w_j \\ \min\limits_{i \in X(U, r, k)} V(U \setminus \{i\}, r - w_i, k) & \text{else if } X(U, r, k) \neq \emptyset \\ \infty & \text{else} \end{cases} \tag{3.24}$$

$$V(U, r, k) \leq V(U, r', k') \qquad \text{if } r \geq r' \wedge k \leq k' \tag{3.25}$$

$$V(U, r, k) \geq \max \begin{cases} \lceil (\sum_{i \in U} w_i - r)/q \rceil \\ \left|Z\left(U, \left(\frac{q}{2}, q\right]\right)\right| + \left\lceil \frac{|Z(U, \frac{q}{2})|}{2} \right\rceil - \mathbb{1}\left(r \geq \frac{q}{2}\right) \\ \left\lceil \frac{|Z(U, \frac{q}{3})|}{3} + \frac{|Z(U, (\frac{q}{3}, \frac{2q}{3}))|}{2} + \frac{2|Z(U, \frac{2q}{3})|}{3} + \left|Z\left(U, \left(\frac{2q}{3}, q\right]\right)\right| \right\rceil - \mathbb{1}\left(r \geq \frac{q}{3}\right). \end{cases} \tag{3.26}$$

Since each transition packs one item, any (U, r, k)- and (U, r', k') solutions have the same length. It is easy to see that (U, r, k) leads to an equal or better solution than (U, r', k') if $r \geq r'$ and $k \leq k'$, so the dominance implied by Inequality (3.25) is valid.

We present the DyPDL representation of the model in Table 3.6. To compute LB2 and LB3, we

need to take the cardinalities of the subsets of U. Although taking the cardinality of a set is possible in YAML-DyPDL, for efficiency, we implement LB2 and LB3 by taking the sum of constants over items included in U. For LB2, we introduce tables of constants a_i and b_i, where $a_i = 1$ if $w_i > \frac{q}{2}$ and $a_i = 0$ otherwise, and $b_i = \frac{1}{2}$ if $w_i = \frac{q}{2}$ and $b_i = 0$ otherwise. Similarly, for LB3, we introduce a table of constants c_i, defined as

$$c_i = \begin{cases} 1 & \text{if } \frac{2q}{3} < w_i \leq q \\ \frac{2}{3} & \text{if } w_i = \frac{2q}{3} \\ \frac{1}{2} & \text{if } \frac{q}{3} < w_i < \frac{2q}{3} \\ \frac{1}{3} & \text{if } w_i = \frac{q}{3} \\ 0 & \text{else.} \end{cases}$$

The indicator function $\mathbb{1}$ is implemented by an 'if-then-else' expression in YAML-DyPDL, which returns one value if a given condition is satisfied and another value otherwise.

Table 3.6: DyPDL representation of the DP model for bin packing.

State variable in $\mathcal{V}$	Type	Target state S^0		
U	set	N		
r	numeric	0		
k	element	0		
Transition τ in $\mathcal{T}$	**eff_τ**	**$\mathsf{cost}_\tau(x, S)$**	**pre_τ**	**forced_τ**
Open and pack i	$\mathsf{eff}_\tau[U](S) = S[U] \setminus \{i\}$ $\mathsf{eff}_\tau[r](S) = q - w_i$ $\mathsf{eff}_\tau[k](S) = S[k] + 1$	$1 + x$	$i \in S[U]$ $i \geq S[k]$ $\forall j \in S[U], S[r] < w_j$	$\top$
Pack i	$\mathsf{eff}_\tau[U](S) = S[U] \setminus \{i\}$ $\mathsf{eff}_\tau[r](S) = S[r] - w_i$ $\mathsf{eff}_\tau[k](S) = S[k]$	x	$i \in S[U]$ $S[r] \geq w_i$ $i + 1 \geq S[k]$	$\bot$
Base cases $\mathcal{B}$	$\{\langle \{S[U] = \emptyset\}, 0 \rangle\}$			
State constraints $\mathcal{C}$	$\emptyset$			
Dominance	$S' \preceq S$ if $S[r] \geq S'[r]$ and $S[k] \leq S'[k]$			
Dual bound η	$\max \left\{ \begin{array}{l} \lceil (\sum_{i \in U} w_i - S[r])/q \rceil \\ \sum_{i \in S[U]} a_i + \left\lceil \sum_{i \in S[U]} b_i \right\rceil - \mathbb{1}\left(S[r] \geq \frac{q}{2}\right) \\ \left\lceil \sum_{i \in S[U]} c_i \right\rceil - \mathbb{1}\left(S[r] \geq \frac{q}{3}\right) \end{array} \right.$			

3.3.6 Simple Assembly Line Balancing Problem (SALBP-1)

The variant of the simple assembly line balancing problem (SALBP) called SALBP-1 [374, 22] is the same as bin packing except for precedence constraints. In SALBP-1, items are called tasks, bins are called stations, and we schedule items in stations. Stations are ordered, and each task must be scheduled in the same or later station than its predecessors $P_i \subseteq N$. SALBP-1 is strongly NP-hard since it is a generalization of bin packing [313].

We formulate a DyPDL model based on that of bin packing and inspired by a problem-specific heuristic search method for SALBP-1 [388, 319]. Due to the precedence constraint, we cannot break symmetry as we do in bin packing. Thus, we do not use an element resource variable k. Now, the set of tasks that can be scheduled in the current station is represented by $X(U, r) = \{i \in U \mid r \geq w_i \wedge P_i \cap U = \emptyset\}$. We introduce a transition to open a new station only when $X(U, r) = \emptyset$, which is called a maximum load pruning rule in the literature [233, 383]. Since bin packing is a relaxation of

SALBP-1, we can use the dual bound function for bin packing.

$$\text{compute } V(N,0) \tag{3.27}$$

$$V(U,r) = \begin{cases} 0 & \text{if } U = \emptyset \\ 1 + V(U,q) & \text{else if } X(U,r) = \emptyset \\ \min\limits_{i \in X(U,r)} V(U \setminus \{i\}, r - w_i) & \text{else} \end{cases} \tag{3.28}$$

$$V(U,r) \leq V(U,r') \qquad \text{if } r \geq r' \tag{3.29}$$

$$V(U,r) \geq \max \begin{cases} \left\lceil (\sum_{i \in U} w_i - r)/q \right\rceil \\ |Z(U,(\frac{q}{2},q])| + \left\lceil \frac{|Z(U,\frac{q}{2})|}{2} \right\rceil - \mathbb{1}\left(r \geq \frac{q}{2}\right) \\ \left\lceil \frac{|Z(U,\frac{q}{3})|}{3} + \frac{|Z(U,(\frac{q}{3},\frac{2q}{3}))|}{2} + \frac{2|Z(U,\frac{2q}{3})|}{3} + |Z(U,(\frac{2q}{3},q])| \right\rceil - \mathbb{1}\left(r \geq \frac{q}{3}\right). \end{cases} \tag{3.30}$$

The length of a (U,r)-solution is the sum of $|U|$ and the number of stations opened, which is the cost of that solution. Therefore, if $r \geq r'$, then state (U,r) leads to an equal or better and shorter solution than (U,r'), so the dominance implied by Inequality (3.29) is valid. We present the DyPDL representation of the model in Table 3.7. The transition to open a station is implemented as a forced transition since other transitions are not applicable when it is applicable.

Table 3.7: DyPDL representation of the DP model for SALBP-1.

State variable in $\mathcal{V}$	Type	Target state S^0		
U	set	N		
r	numeric	0		
Transition τ in $\mathcal{T}$	eff_τ	$\mathsf{cost}_\tau(x,S)$	pre_τ	forced_τ
Open	$\mathsf{eff}_\tau[U](S) = S[U]$ $\mathsf{eff}_\tau[r](S) = q$	$1 + x$	$\begin{pmatrix} \forall i \in S[U], \\ S[r] < w_i \\ \vee P_i \cap S[U] \neq \emptyset \end{pmatrix}$	$\top$
Schedule i	$\mathsf{eff}_\tau[U](S) = S[U] \setminus \{i\}$ $\mathsf{eff}_\tau[r](S) = S[r] - w_i$	x	$i \in S[U]$ $S[r] \geq w_i$ $P_i \cup S[U] = \emptyset$	$\bot$
Base cases $\mathcal{B}$	$\{\langle \{S[U] = \emptyset\}, 0\rangle\}$			
State constraints $\mathcal{C}$	$\emptyset$			
Dominance	$S' \preceq S$ if $S[r] \geq S'[r]$			
Dual bound η	$\max \begin{cases} \left\lceil (\sum_{i \in U} w_i - S[r])/q \right\rceil \\ \sum\limits_{i \in S[U]} a_i + \left\lceil \sum\limits_{i \in S[U]} b_i \right\rceil - \mathbb{1}\left(r \geq \frac{q}{2}\right) \\ \left\lceil \sum\limits_{i \in S[U]} c_i \right\rceil - \mathbb{1}\left(r \geq \frac{q}{3}\right) \end{cases}$			

3.3.7 Single Machine Total Weighted Tardiness ($1||\sum w_i T_i$)

In single machine scheduling to minimize the total weighted tardiness ($1||\sum w_i T_i$) [123], a set of jobs N is given, and each job $i \in N$ has the processing time p_i, the deadline d_i, and the weight w_i, all of which are nonnegative. The objective is to schedule all jobs on a machine while minimizing the sum of the weighted tardiness, $\sum_{i \in N} w_i \max\{0, C_i - d_i\}$ where C_i is the completion time of job i. This problem is strongly NP-hard [285].

We formulate a DyPDL model based on an existing DP model [198, 2], where one job is scheduled at each step. Let F be a set variable representing the set of scheduled jobs. A numeric expression

$T(i,F) = \max\{0, \sum_{j\in F} p_j + p_i - d_i\}$ represents the tardiness of i when it is scheduled after F. We introduce a set P_i, representing the set of jobs that must be scheduled before i. While it is not defined in the problem, P_i can be extracted in preprocessing using precedence theorems without losing optimality [123].

$$\text{compute } V(\emptyset) \tag{3.31}$$

$$V(F) = \begin{cases} 0 & \text{if } F = N \\ \min\limits_{i\in N\setminus F: P_i\setminus F=\emptyset} w_i T(i,F) + V(F\cup\{i\}) & \text{else} \end{cases} \tag{3.32}$$

$$V(F) \geq 0. \tag{3.33}$$

We present the DyPDL representation of the model in Table 3.8.

Table 3.8: DyPDL representation of the DP model for $1||\sum w_i T_i$.

State variable in $\mathcal{V}$	Type	Target state S^0		
F	set	$\emptyset$		
Transition τ in $\mathcal{T}$	eff_τ	$\mathsf{cost}_\tau(x,S)$	pre_τ	forced_τ
Schedule i	$\mathsf{eff}_\tau[F](S) = S[F]\cup\{i\}$	$w_i T(i,S[F]) + x$	$i \notin S[F]$ $P_i \setminus S[F] = \emptyset$	$\bot$
Base cases $\mathcal{B}$	$\{\langle\{S[F] = N\}, 0\rangle\}$			
State constraints $\mathcal{C}$	$\emptyset$			
Dominance	-			
Dual bound η	0			

3.3.8 Talent Scheduling

The talent scheduling problem [74] is to find a sequence of scenes to shoot to minimize the total cost of a film. In this problem, a set of actors A and a set of scenes N are given. In a scene $s \in N$, a set of actors $A_s \subseteq A$ plays for d_s days. An actor a incurs the cost c_s for each day they are on location. If an actor plays on days i and j, they are on location on days $i, i+1, ..., j$ even if they do not play on day $i+1$ to $j-1$. The objective is to find a sequence of scenes such that the total cost is minimized. This problem is strongly NP-hard [74].

We use the DP model proposed by Garcia de la Banda and Stuckey [152]. Let Q be a set variable representing a set of unscheduled scenes. At each step, a scene s to shoot is selected from Q. A set expression

$$L(s,Q) = A_s \cup \left(\bigcup_{s'\in Q} A_{s'} \cap \bigcup_{s'\in N\setminus Q} A_{s'} \right)$$

represents the set of actors on location when s is shot. We need to pay the cost $d_s \sum_{a\in L(s,Q)} c_a$ to shoot s.

A set expression $L(Q) = \bigcup_{s\in Q} A_s \cap \bigcup_{s\in N\setminus Q} A_s$ is the set of actors on location after shooting $N \setminus Q$. If $A_s = L(Q)$, then s should be immediately shot because all actors are already on location: a forced transition.

If there exist two scenes s_1 and s_2 in Q such that $A_{s_1} \subseteq A_{s_2}$ and $A_{s_2} \subseteq \bigcup_{s\in N\setminus Q} A_s \cup A_{s_1}$, it is known that scheduling s_2 before s_1 is always better, denoted by $s_2 \preceq s_1$. Since two scenes with the same set of actors are merged into a single scene in preprocessing without losing optimality, we

can assume that all A_s are different. With this assumption, the relationship is a partial order: it is reflexive because $A_{s_1} \subseteq A_{s_1}$ and $A_{s_1} \subseteq \bigcup_{s \in N \setminus Q} A_s \cup A_{s_1}$; it is antisymmetric because if $s_1 \preceq s_2$ and $s_2 \preceq s_1$, then $A_{s_1} \subseteq A_{s_2}$ and $A_{s_2} \subseteq A_{s_1}$, which imply $s_1 = s_2$; it is transitive because if $s_2 \preceq s_1$ and $s_3 \preceq s_2$, then $A_{s_1} \subseteq A_{s_2} \subseteq A_{s_3}$ and $A_{s_3} \subseteq \bigcup_{s \in N \setminus Q} A_s \cup A_{s_2} \subseteq \bigcup_{s \in N \setminus Q} A_s \cup A_{s_1}$, which imply $s_3 \preceq s_1$. Therefore, the set of candidate scenes to shoot next, $R(Q) = \{s_1 \in Q \mid \nexists s_2 \in Q \setminus \{s_1\}, s_2 \preceq s_1\}$, is not empty.

The cost per day to shoot s is lower bounded by $b_s = \sum_{a \in A_s} c_a$ because actors playing in s must be on location. Overall, we have the following DyPDL model.

$$\text{compute } V(N) \tag{3.34}$$

$$V(Q) = \begin{cases} 0 & \text{if } Q = \emptyset \\ d_s b_s + V(Q \setminus \{s\}) & \text{else if } \exists s \in Q, A_s = L(Q) \\ \min\limits_{s \in R(Q)} d_s \sum_{a \in L(s,Q)} c_a + V(Q \setminus \{s\}) & \text{else} \end{cases} \tag{3.35}$$

$$V(Q) \geq \sum_{s \in Q} d_s b_s. \tag{3.36}$$

We present the DyPDL representation of the model in Table 3.9.

Table 3.9: DyPDL representation of the DP model for talent scheduling.

State variable in $\mathcal{V}$	Type	Target state S^0		
Q	set	N		
Transition τ in $\mathcal{T}$	eff_τ	$\mathsf{cost}_\tau(x, S)$	pre_τ	forced_τ
Shoot i with actors on location	$\mathsf{eff}_\tau[Q](S) = S[Q] \setminus \{i\}$	$d_s b_s + x$	$i \in S[Q]$ $A_s = L(S[Q])$	$\top$
Shoot i	$\mathsf{eff}_\tau[U](S) = S[Q] \setminus \{i\}$	$d_s \sum\limits_{a \in L(s, S[Q])} c_a + x$	$i \in S[Q]$ $\forall s' \in S[Q] \setminus \{s\}, \neg s' \preceq s$	$\bot$
Base cases $\mathcal{B}$	$\{\langle \{S[Q] = \emptyset\}, 0 \rangle\}$			
State constraints $\mathcal{C}$	$\emptyset$			
Dominance	-			
Dual bound η	$\sum\limits_{s \in S[Q]} d_s b_s$			

3.3.9 Minimization of Open Stacks Problem (MOSP)

In the minimization of open stacks problem (MOSP) [446], customers C and products P are given, and each customer c orders a subset of products $P_c \subseteq P$. A solution is a sequence in which products are produced. We produce copies of a product for all customers at once. When producing product i, a stack for customer c with $i \in P_c$ is opened, and it is closed when all of P_c are produced. The objective is to minimize the maximum number of open stacks at a time. MOSP is NP-hard [294].

For MOSP, *customer search* is a state-of-the-art exact method [79]. It searches for an order of customers to close stacks, from which the order of products is determined; for each customer c, all products ordered by c and not yet produced are consecutively produced in an arbitrary order. We formulate customer search as a DyPDL model. A set variable R represents customers whose stacks are not closed, and O represents customers whose stacks have been opened. Let $N_c = \{c' \in C \mid P_c \cap P_{c'} \neq \emptyset\}$ be the set of customers that have at least one product in common with c. When

producing items for customer c, we need to open stacks for customers in $N_c \setminus O$, and stacks for customers in $O \cap R$ remain open.

$$\text{compute } V(C, \emptyset) \tag{3.37}$$

$$V(R, O) = \begin{cases} 0 & \text{if } R = \emptyset \\ \min\limits_{c \in R} \max\left\{|(O \cap R) \cup (N_c \setminus O)|, V(R \setminus \{c\}, O \cup N_c)\right\} & \text{else} \end{cases} \tag{3.38}$$

$$V(R, O) \geq 0. \tag{3.39}$$

We present the DyPDL representation of the model in Table 3.10.

Table 3.10: DyPDL representation of the DP model for MOSP.

State variable in $\mathcal{V}$	Type	Target state S^0		
R	set	C		
Q	set	$\emptyset$		
Transition τ in $\mathcal{T}$	eff_τ	$\mathsf{cost}_\tau(x, S)$	pre_τ	forced_τ
Close c	$\mathsf{eff}_\tau[R](S) = S[R] \setminus \{c\}$ $\mathsf{eff}_\tau[O](S) = S[O] \cup N_c$	$\max\{|(S[O] \cap S[R]) \cup (N_c \setminus S[O])|, x\}$	$c \in S[R]$	$\bot$
Base cases $\mathcal{B}$	$\{\langle \{S[R] = \emptyset\}, 0\rangle\}$			
State constraints $\mathcal{C}$	$\emptyset$			
Dominance	-			
Dual bound η	0			

3.3.10 Graph-Clear

In the graph-clear problem [256], an undirected graph (N, E) with the node weight a_i for $i \in N$ and the edge weight b_{ij} for $\{i, j\} \in E$ is given. In the beginning, all nodes are contaminated. In each step, one node can be made clean by sweeping it using a_i robots and blocking each edge $\{i, j\}$ using b_{ij} robots. However, while sweeping a node, an already swept node becomes contaminated if it is connected by a path of unblocked edges to a contaminated node. The optimal solution minimizes the maximum number of robots per step to make all nodes clean. This optimization problem is NP-hard since finding a solution whose cost is smaller than a given value is NP-complete [256].

Previous work [317] proved that there exists an optimal solution in which a swept node is never contaminated again. Based on this observation, the authors developed a state-based formula as the basis for MIP and CP models. We use the state-based formula directly as a DyPDL model. A set variable C represents swept nodes, and one node in $N \setminus C$ is swept at each step. We block all edges connected to c and all edges from contaminated nodes to already swept nodes. We assume that $b_{ij} = 0$ if $\{i, j\} \notin E$.

$$\text{compute } V(\emptyset) \tag{3.40}$$

$$V(C) = \begin{cases} 0 & \text{if } C = N \\ \min\limits_{c \in N \setminus C} \max\left\{a_c + \sum\limits_{i \in N} b_{ci} + \sum\limits_{i \in C} \sum\limits_{j \in (N \setminus C) \setminus \{c\}} b_{ij}, V(C \cup \{c\})\right\} & \text{else} \end{cases} \tag{3.41}$$

$$V(C) \geq 0. \tag{3.42}$$

We present the DyPDL representation of the model in Table 3.11.

Table 3.11: DyPDL representation of the DP model for graph-clear.

State variable in $\mathcal{V}$	Type	Target state S^0		
C	set	$\emptyset$		
Transition τ in $\mathcal{T}$	eff_τ	$\mathsf{cost}_\tau(x, S)$	pre_τ	forced_τ
Sweep c	$\mathsf{eff}_\tau[C](S) = S[C] \cup \{c\}$	$\max \left\{ a_c + \sum_{i \in N} b_{ci} + \sum_{i \in S[C]} \sum_{j \in (N \setminus S[C]) \setminus \{c\}} b_{ij}, x \right\}$	$c \notin S[C]$	$\bot$
Base cases $\mathcal{B}$	$\{\langle \{S[C] = N\}, 0 \rangle\}$			
State constraints $\mathcal{C}$	$\emptyset$			
Dominance	-			
Dual bound η	0			

3.4 Summary

In this chapter, we introduced Dynamic Programming Description Language (DyPDL), the solver-independent modeling formalism designed for dynamic programming (DP) models of combinatorial optimization problems, and analyzed its theoretical properties. We proved that finding a solution for a DyPDL model is undecidable in general while it is more tractable in some special cases. We also clarified the connection between DyPDL and the Bellman equation, a mathematical model commonly used in DP: a finite and acyclic DyPDL model can be represented as the Bellman equation if the Principle of Optimality is satisfied.

As a practical modeling language for DyPDL, we proposed YAML-DyPDL, which is designed to allow a user to investigate efficient optimization models. In particular, a user can specify resource variables, which define dominance between states, and dual bound functions in YAML-DyPDL. We demonstrated that DP models for eleven combinatorial optimization problem classes can be formulated in DyPDL with these features.

Chapter 4

Heuristic Search Solvers for Domain-Independent Dynamic Programming

In the previous chapter, we defined Dynamic Programming Description Language (DyPDL), the modeling formalism for domain-independent dynamic programming (DIDP), based on a state transition system. In this chapter, we develop DIDP solvers using heuristic search, which has achieved significant success over the past two decades in fields including artificial intelligence (AI) planning [48, 213, 203, 354]. Even in numeric planning [141], which is undecidable in general [202] as in the case with DyPDL, heuristic search algorithms have shown good empirical performance in practice [212, 379, 289, 380, 276, 274].

We show that heuristic search can be applied to solve DyPDL models with the guarantee of optimality under reasonable theoretical conditions. Based on this theory, we develop seven solvers using existing heuristic search algorithms and exploiting redundant information provided in a DyPDL model. Six of the seven DIDP solvers follow the standard in model-based paradigms such as mixed-integer programming (MIP) and constraint programming (CP); they are anytime, i.e., possibly provide feasible solutions and objective bounds before proving the optimality. We compare the developed DIDP solvers with commercial MIP and CP solvers and demonstrate that DIDP outperforms MIP in nine problem classes, CP also in nine problem classes, and both MIP and CP in seven.

In Section 4.1, we formally define heuristic search for DyPDL and show its theoretical properties. In Section 4.2, we introduce solvers for DyPDL using existing heuristic search algorithms. In Section 4.3, we experimentally evaluate the developed solvers. Finally, Section 4.4 summarizes the contributions of this chapter.

Similar to the previous chapter, the work in this chapter is based on the paper currently under review in *Artificial Intelligence* [266], which extends the two papers published in the *Proceedings of the International Conference on Automated Planning and Scheduling* [267, 270].

4.1 Heuristic Search for DyPDL

In this section, we present definitions of heuristic search algorithms and prove that they can be used to solve an optimization problem with a DyPDL model under certain conditions. Once we interpret the state transition system defined by a DyPDL model as a graph, it is intuitive that we can use heuristic search to solve the model. However, our setting has some differences from existing approaches, so we need to formally show that heuristic search is indeed a valid method for a certain class of DyPDL models. For example, the definition of the cost of a solution in DyPDL is more general than the sum of the weights of edges. Previous work generalized Dijkstra's algorithm [105] and A* [190] to more general settings [114], but our setting still has several differences. In addition, we consider a broader class of heuristic search algorithms including anytime algorithms, which iteratively improve a solution and exploit the approximate dominance relation in Definition 17 (Section 3.1.2, p. 39). Although previous work combined A* with dominance while preserving the optimality in AI planning [418], the definition of dominance is different from ours.

4.1.1 State Transition Graph

First, we introduce the notion of the state transition graph for a DyPDL model.

Definition 20. Given a DyPDL model, *the state transition graph* is a directed graph where nodes are reachable states (Definition 8 in Section 3.1, p. 35) and there is an edge from S to S' labeled with τ, (S, S', τ), iff S' is reachable from S with a single transition τ.

It is easy to show that a solution for a DyPDL model corresponds to a path in the state transition graph.

Theorem 8. *Given a DyPDL model, let S be a reachable state. A state S' is reachable from S with a sequence of transitions $\sigma = \langle \sigma_1, ..., \sigma_m \rangle$ iff there exists a path $\langle (S, S^1, \sigma_1), (S^1, S^2, \sigma_2), ..., (S^{m-1}, S', \sigma_m) \rangle$ in the state transition graph.*

Proof. If S' is reachable from S with σ, then by defining $S^i = S[\![\sigma_{:i}]\!]$, edges (S, S^1, σ_1), (S^1, S^2, σ_2), ..., (S^{m-1}, S', σ_m) exist in the state transition graph. If there exist a path $\langle (S, S^1, \sigma_1), (S^1, S^2, \sigma_2), ..., (S^{m-1}, S', \sigma_m) \rangle$, then S^1 is reachable from S, and S^i is reachable from S^{i-1} with σ_i by Definition 20. By Lemma 1 (Section 3.1, p. 35), S' is reachable from S with σ. □

Corollary 1. *Given a DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$, a sequence of transitions $\sigma = \langle \sigma_1, ..., \sigma_m \rangle$ is a solution for the model iff there exists a path $\langle (S^0, S^1, \sigma_1), ..., (S^{m-1}, S^m, \sigma_m) \rangle$ in the state transition graph where S^m is a base state.*

We use the term *path* to refer to both a sequence of edges in the state transition graph and a sequence of transitions as they are equivalent. Trivially, if a model is acyclic, the state transition graph is also acyclic.

4.1.2 Cost Algebras

In the minimization or maximization problem with a DyPDL model in Definition 10 (Section 3.1, p. 35), we want to find a solution that minimizes or maximizes the cost. Shortest path algorithms such as Dijkstra's algorithm [105] and A* [190] find the path minimizing the sum of the weights

associated with the edges. In DyPDL, the cost of a solution can be more general, defined by cost expressions of the transitions. Edelkamp, Jabbar, and Lafuente [114] extended the shortest path algorithms to cost-algebraic heuristic search algorithms, which can handle more general cost structures. In their framework, the cost of a path is computed by applying a binary operator to the edge weights, and the best path is determined by an operation. Unlike the conventional shortest path problem, the binary operator is not necessarily addition (+), and the operation to select the best path is not necessarily minimization. Following their approach, first, we define a monoid, which specifies properties of a binary operator.

Definition 21. Let A be a set, $\times : A \times A \to A$ be a binary operator, and $\mathbf{1} \in A$. A tuple $\langle A, \times, \mathbf{1} \rangle$ is a *monoid* if the following conditions are satisfied.

- $x \times y \in A$ for $x, y \in A$.
- $x \times (y \times z) = (x \times y) \times z$ for $x, y, z, \in A$ (associativity).
- $x \times \mathbf{1} = \mathbf{1} \times x = x$ for $x \in A$ (identity).

For example, the set of rational number is a monoid under addition, represented by $\langle \mathbb{Q}, +, 0 \rangle$.

Next, we define isotonicity, a property of a set and a binary operator with regard to comparison. With isotonicity, the order of two elements x and y are preserved after applying a binary operator with the same element z. Since minimization or maximization over rational numbers is sufficient for our use case, we restrict the set A to rational numbers, and the comparison operator to $\leq$. The original paper by Edelkamp, Jabbar, and Lafuente [114] is more general.

Definition 22 (Isotonicity)**.** Given a set $A \subseteq \mathbb{Q} \cup \{-\infty, \infty\}$ and a binary operator $\times : A \times A \to A$, A is *isotone* if $x \leq y \to x \times z \leq y \times z$ and $x \leq y \to z \times x \leq z \times y$ for $x, y, z \in A$.

For example, $\mathbb{Q}$ is isotone under addition. With a monoid and isotonicity, we define a cost algebra.

Definition 23. Let $\langle A, \times, \mathbf{1} \rangle$ be a monoid where $A \subseteq \mathbb{Q} \cup \{-\infty, \infty\}$ is isotone. The monoid $\langle A, \times, \mathbf{1} \rangle$ is a *cost algebra* if $\forall x \in A, \mathbf{1} \leq x$ for minimization or $\forall x \in A, \mathbf{1} \geq x$ for maximization.

4.1.3 Cost-Algebraic DyPDL Models

To apply cost-algebraic heuristic search, we focus on DyPDL models where cost expressions satisfy particular conditions. First, we define a monoidal DyPDL model, where cost expressions are represented by a binary operator in a monoid.

Definition 24. Let $\langle A, \times, \mathbf{1} \rangle$ be a monoid where $A \subseteq \mathbb{Q} \cup \{-\infty, \infty\}$. A DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$ is *monoidal* with $\langle A, \times, \mathbf{1} \rangle$ if the cost expression of every transition $\tau \in \mathcal{T}$ is represented as $\mathsf{cost}_\tau(x, S) = w_\tau(S) \times x$ where w_τ is a numeric expression returning a value in $A \setminus \{-\infty, \infty\}$, and the cost cost_B of each base case $B \in \mathcal{B}$ is a numeric expression returning a value in $A \setminus \{-\infty, \infty\}$.

We also define a cost-algebraic DyPDL model, which requires stricter conditions.

Definition 25. A monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$ with a monoid $\langle A, \times, \mathbf{1} \rangle$ is *cost-algebraic* if $\langle A, \times, \mathbf{1} \rangle$ is a cost algebra.

For example, the dynamic programming (DP) model for the traveling salesperson problem with time windows (TSPTW) in Section 3.2.1 is cost-algebraic with a cost algebra $\langle \mathbb{Q}_0^+, +, 0 \rangle$ since the cost expression of each transition is defined as $c_{S[i],j} + x$ with $c_{S[i],j} \geq 0$.

When a model is monoidal, we can associate a weight to each edge in the state transition graph. The weight of a path can be computed by repeatedly applying the binary operator to the weights of the edges in the path.

Definition 26. Given a monoidal DyPDL model with $\langle A, \times, \mathbf{1} \rangle$, the *weight of an edge* (S, S', τ) is $w_\tau(S)$. The *weight of a path* $\langle (S, S^1, \sigma_1), (S^1, S^2, \sigma_2), ..., (S^{m-1}, S^m, \sigma_m) \rangle$ defined by a sequence of transitions σ is

$$w_\sigma(S) = w_{\sigma_1}(S) \times w_{\sigma_2}(S^1) \times ... \times w_{\sigma_m}(S^{m-1}).$$

For an empty path $\langle\rangle$, the weight is $\mathbf{1}$.

The order of applications of the binary operator $\times$ does not matter due to the associativity. Differently from the original cost-algebraic heuristic search, the weight of a path corresponding to an S-solution may not be equal to the cost of the S-solution in Definition 10 (Section 3.1, p. 35) due to our inclusion of the cost of a base state. In the following lemma, we associate the weight of a path with the cost of a solution.

Lemma 2. *Given a monoidal DyPDL model with a monoid $\langle A, \times, \mathbf{1} \rangle$ and a state S, let σ be an S-solution. For minimization, $\mathsf{cost}_\sigma(S) = w_\sigma(S) \times \min_{B \in \mathcal{B}: S[\![\sigma]\!] \models C_B} \mathsf{cost}_B(S[\![\sigma]\!])$. For maximization, we replace* min *with* max*.*

Proof. If σ is an empty sequence, since $w_\sigma(S) = \mathbf{1}$ and $S[\![\sigma]\!] = S$,

$$w_\sigma(S) \times \min_{B \in \mathcal{B}: S[\![\sigma]\!] \models C_B} \mathsf{cost}_B(S[\![\sigma]\!]) = \min_{B \in \mathcal{B}: S \models C_B} \mathsf{cost}_B(S) = \mathsf{cost}_\sigma(S)$$

by Definition 10. Otherwise, let $\sigma = \langle \sigma_1, ..., \sigma_m \rangle$, $S^1 = S[\![\sigma_1]\!]$, and $S^{i+1} = S^i[\![\sigma_{i+1}]\!]$ for $i = 1, ..., m-1$. Following Definitions 10 and 24,

$$\mathsf{cost}_\sigma(S) = \mathsf{cost}_{\sigma_1}(\mathsf{cost}_{\sigma_{2:}}(S^1), S) = w_{\sigma_1}(S) \times \mathsf{cost}_{\sigma_{2:}}(S^1).$$

For $2 \leq i \leq m$, we get

$$\mathsf{cost}_{\sigma_{i:}}(S^{i-1}) = \mathsf{cost}_{\sigma_i}(\mathsf{cost}_{\sigma_{i+1:}}(S^i), S^{i-1}) = w_{\sigma_i}(S^{i-1}) \times \mathsf{cost}_{\sigma_{i+1:}}(S^i).$$

For $i = m+1$, $\mathsf{cost}_{\sigma_{m+1:}}(S^m) = \mathsf{cost}_{\langle\rangle}(S^m) = \min_{B \in \mathcal{B}: S^m \models C_B} \mathsf{cost}_B(S^m)$. Thus, we get the equation in the lemma by Definition 26. The proof for maximization is similar. □

We show that isotonicity is sufficient for the Principle of Optimality in Definition 19 (Section 3.1.3, p. 40). First, we prove its generalized version in Theorem 9.

Theorem 9. *Let $\langle A, \times, \mathbf{1} \rangle$ be a monoid where $A \subseteq \mathbb{Q} \cup \{-\infty, \infty\}$ and A is isotone. For the minimization problem with a monoidal DyPDL model with $\langle A, \times, \mathbf{1} \rangle$, let S' and S'' be states reachable from S with sequences of transitions σ' and σ'', respectively, with $w_{\sigma'}(S) \leq w_{\sigma''}(S)$. If there exist S'- and S''-solutions σ^1 and σ^2 with $\mathsf{cost}_{\sigma^1}(S') \leq \mathsf{cost}_{\sigma^2}(S'')$, then $\langle \sigma'; \sigma^1 \rangle$ and $\langle \sigma''; \sigma^2 \rangle$ are S-solutions with $\mathsf{cost}_{\langle \sigma'; \sigma^1 \rangle}(S) \leq \mathsf{cost}_{\langle \sigma''; \sigma^2 \rangle}(S)$. For maximization, we replace $\leq$ with $\geq$.*

Proof. The sequences of transitions $\langle \sigma'; \sigma^1 \rangle$ and $\langle \sigma''; \sigma^2 \rangle$ are S-solutions by Definition 9 (Section 7, p. 35). By Definition 26, $\mathsf{cost}_{\langle \sigma'; \sigma^1 \rangle}(S) = w_{\sigma'}(S) \times \mathsf{cost}_{\sigma^1}(S')$. Since A is isotone,

$$\mathsf{cost}_{\langle \sigma'; \sigma^1 \rangle}(S) = w_{\sigma'}(S) \times \mathsf{cost}_{\sigma^1}(S') \leq w_{\sigma''}(S) \times \mathsf{cost}_{\sigma^1}(S') \leq w_{\sigma''}(S) \times \mathsf{cost}_{\sigma^2}(S') = \mathsf{cost}_{\langle \sigma''; \sigma^2 \rangle}(S).$$

The proof for maximization is similar. □

Corollary 2. *Let $\langle A, \times, \mathbf{1} \rangle$ be a monoid where $A \subseteq \mathbb{Q} \cup \{-\infty, \infty\}$ and A is isotone. A monoidal DyPDL model with $\langle A, \times, \mathbf{1} \rangle$ satisfies the Principle of Optimality in Definition 19.*

4.1.4 Formalization of Heuristic Search for DyPDL

A heuristic search algorithm searches for a path between nodes in a graph. In particular, we focus on *unidirectional heuristic search* algorithms, which visit nodes by traversing edges from one node (the initial node) to find a path to one of the nodes satisfying particular conditions (goal nodes). In unidirectional heuristic search, given a node S, a heuristic function h returns a heuristic value (h-value) $h(S)$. Heuristic values are used in two ways: search guidance and pruning.

For search guidance, typically, the priority of a node S is computed from $h(S)$, and the node to visit next is selected based on it. For example, greedy best-first search [27] visits the node that minimizes $h(S)$.

For pruning, a heuristic function needs to be *admissible*: $h(S)$ is a lower bound of the shortest path weight from a node S to a goal node. In the conventional shortest path problem, if a heuristic function h is admissible, given $g(S)$, the weight of a path $\sigma(S)$ from the initial node to S, $g(S)+h(S)$ is a lower bound on the weight of any path extending $\sigma(S)$ to a goal node. Therefore, when we have found a path from the initial node to a goal node with weight $\overline{\gamma}$, we can prune the path $\sigma(S)$ if $g(S) + h(S) \geq \overline{\gamma}$. With this pruning, a heuristic search algorithm can be considered a branch-and-bound algorithm [225, 325].

While the above two functionalities of a heuristic function are fundamentally different, it is common that a single admissible heuristic function is used for both purposes. In particular, A* [190] visits the node that minimizes the f-value, $f(S) = g(S) + h(S)$. While A* does not explicitly prune paths, if the weights of edges are nonnegative, it never discovers a path $\sigma(S)$ such that $g(S)+h(S) > \gamma^*$, where γ^* is the shortest path weight from the initial node to a goal node.[1] Thus, A* implicitly prunes non-optimal paths while guiding the search with the f-values. However, in general, we can use different heuristic functions for the two functionalities, and the one used for search guidance need not be admissible. Such multi-heuristic search algorithms have been developed particularly for the bounded-suboptimal setting, where we want to find a solution whose suboptimality is bounded by a constant factor, and the anytime setting, where we want to find increasingly better solutions until proving optimality [336, 68, 14, 417, 5, 136].

In DyPDL, a dual bound function can be used as an admissible heuristic function, but we may use a different heuristic function for search guidance. In this section, we do not introduce heuristic functions for search guidance and do not specify how to select the next node to visit. Instead, we provide a generic heuristic search algorithm that uses a dual bound function *only* for pruning and discuss its completeness and optimality. To explicitly distinguish pruning from search guidance, for

[1] While f^* is conventionally used to represent the optimal path weight, we use γ^* to explicitly distinguish it from f-values.

Algorithm 9 Heuristic search for minimization with a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$ with $\langle A, \times, \mathbf{1} \rangle$. An approximate dominance relation $\preceq_a$ and a dual bound function η are given as input.

1: **if** $S^0 \not\models \mathcal{C}$ **then return** NULL
2: $\overline{\gamma} \leftarrow \infty, \overline{\sigma} \leftarrow$ NULL ▷ Initialize the solution.
3: $\sigma(S^0) \leftarrow \langle\rangle$, $g(S^0) \leftarrow \mathbf{1}$ ▷ Initialize the g-value.
4: $G, O \leftarrow \{S^0\}$ ▷ Initialize the open list.
5: **while** $O \neq \emptyset$ **do**
6: Let $S \in O$ ▷ Select a state.
7: $O \leftarrow O \setminus \{S\}$ ▷ Remove the state.
8: **if** $\exists B \in \mathcal{B}, S \models C_B$ **then**
9: current_cost $\leftarrow g(S) \times \min_{B \in \mathcal{B}: S \models C_B} \mathsf{cost}_B(S)$ ▷ Compute the solution cost.
10: **if** current_cost $< \overline{\gamma}$ **then**
11: $\overline{\gamma} \leftarrow$ current_cost, $\overline{\sigma} \leftarrow \sigma(S)$ ▷ Update the best solution.
12: $O \leftarrow \{S' \in O \mid g(S') \times \eta(S') < \overline{\gamma}\}$ ▷ Prune states in the open list.
13: **else**
14: **for all** $\tau \in \mathcal{T}(S) : S[\![\tau]\!] \models \mathcal{C}$ **do**
15: $g_{\text{current}} \leftarrow g(S) \times w_\tau(S)$ ▷ Compute the g-value.
16: **if** $\nexists S' \in G$ such that $S[\![\tau]\!] \preceq_a S'$ and $g_{\text{current}} \geq g(S')$ **then**
17: **if** $g_{\text{current}} \times \eta(S[\![\tau]\!]) < \overline{\gamma}$ **then**
18: **if** $\exists S' \in G$ such that $S' \preceq_a S[\![\tau]\!]$ and $g_{\text{current}} \leq g(S')$ **then**
19: $G \leftarrow G \setminus \{S'\}$, $O \leftarrow O \setminus \{S'\}$ ▷ Remove a dominated state.
20: $\sigma(S[\![\tau]\!]) \leftarrow \langle \sigma(S); \tau \rangle$, $g(S[\![\tau]\!]) \leftarrow g_{\text{current}}$ ▷ Update the g-value.
21: $G \leftarrow G \cup \{S[\![\tau]\!]\}$, $O \leftarrow O \cup \{S[\![\tau]\!]\}$ ▷ Insert the successor state.
22: **return** $\overline{\sigma}$ ▷ Return the solution.

a dual bound function, we use η as in Definition 18 (Section 3.1.2, p. 40) instead of h and do not use f.

We show generic pseudo-code of a heuristic search algorithm for the minimization problem with a monoidal DyPDL model in Algorithm 9. The algorithm starts from the target state S^0 and searches for a path to a base state by traversing edges in the state transition graph. The open list O stores candidate states to search. The set G stores generated states to detect duplicate or dominated states. If the model satisfies isotonicity, with Theorem 9, we just need to consider the best path to each state in terms of the weight. The sequence of transitions $\sigma(S)$ represents the best path found so far from the target state S^0 to S. The g-value of S, $g(S)$, is the weight of the path $\sigma(S)$. The function η is a dual bound function, which underestimates the cost of an S-solution by the η-value of S, $\eta(S)$. The best solution found so far, $\overline{\sigma}$, and its cost $\overline{\gamma}$ (i.e., the primal bound) is also maintained. For maximization, ∞ is replaced with $-\infty$, min is replaced with max, $<$ is replaced with $>$, and $\geq$ and $\leq$ are swapped in Algorithm 9. All the theoretical results shown later can be easily adapted to maximization.

If the target state S^0 violates the state constraints, the model does not have a solution, so we return NULL (line 1). Otherwise, the open list O and G are initialized with S^0 (line 4). The g-value of S^0 is initialized to $\mathbf{1}$ following Definition 26 (line 3). Initially, the solution cost $\overline{\gamma} = \infty$, and $\overline{\sigma} =$ NULL (line 2). When O is empty, $\overline{\sigma}$ is returned (line 22). In such a case, the state transition graph is exhausted, and the current solution $\overline{\sigma}$ is an optimal solution, or the model does not have a solution if $\overline{\sigma} =$ NULL.

When O is not empty, a state $S \in O$ is selected and removed from O (lines 6 and 7). We do

not specify how to select S in Algorithm 9 as it depends on the concrete heuristic search algorithms implemented. If S is a base state, $\sigma(S)$ is a solution, so we update the best solution if $\sigma(S)$ is better (lines 8–11). If the best solution is updated, we prune a state S' in O such that $g(S') \times \eta(S')$ is not better than the new solution cost since the currently found paths to such states do not lead to a better solution (line 12).

If S is not a base state, S is *expanded*. A successor states $S[\![\tau]\!]$ is generated for each edge $(S, S[\![\tau]\!], \tau)$ in the state transition graph (line 14). In doing so, successor states violating state constraints are discarded. For each successor state, we check if a state S' that dominates $S[\![\tau]\!]$ and has a better or equal g-value is already generated (line 16). In such a case, $\sigma(S')$ leads to a better or equal solution, so we prune $S[\![\tau]\!]$. Since $S[\![\tau]\!]$ itself dominates $S[\![\tau]\!]$, this check also works as duplicate detection. If a dominating state in G is not detected, and $g_{\text{current}} \times \eta(S[\![\tau]\!])$ is better than the primal bound (line 17), we insert it into G and O (line 21). Before doing so, we remove an existing state S' from G if S' is dominated by $S[\![\tau]\!]$ with a worse or equal g-value (line 18).

Algorithm 9 terminates in finite time if the model is finite and cost-algebraic. In addition, even if the model is not cost-algebraic, if it is acyclic, it still terminates in finite time. First, we show the termination for a finite and acyclic model. Intuitively, with such a model, Algorithm 9 enumerates a finite number of paths from the target state, so it eventually terminates.

Theorem 10. *Given a finite, acyclic, and monoidal DyPDL model, Algorithm 9 terminates in finite time.*

Proof. Unless the target state violates the state constraints, the algorithm terminates when O becomes an empty set. In each iteration of the loop in lines 5–21, at least one state is removed from O by line 7. However, multiple successor states can be added to O in each iteration by line 21. We prove that the number of iterations that reach line 21 is finite. With this property, O eventually becomes an empty set with finite iterations.

A successor state $S[\![\tau]\!]$ is inserted to O if it is not dominated by a state in G with a better or equal g-value, and $g_{\text{current}} \times \eta(S[\![\tau]\!])$ is less than the current solution cost. Suppose that $S[\![\tau]\!]$ was inserted into O and G in line 21, and now the algorithm generates $S[\![\tau]\!]$ again in line 14. Suppose that $g_{\text{current}} = g(S) \times w_\tau(S) \geq g(S[\![\tau]\!])$. If $S[\![\tau]\!] \in G$, then we do not add $S[\![\tau]\!]$ to O due to line 18. If $S[\![\tau]\!] \notin G$, then $S[\![\tau]\!]$ was removed from G, so we should have generated a state S' such that $S[\![\tau]\!] \preceq_a S'$ and $g(S') \leq g(S[\![\tau]\!])$ (lines 16 and 19). It is possible that S' was also removed from G, but in such a case, we have another state $S'' \in G$ such that $S[\![\tau]\!] \preceq_a S' \preceq_a S''$ and $g(S'') \leq g(S') \leq g(S[\![\tau]\!])$, so $S[\![\tau]\!]$ is not inserted into O again. Thus, if $S[\![\tau]\!]$ was ever inserted to G, then $S[\![\tau]\!]$ is inserted to O in line 21 only if $g_{\text{current}} < g(S[\![\tau]\!])$. We need to find a better path from S^0 to $S[\![\tau]\!]$. Since the model is finite and acyclic, the number of paths from S^0 to each state is finite. Therefore, each state is inserted to O finite times. Since the model is finite, the number of reachable states is finite. By line 14, we only generate reachable states. Thus, we reach line 21 finitely many times. □

When the state transition graph contains cycles, there can be an infinite number of paths even if the graph is finite. However, if the model is cost-algebraic, the cost monotonically changes along a path, so having a cycle does not improve a solution. Thus, the algorithm terminates in finite time by enumerating a finite number of acyclic paths. We start with the following lemma, which confirms that the g-value is the weight of the path from the target state.

Lemma 3. *After line 4 of Algorithm 9, for each state $S \in O$, S is the target state S^0, or S is reachable from S^0 with $\sigma(S)$ such that $g(S) = w_{\sigma(S)}(S^0)$ at all lines except for 20–21.*

Proof. Assume that the following condition holds at the beginning of the current iteration: for each state $S \in O$, S is the target state S^0 with $g(S^0) = \mathbf{1}$, or S is reachable from S^0 with $\sigma(S)$ and $g(S) = w_{\sigma(S)}(S)$. In the first iteration, $O = \{S^0\}$, so the assumption holds. When the assumption holds, the condition continues to hold until reaching lines 20–21, where the g-value is updated, and a new state is added to O. If we reach these lines, a non-base state S was removed from O in line 7. Each successor state $S[\![\tau]\!]$ is reachable from S with τ since S satisfies the state constraints by line 14. By the assumption, $S = S^0$, or S is reachable from S^0 with $\sigma(S)$. Therefore, $S[\![\tau]\!]$ is reachable from S^0 with $\langle \sigma(S); \tau \rangle$ by Theorem 1. If $S[\![\tau]\!]$ is inserted into O, then $\sigma(S[\![\tau]\!]) = \langle \sigma(S); \tau \rangle$. If $S = S^0$,

$$g(S[\![\tau]\!]) = g(S^0) \times w_\tau(S^0) = \mathbf{1} \times w_\tau(S^0) = w_\tau(S^0) = w_{\sigma(S[\![\tau]\!])}(S^0).$$

If S is not the target state, since $g(S) = w_{\sigma(S)}(S^0)$, by Definition 26,

$$g(S[\![\tau]\!]) = g(S) \times w_\tau(S) = w_{\sigma(S)}(S^0) \times w_\tau(S) = w_{\sigma(S[\![\tau]\!])}(S^0).$$

Thus, $S[\![\tau]\!]$ is reachable from S^0 with $\sigma(S[\![\tau]\!])$ and $g(S[\![\tau]\!]) = w_{\sigma(S[\![\tau]\!])}(S^0)$, so the condition holds after line 21. By mathematical induction, the lemma is proved. □

Theorem 11. *Given a finite and cost-algebraic DyPDL model, Algorithm 9 terminates in finite time.*

Proof. The proof is almost the same as the proof of Theorem 10. However, now, there may be an infinite number of paths to a state since the state transition graph may contain cycles. We show that the algorithm never considers a path containing cycles when the model is cost-algebraic. Assume that for each state S, the best-found path $\sigma(S)$ is acyclic up to the current iteration. This condition holds at the beginning since $\sigma(S^0) = \langle\rangle$ is acyclic. Suppose that the algorithm generates a successor state $S[\![\tau]\!]$ that is already included in the path $\sigma(S)$. Then, $S[\![\tau]\!]$ was generated before. In addition, $S[\![\tau]\!]$ is not a base state since it has a successor state on $\sigma(S)$. Since $\sigma(S)$ is acyclic, $S[\![\tau]\!]$ is included only once. Let $\sigma(S) = \langle \sigma^1; \sigma^2 \rangle$ where σ^1 is the path from S^0 to $S[\![\tau]\!]$. By Lemma 3, we have

$$g_{\text{current}} = g(S) \times w_\tau(S) = w_{\sigma^1}(S^0) \times w_{\sigma^2}(S[\![\tau]\!]) \times w_\tau(S).$$

If $g(S[\![\tau]\!])$ and $\sigma(S[\![\tau]\!])$ were updated after $S[\![\tau]\!]$ was generated with $\sigma(S[\![\tau]\!]) = \sigma^1$, then a path from S^0 to $S[\![\tau]\!]$ with a smaller weight was found by line 16. Thus, $g(S[\![\tau]\!]) \leq w_{\sigma^1}(S^0) = w_{\sigma^1}(S^0) \times \mathbf{1}$. By Definition 23, $\mathbf{1} \leq w_{\sigma^2}(S[\![\tau]\!]) \times w_\tau(S)$. Since A is isotone,

$$g_{\text{current}} = w_{\sigma^1}(S^0) \times w_{\sigma^2}(S[\![\tau]\!]) \times w_\tau(S) \geq w_{\sigma^1}(S^0) \times \mathbf{1} \geq g(S[\![\tau]\!]).$$

Therefore, $S[\![\tau]\!]$ is not inserted into O, and $\sigma(S[\![\tau]\!])$ remains acyclic. Thus, by mathematical induction, for each state, the number of insertions into O is at most the number of acyclic paths to that state, which is finite. □

We confirm that $\overline{\sigma}$ is a solution for a model when it is not NULL even during execution. In other words, Algorithm 9 is an anytime algorithm that can return a solution before proving the optimality.

Theorem 12. *After line 11 of Algorithm 9, if $\overline{\sigma} \neq$ NULL, then $\overline{\sigma}$ is a solution for the DyPDL model with $\overline{\gamma} = \mathsf{cost}_{\overline{\sigma}}(S^0)$.*

Proof. The solution $\overline{\sigma}$ is updated in line 11 when a base state S is removed from O in line 7. If $S = S^0$, then $\overline{\sigma} = \langle\rangle$, which is a solution. Since $g(S^0) = \mathbf{1}$, $\overline{\gamma} = \min_{B \in \mathcal{B}: S^0 \models C_B} \mathsf{cost}_B(S^0) = \mathsf{cost}_{\overline{\sigma}}(S^0)$ by Definition 9. If S is not the target state, $\overline{\sigma} = \sigma(S)$, which is a solution since S is reachable from S^0 with $\sigma(S)$ by Lemma 3, and S is a base state. Since $g(S) = w_{\sigma(S)}(S^0)$, it holds that $\overline{\gamma} = w_{\sigma(S)}(S^0) \times \min_{B \in \mathcal{B}: S \models C_B} \mathsf{cost}_B(S) = \mathsf{cost}_{\overline{\sigma}}(S^0)$ by Lemma 2. □

Finally, we prove the optimality of Algorithm 9. Intuitively, the proof clarifies that an optimal solution is not pruned by dominance.

Lemma 4. *In Algorithm 9, suppose that a solution exists for the DyPDL model, and let $\hat{\gamma}$ be its cost. When reaching line 5, at least one of the following two conditions is satisfied:*

- $\overline{\gamma} \leq \hat{\gamma}$.
- *O contains a state $\hat{S}$ such that an $\hat{S}$-solution $\hat{\sigma}$ exists, and $\langle \sigma(\hat{S}); \hat{\sigma} \rangle$ is a solution for the model with $\mathsf{cost}_{\langle \sigma(\hat{S}); \hat{\sigma} \rangle}(S^0) \leq \hat{\gamma}$.*

Proof. Once $\overline{\gamma} \leq \hat{\gamma}$ holds, $\overline{\gamma}$ never increases, so the lemma continues to hold. We only consider $\overline{\gamma} > \hat{\gamma}$ in the current iteration and examine if the lemma will hold in the next iteration. Now, we further specify our assumption: $\overline{\gamma} \leq \hat{\gamma}$, or O contains a state $\hat{S}$ such that an $\hat{S}$-solution $\hat{\sigma}$ exists, $\langle \sigma(\hat{S}); \hat{\sigma} \rangle$ is a solution for the model with $\mathsf{cost}_{\langle \sigma(\hat{S}); \hat{\sigma} \rangle}(S^0) \leq \hat{\gamma}$, and $|\hat{\sigma}| \leq |\sigma'|$ for each S'-solution σ' with $\mathsf{cost}_{\langle \sigma(S'); \sigma' \rangle}(S^0) \leq \hat{\gamma}$ for each $S' \in G$. At the beginning, $\hat{S} = S^0 \in O$, any solution is an extension of $\sigma(S^0)$, and $G = \{S^0\}$, so the assumption holds.

In line 7, S is removed from O. If S is a base state, $\overline{\gamma}$ and $\overline{\sigma}$ can be updated. If $\overline{\gamma}$ becomes less than or equal to $\hat{\gamma}$, the assumption holds in the next iteration. Otherwise, $\overline{\gamma} > \hat{\gamma}$. Since there exists a solution extending $\sigma(\hat{S})$ with the cost at most $\hat{\gamma}$ by the assumption, $S \neq \hat{S}$. By Lemma 3, $g(\hat{S}) \times \eta(\hat{S}) = w_{\sigma(\hat{S})}(S^0) \times \eta(\hat{S})$. By Definition 18, $\eta(\hat{S}) \leq \mathsf{cost}_{\hat{\sigma}}(\hat{S})$. Since A is isotone, $g(\hat{S}) \times \eta(\hat{S}) \leq w_{\sigma(\hat{S})}(S^0) \times \mathsf{cost}_{\hat{\sigma}}(\hat{S}) \leq \hat{\gamma} < \overline{\gamma}$. Thus, $\hat{S}$ is not removed from O in line 12.

If S is not a base state, its successor states are generated in lines 14–21. Since S was included in O, $S \in G$ by lines 19 and 21. If there exists an S-solution σ^1 with $\mathsf{cost}_{\langle \sigma(S); \sigma^1 \rangle}(S^0) \leq \hat{\gamma}$, then $|\sigma^1| \geq |\hat{\sigma}|$ by the assumption. We consider the following cases.

1. There does not exist an S-solution σ^1 satisfying both $\mathsf{cost}_{\langle \sigma(S); \sigma^1 \rangle}(S^0) \leq \hat{\gamma}$ and $|\sigma^1| \leq |\hat{\sigma}|$.
2. There exists an S-solution σ^1 with $\mathsf{cost}_{\langle \sigma(S); \sigma^1 \rangle}(S^0) \leq \hat{\gamma}$ and $|\sigma^1| = |\hat{\sigma}|$.

In the first case, $S \neq \hat{S}$. For each successor state $S[\![\tau]\!]$, if there does not exist an $S[\![\tau]\!]$-solution σ^2 such that $\mathsf{cost}_{\langle \sigma(S); \tau; \sigma^2 \rangle}(S^0) \leq \hat{\gamma}$ holds, adding $S[\![\tau]\!]$ to O in line 21 does not affect the assumption as long as $\hat{S} \in G$. Suppose that there exists an $S[\![\tau]\!]$-solution σ^2 such that $\mathsf{cost}_{\langle \sigma(S); \tau; \sigma^2 \rangle}(S^0) \leq \hat{\gamma}$ holds. Then, since $\langle \tau; \sigma^2 \rangle$ is an S-solution, $|\langle \tau; \sigma^2 \rangle| > |\hat{\sigma}|$, so $|\sigma^2| \geq |\hat{\sigma}|$. Again, as long as $\hat{S} \in G$, adding $S[\![\tau]\!]$ to G does not affect the assumption. Removing $\hat{S}$ from G in line 16 is possible only if $\hat{S} \preceq_a S[\![\tau]\!]$ and $g_{\text{curent}} = g(S) \times w_\tau(S) \leq g(\hat{S})$ in line 19. In such a case, since $\hat{S} \preceq S[\![\tau]\!]$, there exists an $S[\![\tau]\!]$-solution σ^2 such that $\mathsf{cost}_{\sigma^2}(S[\![\tau]\!]) \leq \mathsf{cost}_{\hat{\sigma}}(\hat{S})$ with $|\sigma^2| \leq |\hat{\sigma}|$. Since $\langle \sigma(S); \tau; \sigma^2 \rangle$ is a solution for the model, by Lemma 3,

$$\mathsf{cost}_{\langle \sigma(S); \tau; \sigma^2 \rangle}(S^0) = g(S) \times w_\tau(S) \times \mathsf{cost}_{\sigma^2}(S[\![\tau]\!]).$$

Since $S[\![\tau]\!]$ is reachable from S^0 with $\langle \sigma(S); \tau \rangle$, by Theorem 9,

$$\mathsf{cost}_{\langle \sigma(S);\tau;\sigma^2 \rangle}(S^0) = g(S) \times w_\tau(S) \times \mathsf{cost}_{\sigma^2}(S[\![\tau]\!]) \leq g(\hat{S}) \times \mathsf{cost}_{\hat{\sigma}}(\hat{S}) = \mathsf{cost}_{\langle \sigma(\hat{S});\hat{\sigma} \rangle}(S^0) \leq \hat{\gamma}.$$

Because $|\sigma^2| \leq |\hat{\sigma}|$, by considering $S[\![\tau]\!]$ as a new $\hat{S}$, the assumption will hold in the next iteration.

In the second case, there exists a transition $\tau \in \mathcal{T}(S)$ such that there exists an $S[\![\tau]\!]$-solution σ^2, and $\langle \sigma(S); \tau; \sigma^2 \rangle$ is a solution with $\mathsf{cost}_{\langle \sigma(S);\tau;\sigma^2 \rangle}(S^0) \leq \hat{\gamma}$ and $|\langle \tau; \sigma^2 \rangle| = |\hat{\sigma}|$, which implies $|\sigma^2| = |\hat{\sigma}| - 1$. Let $S[\![\tau]\!]$ be the first one considered in the loop among such successor states. First, we show that $S[\![\tau]\!]$ is inserted into O in line 21. Then, we prove that $S[\![\tau]\!]$ or another successor state replacing $S[\![\tau]\!]$ in line 19 can be considered a new $\hat{S}$ in the next iteration. For a successor state $S[\![\tau']\!]$ considered before $S[\![\tau]\!]$, if there exists an $S[\![\tau']\!]$-solution σ^3 with $\mathsf{cost}_{\langle \sigma(S[\![\tau']\!]);\sigma^3 \rangle}(S^0) \leq \hat{\gamma}$, then $|\langle \tau'; \sigma^3 \rangle| > |\hat{\sigma}|$, so $|\sigma^3| \geq |\hat{\sigma}| > |\sigma^2|$. Therefore, adding $S[\![\tau']\!]$ to G does not affect the assumption. Suppose that $S[\![\tau]\!]$ is not added to O due to line 16. Then, there exists a state $S' \in G$ such that $S[\![\tau]\!] \preceq_a S'$ and $g_{\text{current}} = g(S) \times w_\tau(S) \geq g(S')$. Since $S[\![\tau]\!] \preceq_a S'$, there exists an S'-solution σ' with $\mathsf{cost}_{\sigma'}(S') \leq \mathsf{cost}_{\sigma^2}(S[\![\tau]\!])$ and $|\sigma'| \leq |\sigma^2|$. However, by the assumption, $|\sigma'| \geq |\hat{\sigma}| > |\sigma^2|$, which is a contradiction. Therefore, there does not exist such S', and the condition in line 16 is true. Next, we examine the condition in line 17. By Lemma 3,

$$\mathsf{cost}_{\langle \sigma(S);\tau;\sigma^2 \rangle}(S^0) = g(S) \times w_\tau(S) \times \mathsf{cost}_{\sigma^2}(S[\![\tau]\!]) \leq \hat{\gamma}.$$

Since $\eta(S[\![\tau]\!]) \leq \mathsf{cost}_{\sigma^2}(S[\![\tau]\!])$ and A is isotone,

$$g_{\text{current}} \times \eta(S[\![\tau]\!]) = g(S) \times w_\tau(S) \times \eta(S[\![\tau]\!]) \leq g(S) \times w_\tau(S) \times \mathsf{cost}_{\sigma^2}(S[\![\tau]\!]) \leq \hat{\gamma} < \overline{\gamma}.$$

Therefore, the condition in line 17 is true, and $S[\![\tau]\!]$ is inserted into O. For a successor state $S[\![\tau']\!]$ generated after $S[\![\tau]\!]$, suppose that there does not exist an $S[\![\tau']\!]$-solution σ^3 with $\mathsf{cost}_{\langle \sigma(S);\tau';\sigma^3 \rangle}(S^0) \leq \hat{\gamma}$. Adding $S[\![\tau']\!]$ to G does not affect the assumption as long as $S[\![\tau]\!] \in G$. If there exists such σ^3, then $|\sigma^3| = |\hat{\sigma}| - 1$ or $|\sigma^3| \geq |\hat{\sigma}|$ by the assumption. In the former case, adding $S[\![\tau']\!]$ to G does not affect the assumption as long as $S[\![\tau']\!] \in G$ since we can consider $S[\![\tau']\!]$ as a new $\hat{S}$ in the next iteration. In the latter case, adding $S[\![\tau']\!]$ to G does not affect the assumption as long as $S[\![\tau]\!] \in G$ since $|\sigma^3| > |\sigma^2|$. The remaining problem is the possibility that $S[\![\tau]\!]$ is removed in line 19. If the condition in line 16 is true, then $g(S) \times w_{\tau'}(S) \leq g(S) \times w_\tau(S)$, and there exists an $S[\![\tau']\!]$-solution σ^3 with $\mathsf{cost}_{\sigma^3}(S[\![\tau']\!]) \leq \mathsf{cost}_{\sigma^2}(S[\![\tau]\!])$ and $|\sigma^3| \leq |\sigma^2|$. By Theorem 9,

$$\mathsf{cost}_{\langle \sigma(S);\tau';\sigma^3 \rangle}(S^0) = g(S) \times w_{\tau'}(S) \times \mathsf{cost}_{\sigma^3}(S[\![\tau']\!]) \leq g(S) \times w_\tau(S) \times \mathsf{cost}_{\sigma^2}(S[\![\tau]\!]) \leq \hat{\gamma}.$$

Therefore, if we consider $S[\![\tau']\!]$ as a new $\hat{S}$ in the next iteration instead of $S[\![\tau]\!]$, the situation does not change. Similarly, if $S[\![\tau']\!]$ is replaced with another successor state, by considering it as a new $\hat{S}$, the situation does not change, and the assumption will hold in the next iteration. □

Theorem 13. *Let $\langle A, \times, \mathbf{1} \rangle$ be a monoid where $A \subseteq \mathbb{Q} \cup \{-\infty, \infty\}$ and A is isotone. Given a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$ with $\langle A, \times, \mathbf{1} \rangle$, if an optimal solution exists for the minimization problem with the model, and Algorithm 9 returns a solution that is not NULL, then the solution is optimal. If Algorithm 9 returns NULL, then the model is infeasible.*

Proof. Suppose that a solution exists, and let $\hat{\gamma}$ be its cost. By Lemma 4, when we reach line 5

with $O = \emptyset$, $\overline{\gamma} \leq \hat{\gamma}$. Since $\overline{\gamma} \neq \infty$, it holds that $\overline{\sigma} \neq$ NULL by line 11. Therefore, if a solution exists, NULL is never returned, i.e., NULL is returned only if the model is infeasible. Suppose that an optimal solution exists, and let γ^* be its cost. Now, consider the above discussion with $\hat{\gamma} = \gamma^*$. When we reach line 5 with $O = \emptyset$, $\overline{\gamma} \leq \gamma^*$. By Lemma 4, $\overline{\sigma}$ is a solution with $\mathsf{cost}_{\overline{\sigma}}(S^0) = \overline{\gamma}$. Since $\mathsf{cost}_{\overline{\sigma}}(S^0) \geq \gamma^*$, $\overline{\gamma} = \gamma^*$ and $\overline{\sigma}$ is an optimal solution. Therefore, if an optimal solution exists, and the algorithm returns a solution, the solution is optimal. □

Corollary 3. *Given a finite and cost-algebraic DyPDL model, the minimization or maximization problem with the model has an optimal solution, or the model is infeasible. A problem to decide if a solution whose cost is less (greater) than a given rational number exists for minimization (maximization) is decidable.*

Note that Theorem 13 does not require a model to be finite, acyclic, or cost-algebraic. While the algorithm terminates in finite time if the model is finite and acyclic or cost-algebraic, there is no guarantee in general due to the undecidability in Theorem 1. However, even for such a model, if the algorithm terminates, the optimality or infeasibility is proved.

As shown in the proof of Lemma 4, when an optimal solution exists, a state $\hat{S}$ such that there exists an optimal solution extending $\sigma(\hat{S})$ is included in the open list. For minimization (maximization), by taking the minimum (maximum) $g(S) \times \eta(S)$ value in the open list, we can obtain a dual bound, i.e., a lower (upper) bound on the optimal solution cost.

Theorem 14. *Let $\langle A, \times, \mathbf{1} \rangle$ be a monoid where $A \subseteq \mathbb{Q} \cup \{-\infty, \infty\}$ and A is isotone. Given a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$ with $\langle A, \times, \mathbf{1} \rangle$, if an optimal solution for the minimization problem with the model exists and has the cost γ^*, and O is not empty in line 5, then*

$$\min_{S \in O} g(S) \times \eta(S) \leq \gamma^*.$$

Proof. By Lemma 4, if $O \neq \emptyset$, then $\overline{\gamma} = \gamma^*$, or there exists a state $\hat{S} \in O$ on an optimal solution, i.e., there exists an $\hat{S}$-solution $\hat{\sigma}$ such that $\langle \sigma(\hat{S}); \hat{\sigma} \rangle$ is an optimal solution. If $\overline{\gamma} = \gamma^*$, by lines 12 and 17, $\min_{S \in O} g(S) \times \eta(S) < \gamma^*$. Otherwise, $\hat{S} \in O$. Since $\eta(\hat{S}) \leq \mathsf{cost}_{\hat{\sigma}}(\hat{S})$ and A is isotone,

$$\min_{S \in O} g(S) \times \eta(S) \leq g(\hat{S}) \times \eta(\hat{S}) \leq g(\hat{S}) \times \mathsf{cost}_{\hat{\sigma}}(\hat{S}) = \gamma^*.$$

□

4.2 Heuristic Search Algorithms for DyPDL

We introduce existing heuristic search algorithms as instantiations of Algorithm 9 so that we can use them for DyPDL. In particular, each algorithm differs in how to select a state S to remove from the open list O in line 6. In addition to A*, which is the most fundamental heuristic search algorithm, we select anytime algorithms that have been applied to combinatorial optimization problems in problem-specific settings. For detailed descriptions of the algorithms, please refer to the papers that proposed them. Similar to A*, in our configuration, these algorithms use a heuristic function h and guide the search with the f-value, which is computed as $f(S) = g(S) \times h(S)$, where $\times$ is a binary operator such as $+$. As we discussed in Section 4.1.4, h is not necessarily a dual bound function and not necessarily admissible.

4.2.1 CAASDy: Cost-Algebraic A* Solver for DyPDL

A* selects a state with the best f-value in lines 7 (i.e., the minimum f-value for minimization and the maximum f-value for maximization). If there are multiple states with the best f-value, one is selected according to a tie-breaking strategy. Among states with the same f-value, we select a state with the best h-value (with "best" defined accordingly to the best f-value). In what follows, if we select a state according to the f-values in other algorithms, we also assume that a state with the best f-value is selected, and ties are broken by the h-values. If there are multiple states with the best f- and h-values, we use another tie-breaking strategy, which is not specified here and discussed later when we describe the implementation. We call our solver cost-algebraic A* solver for DyPDL (CAASDy) as we originally proposed it only for cost-algebraic models [267]. However, as shown in Theorems 12 and 13, CAASDy is applicable to monoidal and acyclic models with a monoid $\langle A, \otimes, \mathbf{1} \rangle$ if A is isotone.

In original A*, if h is admissible, the first solution found is optimal. Previous work has generalized A* to cost-algebraic heuristic search with this property [114]. In our case, if a model is not cost-algebraic, the first solution may not be an optimal solution. In addition, even if a model is cost-algebraic, our problem setting is slightly different from Edelkamp, Jabbar, and Lafuente [114]: a base case has a cost, so the cost of a solution can be different from the weight of the corresponding path. If a model is cost-algebraic and the costs of the base cases do not matter, we can prove that the first solution found by CAASDy is optimal.

Theorem 15. *Given a cost-algebraic DyPDL model with a monoid $\langle A, \otimes, \mathbf{1} \rangle$, let h be an admissible heuristic function, i.e., given any reachable solution S and any S-solution σ, $h(S) \leq \mathsf{cost}_\sigma(S)$ for minimization. If an optimal solution exists for the model, the costs of base cases are $\mathbf{1}$, i.e., $\forall B \in \mathcal{B}, \mathsf{cost}_B(S) = \mathbf{1}$, and $h(S) \geq \mathbf{1}$ for any reachable state S, then the first found solution by CAASDy is optimal.*

Proof. Let $\overline{\sigma} = \sigma(S)$ be the first found solution with the cost $\overline{\gamma}$ in line 11. Since $\min_{B \in \mathcal{B}: S \models C_B} \mathsf{cost}_B(S) = \mathbf{1}$, $\overline{\gamma} = g(S)$. Since $\mathbf{1} \leq h(S) \leq \min_{B \in \mathcal{B}: S \models C_B} \mathsf{cost}_B(S) = \mathbf{1}$, $f(S) = g(S) \otimes h(S) = g(S) = \overline{\gamma}$. If $\sigma(S)$ is not an optimal solution, by Lemma 4, O contains a state $\hat{S} \in O$ such that there exists an $\hat{S}$-solution $\hat{\sigma}$ with $\mathsf{cost}_{\langle \sigma(\hat{S}); \hat{\sigma} \rangle}(S^0) = \gamma^* < \overline{\gamma}$. Since A is isotone, $\mathsf{cost}_{\langle \sigma(\hat{S}); \hat{\sigma} \rangle}(S^0) = g(\hat{S}) \otimes \mathsf{cost}_{\hat{\sigma}}(\hat{S}) \geq g(\hat{S}) \otimes h(\hat{S})$. Therefore,

$$f(\hat{S}) = g(\hat{S}) \otimes h(\hat{S}) \leq \mathsf{cost}_{\langle \sigma(\hat{S}); \hat{\sigma} \rangle}(S^0) < \overline{\gamma} = f(S).$$

Thus, $\hat{S}$ should have been expanded before S, which is a contradiction. □

4.2.2 Depth-First Branch-and-Bound (DFBnB)

Depth-First Branch-and-Bound (DFBnB) expands states in the depth-first order, i.e., a state S that maximizes $|\sigma(S)|$ is selected in line 6. Concretely, the open list O is implemented as a stack, and the state on the top of the stack (the state added to O most recently) is selected in line 6. Successor states of the same state have the same priority, so ties are broken by the f-values. DFBnB has been applied to state-based formulations of combinatorial optimization problems such as the sequential ordering problem (SOP) [291], the traveling salesperson problem (TSP), and single machine scheduling [427, 426].

4.2.3 Cyclic-Best First Search (CBFS)

Cyclic-best first search (CBFS) [239] partitions the open list O into layers O_i for each depth i. A state S is inserted into O_i if $|\sigma(S)| = i$. At the beginning, $O_0 = \{S^0\}$ and $O_i = \emptyset$ for $i > 0$. Starting with $i = 0$, if $O_i \neq \emptyset$, CBFS selects a state having the best priority from O_i in line 6 and inserts successor states into O_{i+1} in line 21. We use the f-value as the priority. After that, CBFS increases i by 1. However, when i is the maximum depth, CBFS resets i to 0 instead of incrementing it. The maximum depth is usually known in a problem-specific setting, but we do not use a fixed parameter in our setting. Instead, we set i to 0 when a new best solution is found after line 11, or $O_j = \emptyset$ for all $j \geq i$. In problem specific settings, CBFS was used in single machine scheduling [239] and the simple assembly line balancing problem (SALBP-1) [388, 319].

4.2.4 Anytime Column Search (ACS)

Anytime column search (ACS) [427] can be considered a generalized version of CBFS, expanding b states at each depth. ACS also partitions the open list O into O_i for each depth i and selects a state from O_i in line 6, starting with $i = 0$ and $O_0 = \{S_0\}$. ACS increases i by 1 after removing b states from O_i or when O_i becomes empty, where b is a parameter. We remove the best b states according to the f-values.

Anytime column progressive search (ACPS) [427] is a non-parametric version of ACS, which starts from $b = 1$ and increases b by 1 when it reaches the maximum depth. Similarly to CBFS, we set i to 0 and increase b by 1 when a new best solution is found or $\forall j \geq i, O_j = \emptyset$. For combinatorial optimization, ACS and ACPS were evaluated on TSP [427].

4.2.5 Anytime Pack Search (APS)

Anytime pack search (APS) [426] maintains the set of the best states $O_b \subseteq O$, initialized with $\{S^0\}$, the set of the best successor states $O_c \subseteq O$, and a suspend list $O_s \subseteq O$. APS selects states from O_b in line 6 and inserts the best b successor states according to a priority into O_c and other successor states into O_s. When there are fewer than b successor states, all of them are inserted into O_c. After expanding states in O_b, APS swaps O_b and O_c and continues the procedure. If O_b and O_c are empty, the best b states are moved from O_s to O_b. We use the f-value as the priority to select states.

Anytime pack progressive search (APPS) [426] starts from $b = 1$ and increases b by δ if $b < \overline{b}$ when the best b states are moved from O_s to O_b, where δ and $\overline{b}$ are parameters. We use $\delta = 1$ and $\overline{b} = \infty$ following the configuration in TSP and single machine scheduling [426].

4.2.6 Discrepancy-Based Search

Discrepancy-based search [192] considers the discrepancy of a state, the number of deviations from the heuristically best path to the state. The target state has a discrepancy of 0. When a state S has a discrepancy of d, its successor states are assigned priorities, and the state with the best priority has the discrepancy of d. Other successor states have the discrepancy of $d + 1$.

Discrepancy-bounded depth-first search (DBDFS) [26] performs depth-first search that only expands states having the discrepancy between $(i - 1)k$ and $ik - 1$ inclusive, where i starts from 1 and increases by 1 when all states within the range are expanded, and k is a parameter. We use the

f-value as the priority. The open list is partitioned into two sets O_0 and O_1, and $O_0 = \{S^0\}$ and $O_1 = \emptyset$ at the beginning. A state is selected from O_0 in line 6. Successor states with the discrepancy between $(i-1)k$ and $ik-1$ are added to O_0, and other states are added to O_1. When O_0 becomes empty, it is swapped with O_1, and i is increased by 1. The discrepancy of states in O_1 is ik because the discrepancy is increased by at most 1 at a successor state. Therefore, after swapping O_0 with O_1, the discrepancy of states in O_0 falls between the new bounds, ik and $(i+1)k-1$. For depth-first search, when selecting a state to remove from O_0, we break ties by the f-values. We use $k = 1$ in our configuration. Discrepancy-based search was originally proposed as tree search [192, 26] and later applied to state space search for SOP [291].

4.2.7 Complete Anytime Beam Search (CABS)

Beam search is a heuristic search algorithm that searches the state transition graph layer by layer. Unlike the other heuristic search algorithms presented above, beam search cannot be considered an instantiation of Algorithm 9, so we provide dedicated pseudo-code in Algorithm 10. Beam search maintains states in the same layer, i.e., states that are reached with the same number of transitions, in the open list O, which is initialized with the target state. Beam search expands all states in O, inserts the best b successor states into O, and discards the remaining successor states, where b is a parameter called a beam width. Beam search may discard all successor states leading to solutions, so it is incomplete, i.e., it may not find a solution.

Complete anytime beam search (CABS) [447] is a complete version of beam search. Zhang [447] considered a generalized version of beam search, i.e., successor states inserted into O are decided by a user-provided pruning rule, which can be different from selecting the best b states. In the original definition, CABS iteratively executes beam search while relaxing the pruning rule so that more and more states are inserted into O. CABS terminates when it finds a satisfactory solution according to some criterion. In our case, we use the common definition of beam search, which selects the best b states according to the f-values. CABS iteratively executes beam search while increasing b in each iteration until finding an optimal solution or proving infeasibility. In particular, we start from $b = 1$ and double b after each iteration, following the configuration used in SOP [291] and the permutation flowshop [292].

In Algorithm 10, beam search maintains the set of states in the current layer using the open list O, and the set of states in the next layer using G. The open list O is updated to G after generating all successor states while pruning states based on the bound (line 22). If O contains more than b states, only the best b states are kept. This operation may prune optimal solutions, so the flag complete, which indicates the completeness of beam search, becomes $\bot$. Beam search terminates when O becomes empty, or a solution is found (line 5). Even if complete $= \top$, and a solution is found, when O is not empty, we may miss a better solution (line 25). Therefore, we update complete to $\bot$ in such a case (line 26). We can derive the maximization version of Algorithm 10 in a similar way as Algorithm 9.

Beam search in Algorithm 10 has several properties that are different from Algorithm 9.

1. The set G, which is used to detect dominance, contains only states in the next layer.
2. A state S may be removed from the open list O even if $g(S) \times \eta(S) < \overline{\gamma}$ (line 24).
3. Beam search may terminate when a solution is found even if $O \neq \emptyset$.

Algorithm 10 Beam search for minimization with a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$ with $\langle A, \times, \mathbf{1} \rangle$. An approximate dominance relation $\preceq_a$, a dual bound function η, a primal bound $\overline{\gamma}$, and a beam width b are given as input.

1: **if** $S^0 \not\models \mathcal{C}$ **then return** NULL, $\top$
2: $\overline{\sigma} \leftarrow$ NULL, complete $\leftarrow \top$ ▷ Initialize the solution.
3: $l \leftarrow 0$, $\sigma^l(S^0) \leftarrow \langle\rangle$, $g^l(S^0) \leftarrow \mathbf{1}$ ▷ Initialize the g-value.
4: $O \leftarrow \{S^0\}$ ▷ Initialize the open list.
5: **while** $O \neq \emptyset$ and $\overline{\sigma} =$ NULL **do**
6: $G \leftarrow \emptyset$ ▷ Initialize the set of states.
7: **for all** $S \in O$ **do**
8: **if** $\exists B \in \mathcal{B}, S \models C_B$ **then**
9: current_cost $\leftarrow g^l(S) \times \min_{B \in \mathcal{B}: S \models C_B} \mathsf{cost}_B(S)$ ▷ Compute the solution cost.
10: **if** current_cost $< \overline{\gamma}$ **then**
11: $\overline{\gamma} \leftarrow$ current_cost, $\overline{\sigma} \leftarrow \sigma^l(S)$ ▷ Update the best solution.
12: **else**
13: **for all** $\tau \in \mathcal{T}(S) : S[\![\tau]\!] \models \mathcal{C}$ **do**
14: $g_{\text{current}} \leftarrow g^l(S) \times w_\tau(S)$ ▷ Compute the g-value.
15: **if** $\nexists S' \in G$ such that $S[\![\tau]\!] \preceq_a S'$ and $g_{\text{current}} \geq g^{l+1}(S')$ **then**
16: **if** $g_{\text{current}} \times \eta(S[\![\tau]\!]) < \overline{\gamma}$ **then**
17: **if** $\exists S' \in G$ such that $S' \preceq_a S[\![\tau]\!]$ and $g_{\text{current}} \leq g^{l+1}(S')$ **then**
18: $G \leftarrow G \setminus \{S'\}$ ▷ Remove a dominated state.
19: $\sigma^{l+1}(S[\![\tau]\!]) \leftarrow \langle \sigma^l(S); \tau \rangle$, $g^{l+1}(S[\![\tau]\!]) \leftarrow g_{\text{current}}$ ▷ Update the g-value.
20: $G \leftarrow G \cup \{S[\![\tau]\!]\}$ ▷ Insert the successor state.
21: $l \leftarrow l + 1$ ▷ Proceed to the next layer.
22: $O \leftarrow \{S \in G \mid g^l(S) \times \eta(S) < \overline{\gamma}\}$ ▷ Prune states by the bound.
23: **if** $|O| > b$ **then**
24: $O \leftarrow$ the best b states in O, complete $\leftarrow \bot$ ▷ Keep the best b states.
25: **if** complete and $O \neq \emptyset$ **then**
26: complete $\leftarrow \bot$ ▷ A better solution may exist.
27: **return** $\overline{\sigma}$, complete ▷ Return the solution.

With Property (1), beam search can potentially save memory compared to Algorithm 9. This method can be considered layered duplicate detection as proposed in previous work [450]. With this strategy, we do not detect duplicates when the same states appear in different layers. When a generated successor state $S[\![\tau]\!]$ in the next layer is the same as a state S' in the current layer, in line 19, we do not want to update $g(S')$ and $\sigma(S')$ since we do not check if a better path to S' is found. Thus, we maintain l, which is incremented by 1 after each layer (line 21), and use g^l and σ^l to differentiate g and σ for different layers. Our layered duplicate detection mechanism prevents us from using beam search when the state transition graph contains cycles; beam search cannot store states found in the previous layers, so it continues to expand states in a cycle. This issue can be addressed by initializing G with $\{S^0\}$ outside the while loop, e.g., just after line 4, and removing line 6. With this modification, beam search can be used for a cyclic but cost-algebraic DyPDL model. By Properties (2) and (3), there is no guarantee that beam search proves the optimality or infeasibility unless complete $= \top$. However, CABS (Algorithm 11) has the guarantee of optimality as it repeats beam search until complete becomes $\top$. In what follows, we formalize the above points. Once again, we present the theoretical results for minimization, but they can be easily adapted to maximization.

Algorithm 11 CABS for minimization with a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$ with $\langle A, \times, \mathbf{1} \rangle$. An approximate dominance relation $\preceq_a$ and a dual bound function η are given as input.

1: $\overline{\gamma} \leftarrow \infty$, $\overline{\sigma} \leftarrow$ NULL, $b \leftarrow 1$ ▷ Initialization.
2: **loop**
3: σ, complete $\leftarrow$ BeamSearch($\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$, $\preceq_a$, η, $\overline{\gamma}$, b) ▷ Execute Algorithm 10.
4: **if** $\mathsf{cost}_\sigma(S^0) < \overline{\gamma}$ **then**
5: $\overline{\gamma} \leftarrow \mathsf{cost}_\sigma(S^0)$, $\overline{\sigma} \leftarrow \sigma$ ▷ Update the solution.
6: **if** complete **then**
7: **return** $\overline{\sigma}$ ▷ Return the solution.
8: $b \leftarrow 2b$ ▷ Double the beam width.

Theorem 16. *Given a finite, acyclic, and monoidal DyPDL model, beam search terminates in finite time.*

Proof. Suppose that we have generated a successor state $S[\![\tau]\!]$, which was generated before. The difference from Algorithm 9 that we need to consider is Property (1). If $S[\![\tau]\!]$ was generated before as a successor state of a state in the current layer, by the proof of Theorem 10, there exists a state $S' \in G$ with $S[\![\tau]\!] \preceq_a S'$. The successor state $S[\![\tau]\!]$ is inserted into G again only if we find a better path to $S[\![\tau]\!]$. If $S[\![\tau]\!]$ was generated before as a successor state of a state in a previous layer, the path to $S[\![\tau]\!]$ at that time was shorter (in terms of the number of transitions) than that of the current path. The successor state $S[\![\tau]\!]$ may be inserted into G since it is not included in G. In either case, $S[\![\tau]\!]$ is inserted into G again only if we find a new path to it. Since the number of paths to $S[\![\tau]\!]$ is finite, we insert $S[\![\tau]\!]$ into G finite times. The rest of the proof follows that of Theorem 10. □

As we discussed above, with a slight modification, we can remove Property (1) and prove the termination of beam search for a cost-algebraic DyPDL model.

Since Properties (1)–(3) do not affect the proofs of Lemma 3 and Theorem 12, the following theorem holds.

Theorem 17. *After line 11 of Algorithm 10, if $\overline{\sigma} \neq$ NULL, then $\overline{\sigma}$ is a solution for the model with $\overline{\gamma} = \mathsf{cost}_{\overline{\sigma}}(S^0)$.*

We also prove the optimality of beam search when complete $= \top$.

Theorem 18. *Let $\langle A, \times, \mathbf{1} \rangle$ be a monoid where $A \subseteq \mathbb{Q} \cup \{-\infty, \infty\}$ and A is isotone. Given a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$ with $\langle A, \times, \mathbf{1} \rangle$ and $\overline{\gamma} \in A$, if an optimal solution exists for the minimization problem with the model, and beam search returns $\overline{\sigma} \neq$ NULL and complete $= \top$, then $\overline{\sigma}$ is an optimal solution. If beam search returns $\overline{\sigma} =$ NULL and complete $= \top$, then there does not exist a solution whose cost is less than $\overline{\gamma}$.*

Proof. When complete $= \top$ is returned, during the execution, beam search never reached lines 24 and 26. Therefore, we can ignore Properties (2) and (3). If we modify Algorithm 10 so that G contains states in all layers as discussed above, we can also ignore Property (1). By ignoring Properties (1)–(3), we can consider beam search as an instantiation of Algorithm 9. If the model is infeasible, or an optimal solution exists with the cost γ^* and $\overline{\gamma} > \gamma^*$ at the beginning, the proof is exactly the same. If $\overline{\gamma} \leq \gamma^*$ was given as input, beam search has never updated $\overline{\sigma}$ and $\overline{\gamma}$, and NULL is returned if it terminates. In such a case, indeed, no solution has a cost less than $\overline{\gamma} \leq \gamma^*$.

The above proof is for beam search with the modification. We confirm that it is also valid with beam search in Algorithm 10 without modification, i.e., we consider Property (1). The proof of Theorem 13 depends on Lemma 4, which claims that when a solution with a cost $\hat{\gamma}$ exists and $\overline{\gamma} > \hat{\gamma}$, the open list contains a state $\hat{S}$ such that there exists an $\hat{S}$-solution $\hat{\sigma}$ with $\mathsf{cost}_{\langle \sigma(\hat{S});\hat{\sigma}\rangle}(S^0) \leq \hat{\gamma}$. At the beginning, $\hat{S} = S^0$ exists in O. When such a state exists in the current layer, its successor state $\hat{S}[\![\tau]\!]$ with $\tau = \hat{\sigma}_1$ is generated. If $\hat{S}[\![\tau]\!]$ is not inserted into G in line 20, another state $S' \in G$ dominates $\hat{S}[\![\tau]\!]$ with a better or equal g-value, so there exists a solution extending $\sigma(S')$ with the cost at most $\hat{\gamma}$. Thus, S' can be considered a new $\hat{S}$. When $\hat{S}[\![\tau]\!]$ or S' is removed from G by line 18, another state S'' that dominates $\hat{S}[\![\tau]\!]$ or S' with a better or equal g-value is inserted into G, and there exists a solution extending $\sigma(S'')$ with the cost at most $\hat{\gamma}$. □

When we consider Property (1), the proof for Algorithm 10 is simpler than the original one due to the layer by layer nature of beam search. The set G contains only one state for one path. Therefore, we can exclude the possibility that multiple paths dominate each other, e.g., given two states S and S' in G, S dominates a successor state $S'[\![\tau']\!]$ of S', and S' dominates a successor state $S[\![\tau]\!]$ of S. In the proof of Lemma 4, to show that such a situation does not occur, we introduced the restriction that $\hat{S}$ minimizes the length of $\hat{\sigma}$. For layer by layer search, we do not need it.

For CABS, since beam search returns an optimal solution or proves the infeasibility when complete $= \top$ by Theorem 18, the optimality is straightforward by line 6 of Algorithm 11.

Corollary 4. *Let $\langle A, \times, \mathbf{1}\rangle$ be a monoid where $A \subseteq \mathbb{Q} \cup \{-\infty, \infty\}$ and A is isotone. Given a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C}\rangle$ with $\langle A, \times, \mathbf{1}\rangle$, if an optimal solution exists for the minimization problem with the model, and CABS returns a solution that is not NULL, then it is an optimal solution. If CABS returns NULL, then the model is infeasible.*

We prove that CABS terminates when a DyPDL model is finite, monoidal, and acyclic.

Theorem 19. *Given a finite, acyclic, and monoidal DyPDL model, CABS terminates in finite time.*

Proof. When the beam width b is sufficiently large, e.g., equal to the number of reachable states in the model, beam search never reaches line 24. Since the number of reachable states is finite, b eventually becomes such a large number with finite iterations. Suppose that we call beam search with sufficiently large b. If complete $= \top$ is returned, we are done. Otherwise, beam search should have found a new solution whose cost is better than $\overline{\gamma}$ in line 11 and reached line 26. In this case, there exists a solution for the model. Since the state transition graph is finite and acyclic, there are a finite number of solutions, and there exists an optimal solution with the cost γ^*. Since $\overline{\gamma}$ decreases after each call if complete $= \bot$, eventually, $\overline{\gamma}$ becomes γ^*, and complete $= \top$ is returned with finite iterations. By Theorem 16, each call of beam search terminates in finite time. Therefore, CABS terminates in finite time. □

To obtain a dual bound from beam search, we need to slightly modify Theorem 14; since beam search may discard states leading to optimal solutions, we need to keep track of the minimum (or maximum for maximization) $g^l(S) \times \eta(S)$ value for all discarded states in addition to states in O.

Theorem 20. *Let $\langle A, \times, \mathbf{1}\rangle$ be a monoid where $A \subseteq \mathbb{Q} \cup \{-\infty, \infty\}$ and A is isotone. Given a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C}\rangle$ with $\langle A, \times, \mathbf{1}\rangle$ and $\overline{\gamma} \in A$, let D_m be the set of states discarded*

in layer $m \leq l-1$ by line 24 of Algorithm 10. If an optimal solution for the minimization problem with the model exists and has the cost γ^*, just after line 22,

$$\min\left\{\overline{\gamma}, \min_{S\in O} g^l(S)\times\eta(S), \min_{m=1,\dots,l-1}\min_{S\in D_m} g^m(S)\times\eta(S)\right\} \leq \gamma^*$$

where we assume $\min_{S\in O} g^l(S)\times\eta(S) = \infty$ if $O=\emptyset$ and $\min_{S\in D_m} g^m(S)\times\eta(S)=\infty$ if $D_m=\emptyset$.

Proof. If $\overline{\gamma}\leq\gamma^*$, the inequality holds trivially, so we assume $\overline{\gamma}>\gamma^*$. We prove that there exists a state $\hat{S}\in O\cup\bigcup_{m=1}^{l-1} D_m$ on an optimal path, i.e., there exists a $\hat{S}$-solution $\hat{\sigma}$ such that $\langle\sigma^m(\hat{S});\hat{\sigma}\rangle$ is an optimal solution where $m\in\{0,...,l\}$. Initially, $O=\{S^0\}$, so the condition is satisfied. Suppose that a state $\hat{S}$ on an optimal path is included in O just before line 7. If $\hat{S}$ is a base state, we reach line 9, and current_cost $=\gamma^*$. Since $\overline{\gamma}>\gamma^*$, $\overline{\gamma}$ is updated to γ^* in line 11. Then, $\overline{\gamma}=\gamma^*\leq\gamma^*$ will hold after line 22. If $\hat{S}$ is not a base state, by a similar argument to the proof of Theorem 18 (or Theorem 13 if we consider the modified version where states in all layers are kept in G), a state on an optimal path, S', will be included in G just before line 22. Since $g^l(S')\times\eta(S')\leq\gamma^*<\overline{\gamma}$, $S'\in O$ holds after line 22, and $\min_{S\in O} g^l(S)\times\eta(S)\leq g(S')\times\eta(S')\leq\gamma^*$. After line 24, S' will be included in either O or D_l, which can be considered a new $\hat{S}$. Suppose that $\hat{S}\in D_m$ just before line 7. Since $\hat{S}$ is never removed from D_m, $\min_{S\in D_m} g^m(S)\times\eta(S)\leq g^m(\hat{S})\times\eta(\hat{S})\leq\gamma^*$ always holds. By mathematical induction, the theorem is proved. □

4.3 Experimental Evaluation

We implement and experimentally evaluate DIDP solvers using combinatorial optimization problems. For the DIDP solvers, we use the heuristic search algorithms described in Section 4.2. We compare our DIDP solvers with commercial MIP and CP solvers, Gurobi 10.0.1 [188] and IBM ILOG CP Optimizer 22.1.0 [277]. We select state-of-the-art MIP and CP models in the literature when multiple models exist and develop a new model when we do not find an existing one.

4.3.1 Software Implementation of DIDP

We develop didp-rs v0.7.0,[2] a software implementation of DIDP in Rust. It has four components, dypdl,[3] dypdl-heuristic-search,[4] didp-yaml,[5] and DIDPPy.[6] dypdl is a library for modeling, and dypdl-heuristic-search is a library for heuristic search solvers. didp-yaml is a library for parsing YAML-DyPDL with a command line interface that takes YAML-DyPDL domain and problem files and a YAML file specifying a solver as input and returns the result. DIDPPy is a Python library described in Section 3.2.3. In our experiment, we use didp-yaml.

dypdl-heuristic-search implements CAASDy, DFBnB, CBFS, ACPS, APPS, DBDFS, and CABS as DIDP solvers. These solvers can handle monoidal DyPDL models with a monoid $\langle A,\times,\mathbf{1}\rangle$ where $A\subseteq\mathbb{Q}\cup\{-\infty,\infty\}$, $\times\in\{+,\max\}$, and $\mathbf{1}=0$ if $\times=+$ or $\mathbf{1}$ is the minimum value in A if $\times=\max$. Note that $\langle A,\times,\mathbf{1}\rangle$ is a monoid (Definition 21), and A is isotone (Definition 22), which is a sufficient

[2] `https://github.com/domain-independent-dp/didp-rs/releases/tag/v0.7.0`
[3] `https://crates.io/crates/dypdl`
[4] `https://crates.io/crates/dypdl-heuristic-search`
[5] `https://crates.io/crates/didp-yaml`
[6] `https://didppy.readthedocs.io`

condition to prove the optimality or infeasibility when a heuristic search algorithm terminates in Theorem 13 and Corollary 4. Assuming that $x + y \in A$ for $x, y \in A$ and $0 \in A$, by Definition 21, $\langle A, +, 0\rangle$ is a monoid since $x + (y + z) = (x + y) + z$ (associativity) and $x + 0 = 0 + x = x$ (identity) for $x, y, z \in A$. Since $x \leq y \to x + z \leq y + z$ and $x \leq y \to z + x \leq z + y$ for $x, y, z \in A$, A is isotone. In addition, for minimization, if $A \subseteq \mathbb{Q}_0^+$, then $\langle A, +, 0\rangle$ is a cost algebra (Definition 23) since $\forall x \in \mathbb{Q}_0^+, 0 \leq x$, which is one of the sufficient conditions for a heuristic search algorithm to terminate in finite time in Theorem 11. Similarly, assuming that $\max\{x, y\} \in A$ for $x, y \in A$ and $\mathbf{1}$ is the minimum value in A, $\langle A, \max, \mathbf{1}\rangle$ is a monoid since $\max\{x, \max\{y, z\}\} = \max\{\max\{x, y\}, z\}$ (associativity) and $\max\{x, \mathbf{1}\} = \max\{\mathbf{1}, x\} = x$ (identity) for $x, y, z \in A$. Since $x \leq y \to \max\{x, z\} \leq \max\{y, z\}$ and $x \leq y \to \max\{z, x\} \leq \max\{z, y\}$ for $x, y, z \in A$, A is isotone. For minimization, $\langle A, \max, \mathbf{1}\rangle$ is a cost algebra since $\forall x \in A, \mathbf{1} \leq x$.

In all solvers, we use the dual bound function provided with a DyPDL model as a heuristic function. Thus, $f(S) = g(S) \times h(S) = g(S) \times \eta(S)$. By Theorem 14, the best f-value in the open list is a dual bound. In CAASDy, states in the open list are ordered by the f-values in a binary heap, so a dual bound can be obtained by checking the top of the binary heap. Similarly, in DFBnB, CBFS, and ACPS, since states with each depth are ordered by the f-values, by keeping track of the best f-value in each depth, we can compute a dual bound. In APPS, when the set of the best states O_b and the set of the best successor states O_c become empty, the best f-value of states in the suspend list O_s is a dual bound, where states are ordered by the f-values. In DBDFS, we keep track of the best f-value of states inserted into O_1 and use it as a dual bound when O_0 becomes empty. In CABS, based on Theorem 20, the best f-value of discarded states is maintained, and a dual bound is computed after generating all successor states in a layer. In CAASDy, CBFS, ACPS, APPS, and CABS, when the f- and h-values of two states are the same, the tie is broken according to the implementation of the binary heap that is used to implement the open list. In DFBnB and DBDFS, the open list is implemented with a stack, and successor states are sorted before being pushed to the stack, so the tie-breaking depends on the implementation of the sorting algorithm. While a dual bound function is provided in each DyPDL model used in our experiment, it is not required in general; when no dual bound function is provided, the DIDP solvers use the g-value instead of the f-value to guide the search and do not perform pruning.

4.3.2 Benchmarks

We used the ten combinatorial optimization problem classes and their DyPDL models introduced in Section 3.3 in addition to TSPTW introduced in Section 3.2. These models can all be handled by our DIDP solvers. As discussed in Section 4.3.1, given a monoidal DyPDL model with a monoid $\langle A, +, 0\rangle$ or $\langle A, \max, \mathbf{1}\rangle$ where $A \subseteq \mathbb{Q} \cup \{-\infty, \infty\}$ and $\mathbf{1}$ is the minimum value in A, a solver proves the optimality or infeasibility if it terminates. In TSPTW, the capacitated vehicle routing problem (CVRP), and the multi-commodity pickup and delivery traveling salesperson problem (m-PDTSP), the cost expression of each transition is defined as $c_{S[i],j} + x$ with $c_{S[i],j} \geq 0$ (or $c_{S[i],0} + c_{0j} + x$ with $c_{S[i],0}, c_{0j} \geq 0$), the cost of the base case is $c_{S[i],0} \geq 0$, and the objective is minimization, so the models are cost-algebraic with a cost-algebra $\langle \mathbb{Q}_0^+, +, 0\rangle$. Similarly, for $1||\sum w_i T_i$ and talent scheduling, each cost expression adds a nonnegative value to the cost of the successor state (x), and the objective is minimization, so the models are cost-algebraic. For the orienteering problem with time windows (OPTW) and the multi-dimensional knapsack problem (MDKP), the cost expressions

are similar to the above problem classes, so the models are monoidal with a monoid $\langle \mathbb{Q}_+^0, +, 0 \rangle$. However, the DyPDL models are not cost-algebraic since the objective is maximization. For bin packing and SALBP-1, each cost expression adds 1 to the cost of the successor state $(1 + x)$ or keeps it as is (x), which can be viewed as adding 0 $(0 + x)$. Since the objective is minimization, the DyPDL models are cost-algebraic with $\langle \mathbb{Z}_0^+, +, 0 \rangle$. For the minimization of open stacks problem (MOSP) and graph-clear, the cost expression takes the maximum of a nonnegative integer and the cost of the successor state, and the objective is minimization, so the models are cost-algebraic with $\langle \mathbb{Z}_0^+, \max, 0 \rangle$. Since all models are finite and acyclic, by Theorems 10 and 19, our DIDP solvers terminate in finite time.

We describe benchmark instances and MIP and CP models for each problem class. We present all MIP and CP models in Appendix B. All benchmark instances are in text format, so they are converted to YAML-DyPDL problem files by a Python script. All instances in one problem class share the same YAML-DyPDL domain file except for MDKP, where the number of state variables depends on an instance, and thus a domain file is generated for each instance by the Python script.

TSPTW

We use 340 instances from Dumas et al. [109], Gendreau et al. [160], Ohlmann and Thomas [328], and Ascheuer [10], where travel times are integers; while didp-rs can handle floating point numbers, the CP solver we use, CP Optimizer, does not. In these instances, the deadline to return to the depot, b_0, is defined, but $\forall i \in N, b_i + c_{i0} \leq b_0$ holds, i.e., we can always return to the depot after visiting the final customer. Thus, b_0 is not considered in the DyPDL model. For MIP, we use Formulation (1) proposed by Hungerländer and Truden [222]. When there are zero-cost edges, flow-based subtour elimination constraints [157] are added. We adapt a CP model for a single machine scheduling problem with time windows and sequence-dependent setup times [49] to TSPTW, where an interval variable represents the time to visit a customer. We change the objective to the sum of travel costs (setup time in their model) and add a First constraint ensuring that the depot is visited first.

CVRP

We use 207 instances in A, B, D, E, F, M, P, and X sets from CVRPLIB [425]. We use a MIP model proposed by Gadegaard and Lysgaard [151] and a CP model proposed by Rabbouch, Saâdaoui, and Mraihi [350].

m-PDTSP

We use 1178 instances from Hernández-Pérez and Salazar-González [208], which are divided into class1, class2, and class3 sets. For MIP, we use the MCF2C+IP formulation [287]. We use the CP model proposed by Castro, Cire, and Beck [65]. In all models, unnecessary edges are removed by a preprocessing method [287].

OPTW

We use 144 instances from Righini and Salani [356, 357], Montemanni and Gambardella [315], and Vansteenwegen et al. [434]. In these instances, service time s_i spent at each customer i is defined,

so we incorporate it in the travel time, i.e., we use $s_i + c_{ij}$ as the travel time from i to j. We use the MIP model described in Vansteenwegen, Souffriau, and Oudheusden [433]. For CP, we develop a model similar to that of TSPTW, described in Appendix B.

MDKP

We use 276 instances from OR-Library [23], excluding one instance that has fractional item weights; while the DIDP solvers can handle fractional weights, the CP solver does not. We use the MIP model described in Cacchiani et al. [61]. For CP, we develop a model using the Pack global constraint [391] for each dimension (see Appendix B).

Bin Packing

We use 1615 instances in BPPLIB [100], proposed by Falkenauer [130] (Falkenauer U and Falkenauer T), Scholl, Klein, and Jürgens [384] (Scholl 1, Scholl 2, and Scholl 3), Wäscher and Gau [441] (Wäscher), Schwerin and Wäscher [385] (Schwerin 1 and Schwerin 2), and Schoenfield [382] (Hard28). We use the MIP model by Martello and Toth [306] extended with inequalities ensuring that bins are used in order of index and item j is packed in the j-th bin or earlier as described in Delorme, Iori, and Martello [99]. We implement a CP model using Pack while ensuring that item j is packed in bin j or before. For MIP and CP models, the upper bound on the number of bins is computed by the first-fit decreasing heuristic. We show the CP model in Appendix B.

SALBP-1

We use 2100 instances proposed by Morrison, Sewell, and Jacobson [319]. For MIP, we use the NF4 formulation [363]. We use a CP model proposed by Bukchin and Raviv [54] but implement it using the global constraint Pack in CP Optimizer as it performs better than the original model (see Appendix B). In addition, the upper bound on the number of stations is computed in the same way as the MIP model instead of using a heuristic.

Single Machine Total Weighted Tardiness

We use 375 instances in OR-Library [23] with 40, 50, and 100 jobs. For MIP, we use the formulation with assignment and positional date variables (F4) [248]. For CP, we formulate a model using interval variables and precedence constraints, as described in Appendix B. We extract precedence relations between jobs using the method proposed by Kanet [237] and incorporate them into the DP and CP models but not into the MIP model as its performance is not improved.

Talent Scheduling

Garcia de la Banda and Stuckey [152] considered instances with $8, 10, 12, 14, 16, 18, 20, 22$ actors and $16, 18, ..., 64$ scenes, resulting in 200 configurations in total. For each configuration, they randomly generated 100 instances. We use the first five instances for each configuration, resulting in 1000 instances in total. We use a MIP model described in Qin et al. [349]. For CP, we extend the model used in Chu and Stuckey [78] with the AllDifferent global constraint [280], which is redundant but slightly improves the performance in practice, as described in Appendix B. In all models, a problem is simplified by preprocessing as described in Garcia de la Banda and Stuckey [152].

MOSP

We use 570 instances in Constraint Modelling Challenge [402], SCOOP Project,[7] Faggioli and Bentivoglio [129], and Chu and Stuckey [79]. The MIP and CP models are proposed by Martin, Yanasse, and Pinto [307]. From their two MIP models, we select MOSP-ILP-I as it solves more instances optimally in their paper.

Graph-Clear

We generated 135 instances using planar and random graphs in the same way as Morin et al. [317], where the number of nodes in a graph is 20, 30, or 40. All instances and the instance generator are available in our repository.[8] For planar instances, we use a planar graph generator [150] with the input parameter of 1000. We use MIP and CP models proposed by Morin et al. [317]. From the two proposed CP models, we select CPN as it solves more instances optimally.

4.3.3 Metrics

Since all solvers are exact, we evaluate *coverage*, the number of instances where an optimal solution is found and its optimality is proved within time and memory limits. We include the number of instances where infeasibility is proved in coverage. We also evaluate *optimality gap*, a relative difference between primal and dual bounds. Let $\overline{\gamma}$ be a primal bound and $\underline{\gamma}$ be a dual bound found by a solver. We define the optimality gap, $\delta\left(\overline{\gamma}, \underline{\gamma}\right)$, as follows:

$$\delta\left(\overline{\gamma}, \underline{\gamma}\right) = \begin{cases} 0 & \text{if } \overline{\gamma} = \underline{\gamma} = 0 \\ \frac{|\overline{\gamma} - \underline{\gamma}|}{\max\left\{|\overline{\gamma}|, |\underline{\gamma}|\right\}} & \text{else.} \end{cases}$$

The optimality gap is 0 when the optimality is proved and positive otherwise. We also use 0 as the optimality gap when the infeasibility is proved. In the second line, when the signs of the primal and dual bounds are the same, since $|\overline{\gamma} - \underline{\gamma}| \leq \max\{|\overline{\gamma}|, |\underline{\gamma}|\}$, the optimality gap never exceeds 1. In practice, we observe that primal and dual bounds found are always nonnegative in our experiment. Therefore, we use 1 as the optimality gap when either a primal or dual bound is not found.

To evaluate the performance of anytime solvers, we use the *primal integral* [39], which considers the balance between the solution quality and computational time. For an optimization problem, let σ^t be a solution found by a solver at time t, σ^* be an optimal (or best-known) solution, and γ be a function that returns the solution cost. The *primal gap* function p is

$$p(t) = \begin{cases} 0 & \text{if } \gamma(\sigma^t) = \gamma(\sigma^*) = 0 \\ 1 & \text{if no } \sigma^t \text{ or } \gamma(\sigma^t)\gamma(\sigma^*) < 0 \\ \frac{\left|\gamma(\sigma^*) - \gamma(\sigma^t)\right|}{\max\{|\gamma(\sigma^*)|, |\gamma(\sigma^t)|\}} & \text{else.} \end{cases}$$

The primal gap takes a value in $[0, 1]$, and lower is better. Let $t_i \in [0, T]$ for $i = 1, ..., l-1$ be a time point when a new better solution is found by a solver, $t_0 = 0$, and $t_l = T$. The primal integral is defined as $P(T) = \sum_{i=1}^{l} p(t_{i-1}) \cdot (t_i - t_{i-1})$. It takes a value in $[0, T]$, and lower is better. $P(T)$

[7] https://cordis.europa.eu/project/id/32998
[8] https://github.com/Kurorororo/didp-models

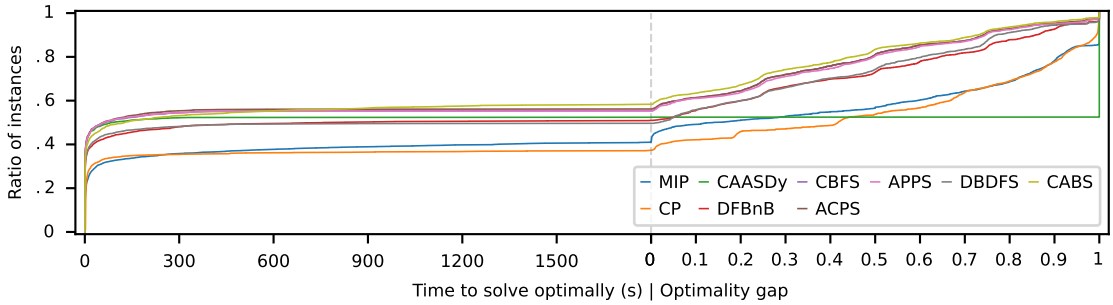


Figure 4.1: The ratio of the coverage against time and the ratio of instances against the optimality gap averaged over all problem classes.

Table 4.1: Coverage (c.) and the number of instances where the memory limit is reached (m.) in each problem class. The coverage of a DIDP solver is in bold if it is higher than MIP and CP, and the higher of MIP and CP is in bold if there is no better DIDP solver. The highest coverage is underlined.

	MIP		CP		CAASDy		DFBnB		CBFS		ACPS		APPS		DBDFS		CABS		CABS/0	
	c.	m.	c.	m.	c.	m.	c.	m.	c.	m.	c.	m.	c.	m.	c.	m.	c.	m.	c.	m.
TSPTW (340)	222	0	47	0	**257**	83	**242**	34	**257**	81	**257**	82	**257**	83	**256**	83	**259**	0	**259**	0
CVRP (207)	**27**	5	0	0	6	201	6	187	6	201	6	201	6	201	6	201	6	0	5	3
m-PDTSP (1178)	940	0	**1049**	0	952	226	985	193	988	190	988	190	988	190	987	191	1035	0	988	15
OPTW (144)	16	0	49	0	**64**	79	**64**	60	**64**	80	**64**	80	**64**	80	**64**	78	**64**	0	-	-
MDKP (276)	**165**	0	6	0	4	272	4	272	5	271	5	271	5	271	4	272	5	1	-	-
Bin Packing (1615)	1159	0	**1234**	0	922	632	526	1038	1115	431	1142	405	1037	520	426	1118	1167	4	242	14
SALBP-1 (2100)	1423	248	1584	0	**1657**	406	**1629**	470	1484	616	**1626**	474	**1635**	465	1404	696	**1802**	0	1204	1
$1\|\|\sum w_i T_i$ (2100)	106	0	150	0	**270**	105	**233**	8	**272**	103	**272**	103	**265**	110	**268**	107	**288**	0	-	-
Talent Scheduling (1000)	0	0	0	0	**207**	793	**189**	388	**214**	786	**214**	786	**206**	794	**205**	795	**239**	0	**231**	0
MOSP (570)	231	2	437	0	**483**	87	**524**	46	**523**	47	**524**	46	**523**	47	**522**	48	**527**	0	-	-
Graph-Clear (135)	17	0	4	0	**78**	57	**99**	36	**101**	34	**101**	34	**99**	36	**82**	53	**103**	19	-	-

decreases if the same solution cost is achieved faster or a better solution is found with the same computational time. When an instance is proved to be infeasible at time t, we use $p(t) = 0$, so $P(T)$ corresponds to the time to prove infeasibility. For TSPTW, CVRP, and $1||\sum w_i T_i$, we use the best-known solutions provided with the instances to compute the primal gap. For other problems, we use the best solutions found by the solvers evaluated.

4.3.4 Experimental Settings

We use Rust 1.70.0 for didp-rs and Python 3.10.2 for the Python scripts to convert instances to YAML-DyPDL files and the Python interfaces of Gurobi and CP Optimizer. All experiments are performed on an Intel Xeon Gold 6418 processor with a single thread, an 8 GB memory limit, and a 30-minute time limit using GNU Parallel [416].

4.3.5 Results

We visualize the coverage and optimality gap in Figure 4.1. In the left-hand side, the x-axis is time in seconds, and the y-axis is the ratio of coverage over the number of instances achieved with x seconds. In the right-hand side, the x-axis is the optimality gap, and the y-axis is the ratio of instance where the optimality gap is less than or equal to x. Since each problem class has a different number of instances, we compute the coverage ratio per problem class and then present the mean coverage ratio over problem classes in Figure 4.1. We present similar plots for each problem class

Table 4.2: Average optimality gap in each problem class. The optimality gap of a DIDP solver is in bold if it is lower than MIP and CP, and the lower of MIP and CP is in bold if there is no better DIDP solver. The lowest optimality gap is underlined.

	MIP	CP	CAASDy	DFBnB	CBFS	ACPS	APPS	DBDFS	CABS	CABS/0
TSPTW (340)	0.2394	0.7175	0.2441	**0.1598**	**0.1193**	**0.1194**	**0.1217**	**0.1408**	**<u>0.1151</u>**	**0.2085**
CVRP (207)	0.8696	0.9868	0.9710	**0.7484**	**0.7129**	**0.7123**	**0.7164**	**0.7492**	**<u>0.6912</u>**	0.9111
m-PDTSP (1178)	0.1820	**<u>0.1095</u>**	0.2746	0.2097	0.1807	0.1807	0.1840	0.2016	0.1599	0.1878
OPTW (144)	0.6643	0.2890	0.5556	0.3583	**<u>0.2683</u>**	**<u>0.2683</u>**	**0.2778**	0.3359	**0.2696**	-
MDKP (276)	**<u>0.0008</u>**	0.4217	0.9855	0.4898	0.4745	0.4745	0.4742	0.4854	0.4676	-
Bin Packing (1615)	0.0438	**<u>0.0043</u>**	0.4291	0.0609	0.0083	0.0075	0.0105	0.0651	0.0049	0.7386
SALBP-1 (2100)	0.2704	0.0108	0.2100	0.0257	0.0115	**0.0096**	**0.0094**	0.0273	**<u>0.0057</u>**	0.3695
$1\|\|\sum w_iT_i$ (2100)	0.4897	0.3709	**0.2800**	0.3781	**0.2678**	**0.2679**	**0.2878**	**0.2845**	**<u>0.2248</u>**	-
Talent Scheduling (1000)	0.8869	0.9509	**0.7930**	**0.2368**	**0.1884**	**0.1884**	**0.2003**	**0.2462**	**<u>0.1697</u>**	**0.6667**
MOSP (570)	0.3254	0.1931	**0.1526**	**0.0713**	**0.0362**	**0.0359**	**0.0392**	**0.0655**	**<u>0.0200</u>**	-
Graph-Clear (135)	0.4744	0.4560	**0.4222**	**0.2359**	**0.0995**	**0.0996**	**0.1089**	**0.2636**	**<u>0.0607</u>**	-

in Appendix C and summarize the results by showing the coverage and the average optimality gap at the time limit in each problem class in Tables 4.1 and 4.2. Table 4.1 also presents the number of instances where the memory limit is reached. In the tables, if a DIDP solver is better than MIP and CP, it is emphasized in bold. If MIP or CP is better than all DIDP solvers, its value is in bold. The best value is underlined.

On average, the DIDP solvers solve a larger ratio of instances than MIP and CP. Except for CAASDy, they also outperform MIP and CP in the optimality gap. Since we use the dual bound function as an admissible heuristic function, by Theorem 15, CAASDy does not find a feasible solution before proving the optimality when a model is cost-algebraic and the costs of all base cases are the identity element $\mathbf{1}$, e.g., 0 for rational numbers under addition. Since the DyPDL models for bin packing, SALBP-1, $1||\sum w_iT_i$, talent scheduling, MOSP, and graph-clear satisfy these conditions, CAASDy does not find feasible solutions for any unsolved instances. In addition, although the DyPDL models for TSPTW, CVRP, OPTW, and MDKP do not satisfy the conditions,[9] CAASDy does not find feasible solutions for unsolved instances in practice. In these problem instances, since primal bounds are not provided, the optimality gap is 1.0. The only exception is m-PDTSP, where CAASDy finds feasible solutions for eleven unsolved instances.

Comparing the DIDP solvers, CABS is the best on average. CABS has lower coverage with low run-time but eventually outperforms other DIDP solvers. Since CABS repeatedly searches the same states multiple times, it is less time efficient than other DIDP solvers. However, as shown in Table 4.1, while other DIDP solvers run out of memory in most of the instances they are unable to solve, CABS rarely reaches the memory limit, benefitting from the layered duplicate detection. As a result, CABS has the highest coverage on average and in each problem class at the time limit.

CAASDy, ACPS, APPS, and CABS have higher coverage than both MIP and CP in seven problem classes: TSPTW, OPTW, SALBP-1, $1||\sum w_iT_i$, talent scheduling, MOSP, and graph-clear. In addition, the DIDP solvers except for CAASDy outperform MIP and CP in the class1 instances of m-PDTSP (145 (CABS) and 144 (others) vs. 128 (MIP and CP)). MIP has the highest coverage in CVRP and MDKP, and CP in m-PDTSP and bin packing. MIP runs out of memory in some instances while CP never does. In particular, in the MIP model for SALBP-1, the number of decision variables and constraints is quadratic in the number of tasks in the worst case, and MIP reaches the memory limit in 248 instances with 1000 tasks.

[9]The DyPDL models for OPTW and MDKP are not cost-algebraic. The models for TSPTW, CVRP, and m-PDTSP are cost-algebraic with the identity element 0, but the costs of their base cases can be positive.

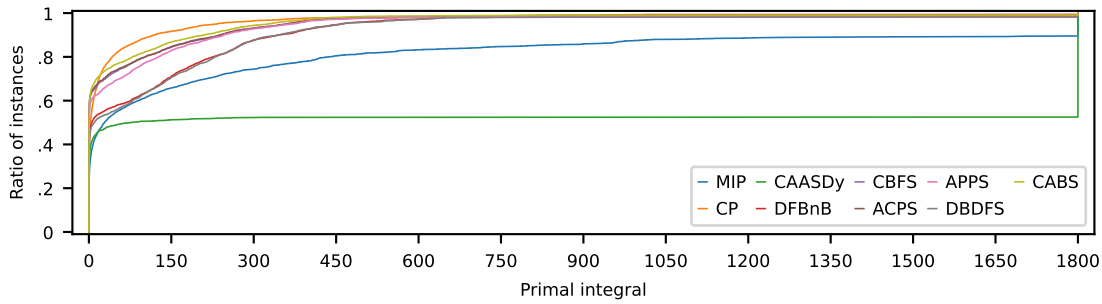


Figure 4.2: The ratio of instances against the primal integral averaged over all problem classes.

Table 4.3: Average primal integral in each problem class. The primal integral of a DIDP solver is in bold if it is lower than MIP and CP, and the lower of MIP and CP is in bold if there is no better DIDP solver. The lowest primal integral is underlined.

	MIP	CP	CAASDy	DFBnB	CBFS	ACPS	APPS	DBDFS	CABS	CABS/0
TSPTW (340)	509.18	48.97	458.26	**46.31**	**9.49**	**10.06**	**29.36**	56.65	**9.25**	**13.71**
CVRP (207)	1147.11	482.89	1748.23	**420.66**	**423.45**	**418.29**	**440.59**	523.42	**333.68**	**335.75**
m-PDTSP (1178)	177.82	26.04	333.53	**23.37**	**6.51**	**6.49**	**9.23**	**17.87**	**5.24**	**5.31**
OPTW (144)	443.70	**15.58**	1018.23	175.49	54.05	54.29	74.37	139.64	57.95	-
MDKP (276)	**0.11**	15.87	1773.92	236.13	211.70	211.63	211.85	238.01	201.73	-
Bin Packing (1615)	88.45	8.07	778.60	104.49	10.00	8.43	13.85	111.87	**5.07**	11.58
SALBP-1 (2100)	536.39	28.43	383.35	35.59	**10.83**	**7.28**	**6.74**	38.80	**1.92**	**19.42**
$1\|\|\sum w_i T_i$ (2100)	64.01	**3.49**	513.24	136.99	111.19	103.76	105.97	97.34	71.21	-
Talent Scheduling (1000)	111.75	**18.86**	1435.12	118.98	40.67	40.34	60.07	143.40	25.36	50.73
MOSP (570)	92.63	13.01	275.48	**4.41**	**1.39**	**1.20**	**1.37**	**7.72**	**0.31**	-
Graph-Clear (135)	337.61	83.49	764.00	**4.63**	**0.70**	**0.74**	**3.45**	87.90	**0.37**	-

The DIDP solvers closed five class1 instances of m-PDTSP that were previously unsolved. All DIDP solvers optimally solve p43.3Q6max (the optimal solution cost is 56600) and p43.3Q15max5 (56325) and prove the infeasibility of p43.3Q4max1 and p43.3Q10max5. In addition, CABS closed p43.3Q7max1 (56120). CP also closed p43.3Q8max1 (29450) and p43.3Q20max5 (29475).

In terms of the optimality gap shown in Table 4.2, ACPS, APPS, and CABS are better than MIP and CP in the seven problem classes where they have higher coverage. In addition, the DIDP solvers except for CAASDy outperform MIP and CP in CVRP, where MIP has the highest coverage; in large instances of CVRP, MIP fails to find feasible solutions, which results in a high average optimality gap. Comparing the DIDP solvers, CABS is the best in all problem classes except for OPTW, where CBFS and ACPS are marginally better. CAASDy is the worst among the DIDP solvers in all problem classes except for $1||\sum w_i T_i$;

Figure 4.2 shows the ratio of instances against the primal integral averaged over all problem classes. We present similar plots for each problem class in Appendix C. On average, the DIDP solvers except for CAASDy outperform MIP, and CABS is the best among the DIDP solvers. As mentioned above, CAASDy does not find feasible solutions for almost all unsolved instances, resulting in the worst primal integral. CP is the best though CBFS, ACPS, APPS, and CABS have a higher ratio of instances than CP when the primal integral is less than a certain threshold. For example, the lines for CABS and CP cross where the primal integral is around 20.

Table 4.3 shows the average primal integral in each problem class. Similar to Tables 4.1 and 4.2, the primal integral of a DIDP solver is in bold if it is better than MIP and CP, and the best value is underlined. Comparing the DIDP solvers, CABS is the best in all problem classes except for OPTW, where CBFS and ACPS are better. CBFS, ACPS, and APPS outperform MIP and CP in

six problem classes (TSPTW, CVRP, m-PDTSP, SALBP-1, MOSP, and graph-clear). In addition to these problem classes, CABS achieves a better primal integral than CP in bin packing. In contrast, CP is the best in OPTW, $1||\sum w_i T_i$, and talent scheduling. While the number of problem classes where CABS is better than CP on average is larger than the number of classes where CP is better, large performance gaps in MDKP and $1||\sum w_i T_i$ (Figures C.16 and C.19 in Appendix C) result in a better the mean for CP in Figure 4.2.

In CVRP, while MIP solves more instances optimally, the DIDP solvers except for CAASDy achieve a better average primal integral since MIP fails to find primal bounds for large instances as mentioned. In m-PDTSP, the DIDP solvers except for CAASDy have a lower primal integral than CP, which has the highest coverage. In contrast, in OPTW, $1||\sum w_i T_i$, and talent scheduling, where the DIDP solvers solve more instances, CP has a better primal integral.

4.3.6 Performance of DIDP Solvers and Problem Characteristics

In CVRP, MIP outperforms the DIDP solvers even though the DyPDL model is similar to other routing problems (i.e., TSPTW, m-PDTSP, and OPTW) where the DIDP solvers are better. Similarly, in bin packing, the DyPDL model is similar to that of SALBP-1, where CABS is the best, but CP solves more instances. One common feature in the subset of the problem classes where DIDP is better than MIP or CP is a sequential dependency. In TSPTW and OPTW, a solution is a sequence of visited customers, the time when a customer is visited depends on the partial sequence of customers visited before, and time window constraints restrict possible sequences. Similarly, in m-PDTSP, SALBP-1, and $1||\sum w_i T_i$, precedence constraints restrict possible sequences. In the DyPDL models, these constraints restrict possible paths in the state transition graphs and thus reduce the number of generated states. In contrast, in CVRP, as long as the capacity constraint is satisfied for each vehicle, customers can be visited in any order. In the DyPDL model for bin packing, while item i must be packed in the i-th or earlier bin and the first item packed in a new bin has the minimum index among possible items, the remaining items can be packed in any order. We hypothesize that this difference, whether a sequential dependency exists or not, may be important for the performance of the DIDP solvers observed in our experiments.

4.3.7 Evaluating the Importance of Dual Bound Functions

As we described above, our DIDP solvers use the dual bound function defined in a DyPDL model as an admissible heuristic function, which is used for both search guidance and state pruning. In $1||\sum w_i T_i$, MOSP, and graph-clear, we use a trivial dual bound function, which always returns 0. Nevertheless, the DIDP solvers show better performance than MIP and CP. This result suggests that representing these problems using state transition systems provides a fundamental advantage on these problems while raising the question of the impact of non-trivial dual bound functions on problem solving performance. To investigate, we evaluate the performance of CABS with DyPDL models where the dual bound function is replaced with a function that always returns 0. In other words, beam search keeps the best b states according to the g-values and prunes a state S if $g(S) \geq \overline{\gamma}$ in minimization, where $\overline{\gamma}$ is a primal bound. Since the zero dual bound function is not valid for OPTW and MDKP, where the DyPDL models maximize the nonnegative total profit, we use only TSPTW, CVRP, m-PDTSP, bin packing, SALBP-1, and talent scheduling. We call this configuration

CABS/0 and show results in Tables 4.1–4.3. In TSPTW, CABS and CABS/0 have the same coverage, and CABS/0 reaches it slightly faster as shown in Figure C.1 in Appendix C, possibly due to less expensive computation per state. However, in other problem classes, CABS/0 has a lower coverage than CABS. In terms of the optimality gap and primal integral, CABS is better than CABS/0 in all problem classes, and the difference is particularly large in bin packing and SALBP-1. Overall, using the trivial dual bound function results in performance degradation in proving optimality and achieving good anytime behavior.

4.3.8 Comparison with Other State-Based Approaches

In Appendix C.2, we compare DIDP with other state-based approaches, domain-independent AI planning and Picat, a logic programming language hybridized with AI planning (Section 2.4.2). They are not competitive with DIDP, which is not surprising as AI planners are not designed for combinatorial optimization. In addition, our evaluation setting is not necessarily the best for these approaches: while we use models adapted from the DyPDL models, different models might be better for these solvers. For domain-independent AI planning, while we use the winners of the most recent International Planning Competition,[10] they are not necessarily the most efficient planners for combinatorial optimization. However, our point is to show that the performance achieved by the DIDP solvers is not a trivial consequence of the state-based modeling approach, and DIDP is doing something that existing approaches are not able to easily do.

In the appendix, we also compare DIDP with a decision diagram solver, ddo [171] (Section 2.4.4) using TSPTW (with a different objective function) and talent scheduling, for which the models for ddo were developed by previous work [169, 87, 88, 89]. DIDP is better than ddo in terms of coverage in both problem classes, but ddo has a better average optimality gap in talent scheduling. Note that ddo requires the specification of a state merging operator which is ignored by DIDP. Thus, the models being solved are not identical.

4.4 Summary

We introduced a generic heuristic search algorithm and proved its completeness and optimality for a subclass of Dynamic Programming Description Language (DyPDL) models. While our algorithm is based on an existing framework, cost-algebraic heuristic search, it has the following new features: it exploits dominance between states for pruning; it is anytime, i.e., can provide feasible solutions and optimality gap during search; it is not restricted to cost algebras if a model is finite and acyclic.

Using existing heuristic search algorithms, we developed seven domain-independent dynamic programming (DIDP) solvers. We experimentally showed that they outperform commercial mixed-integer programming (MIP) and constraint programming (CP) solvers in seven out of eleven combinatorial optimization problem classes evaluated. This result demonstrates that DIDP is a promising model-based paradigm.

Among the DIDP solvers, complete anytime beam search (CABS) is the best in almost all cases; while other DIDP solvers run out of the 8GB memory limit before reaching the 30-minute time limit in most instances, CABS rarely reaches the memory limit, benefitting from the layered duplicate

[10] `https://ipc2023.github.io/`

detection mechanism. This observation raises the importance of memory efficiency in developing heuristic search solvers.

Using CABS/0, we evaluated the importance of a dual bound function, which is used for both search guidance and state pruning. The result shows that using a better dual bound function is essential to achieve better performance in terms of both finding good solutions and proving optimality. Nevertheless, using a dual bound function for search guidance is not necessarily justified as discussed in Section 4.1.4; we may improve the anytime behavior by using an inadmissible heuristic function for search guidance. Disentangling search guidance from pruning and developing better functions for each role is an interesting direction for future work.

Chapter 5

Large Neighborhood Beam Search

In the previous chapter, we demonstrated that heuristic search algorithms, techniques developed in the artificial intelligence (AI) community, can be successfully applied to domain-independent dynamic programming (DIDP). In this chapter, we investigate applying a method used in the constraint programming (CP) and operations research (OR) communities to DIDP. In particular, we focus on large neighborhood search (LNS) [392], an algorithmic framework originally developed in the CP community. An LNS algorithm removes a part of a solution and then performs search in the induced partial search space (neighborhood) to find a better solution, typically using tree search. LNS has been applied to combinatorial optimization problems such as routing [392, 368, 234] and scheduling [93, 332]. LNS algorithms are also used in general-purpose solvers for CP [347, 277, 337] and mixed-integer programming (MIP) [94, 40, 205, 75] as primal heuristics, algorithms to quickly obtain high-quality feasible solutions.

We propose large neighborhood beam search (LNBS), a combination of LNS and state space search designed to achieve high solution quality. LNBS tries to improve a solution path by removing a partial path between two states and then performing beam search to find a better partial path. While LNBS has the freedom to select a neighborhood (i.e., a partial path to remove), we propose a strategy that dynamically adjusts the size of a neighborhood based on a multi-armed bandit problem [38]. With our strategy, LNBS is complete, i.e., it finds and proves an optimal solution given enough time, but, of course, it is aimed at problems where its solution quality is more important than proved optimality.

The novelty of our methodology comes from the adaptive neighborhood selection strategy using a multi-armed bandit algorithm. Although some existing methods can be considered state space search algorithms in a neighborhood [352, 149, 324, 170], they do not adaptively select the neighborhood size. Unlike previous work using multi-armed bandit algorithms with LNS [205, 75, 338], we use a budgeted multi-armed bandit algorithm [442], which tries to maximize the total reward within a time limit.

We implement multiple configurations of LNBS as DIDP solvers. The experimental results show that the LNBS configurations outperform complete anytime beam search (CABS), the best DIDP solver developed in Chapter 4, in six out of the eleven benchmark problem classes in terms of solution quality. In addition, LNBS performs better than a commercial CP solver, which uses LNS [277], in seven problems while CABS is better than CP in only six problems. Since LNBS performs better

than CABS in some problem classes while CABS is better in others, we investigate the reason for this performance. Our analysis suggests that LNBS tends to perform better than CABS when a heuristic function is not informative.

After introducing LNS in Section 5.1, we propose LNBS in Section 5.2 and experimentally evaluate it in Section 5.3. We discuss future directions to improve LNBS in Section 5.4 and similarities and differences between LNBS and existing state space search and LNS algorithms in Section 5.5. Finally, Section 5.6 summarizes the contributions of this chapter.

This work is based on a conference paper published in the *International Conference on Principles and Practice of Constraint Programming* [268]. We extend the paper with the following points:

- We generalize LNBS to maximization.
- We add two additional problem classes in the experimental evaluation.
- We conduct an ablation study of LNBS components using all of the eleven problem classes.
- We propose a new hypothesis that LNBS performs better than CABS when a heuristic function is not informative. Our experimental results are consistent with this hypothesis while they contradict our previous hypothesis, presented in the conference paper, that LNBS performs better when partial path costs are diverse.
- We provide a broader and deeper literature review of related work.

5.1 Large Neighborhood Search

Large neighborhood search (LNS) iteratively removes a part of a solution and solves the resulting subproblem (neighborhood) [392]. We present generic pseudo-code of LNS for minimization following Pisinger and Ropke [343] in Algorithm 12. Let σ be a solution for a problem, and $\gamma(\sigma)$ be its cost. Let D be a *destroy heuristic*, a possibly randomized algorithm that returns a partial solution $D(\sigma)$ given a complete solution σ, and R be a *repair heuristic*, an algorithm that returns a complete solution $R(\psi')$ given a partial solution ψ'. At each iteration, we use the destroy and repair heuristics to obtain a new solution σ' from the current solution σ. While we update the best solution $\overline{\sigma}$ only if a better solution is found, σ can be updated according to a different criterion, represented by a function $A(\sigma', \sigma)$. For example, simulated annealing [252], which accepts non-improving solutions with probabilities decreasing over time, is used by previous work [368].

In MIP and CP, a solution is represented as a complete value assignment to decision variables. Let $x_1, ..., x_n$ be decision variables and $v_1, ..., v_n$ be values of the variables representing a solution. In LNS algorithms for MIP and CP, a subproblem is created by fixing the values of a subset of decision variables [94, 40, 205, 75, 392, 234, 332, 347, 277]. In other words, the subproblem is to assign values to a subset of decision variables $\{x_j \mid j \in Y\}$ given a partial value assignment $x_j = v_j$ for $j \notin Y$, where $Y \subset \{1, ..., n\}$. A destroy heuristic selects the subset of decision variables Y. A repair heuristic is typically a tree search algorithm: a search node is a partial value assignment to decision variables, successor nodes are generated by assigning a value to an unassigned variable, and a solution corresponds to a leaf node, where all variables are assigned values.

Algorithm 12 Generic pseudo-code of LNS for minimization. A feasible solution $\overline{\sigma}$ is given as input.

```
σ ← σ̄                                          ▷ Initialize the current solution
while the stopping criterion is not met do
    ψ ← D(σ)                                    ▷ Remove a part of the solution.
    σ' ← R(ψ)                                   ▷ Find a new solution.
    if A(σ', σ) then
        σ ← σ'                                  ▷ Update the current solution.
    if γ(σ') ≤ γ(σ̄) then
        σ̄ ← σ                                   ▷ Update the best solution.
return σ̄                                        ▷ Return the best solution.
```

Algorithm 13 Restrict DD nodes (states) in a layer in DD-LNS. A set of states G, a function σ returning the best sequence of transitions from the target state to each state, the transition $\overline{\tau}$ in the best solution, and parameters $b \geq 0$ and $p \in [0, 1)$ are given as input.

```
O ← ∅
for S ∈ G do
    if σ(S)_{|σ(S)|} = τ̄ or r ≤ p where r ∼ [0, 1) then
        O ← O ∪ {S}
if |O| < b then
    O ← O∪ the best b − |O| states in G \ O.
return O
```

5.1.1 Large Neighborhood Search with Decision Diagrams

Gillard and Schaus [170] proposed LNS with decision diagrams (DD-LNS), which uses a DP formulation of a problem as input. Although DD-LNS was not explicitly framed as state space search, the authors acknowledged that DD-LNS is a hybridization of LNS and beam search [170]. Therefore, we interpret it as a state space search algorithm. Given a sequence of transitions $\langle \sigma_1, ..., \sigma_n \rangle$, DD-LNS keeps the first d transitions, $\sigma_{:d} = \langle \sigma_1, ..., \sigma_d \rangle$, and searches for the remaining transitions. To find such a sequence, DD-LNS constructs a decision diagram (DD), a directed graph where nodes are partitioned into layers (Section 2.2.8). In the constructed DD, each node corresponds to a state, and each edge corresponds to a transition, so it is a state transition graph. The first layer in the DD contains only the node corresponding to $S^0[\![\sigma_{:d}]\!]$, the state resulting from applying the sequence $\sigma_{:d}$ to the target state S^0. DD-LNS iteratively compiles a layer by applying transitions to the states in the current layer.

When compiling a DD, DD-LNS prunes a node based on bounds. For each node, a lower bound on the path cost via that node is computed. If this lower bound is greater than or equal to the current best solution cost, the node is removed from the DD as it does not lead to a better solution.[1] While this lower bound could be computed by constructing a relaxed DD from the node (see Section 2.2.8), doing so for each node is time-consuming. DD-LNS uses computationally less expensive lower bound functions for this purpose, e.g., the size of the minimum spanning tree for routing problems. The lower bounds used here are called rough lower bounds (RLBs) in previous work [169, 170] as they are potentially rougher than the lower bound obtained from relaxed DDs. This procedure can be viewed as pruning based on the dual bound used by heuristic search algorithms for Dynamic

[1] DD-LNS was originally proposed for minimization.

Algorithm 14 Compiling a restricted DD from a state $\hat{S}$ in DD-LNS for minimization with a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$ with $\langle A, \times, \mathbf{1} \rangle$. An approximate dominance relation $\preceq_a$, a dual bound function η, the best $\hat{S}$-solution $\hat{\sigma}$, a beam width b, and a probability p are given as input.

if $\hat{S} \not\models \mathcal{C}$ **then return** NULL, $\top$
$\overline{\sigma} \leftarrow$ NULL, $\overline{\gamma} \leftarrow \mathsf{cost}_{\hat{\sigma}}(\hat{S})$, complete $\leftarrow \top$ ▷ Initialize the solution.
$l \leftarrow 0$, $\sigma^l(\hat{S}) \leftarrow \langle\rangle$, $g^l(\hat{S}) \leftarrow \mathbf{1}$ ▷ Initialize the g-value.
$O \leftarrow \{\hat{S}\}$ ▷ Initialize the open list.
while $O \neq \emptyset$ and $\overline{\sigma} =$ NULL **do**
    $G \leftarrow \emptyset$ ▷ Initialize the set of states.
    for all $S \in O$ **do**
        if $\exists B \in \mathcal{B}, S \models C_B$ **then**
            current_cost $\leftarrow g^l(S) \times \min_{B \in \mathcal{B}: S \models C_B} \mathsf{cost}_B(S)$ ▷ Compute the solution cost.
            if current_cost $< \overline{\gamma}$ **then**
                $\overline{\gamma} \leftarrow$ current_cost, $\overline{\sigma} \leftarrow \sigma^l(S)$ ▷ Update the best solution.
        else
            for all $\tau \in \mathcal{T}(S) : S[\![\tau]\!] \models \mathcal{C}$ **do**
                $g_{\mathrm{current}} \leftarrow g^l(S) \times w_\tau(S)$ ▷ Compute the g-value.
                if $\nexists S' \in G$ such that $S[\![\tau]\!] \preceq_a S'$ and $g_{\mathrm{current}} \geq g^{l+1}(S')$ **then**
                    if $g_{\mathrm{current}} \times \eta(S[\![\tau]\!]) < \overline{\gamma}$ **then**
                        if $\exists S' \in G$ such that $S' \preceq_a S[\![\tau]\!]$ and $g_{\mathrm{current}} \leq g^{l+1}(S')$ **then**
                            $G \leftarrow G \setminus \{S'\}$ ▷ Remove a dominated state.
                        $\sigma^{l+1}(S[\![\tau]\!]) \leftarrow \langle \sigma^l(S); \tau \rangle$, $g^{l+1}(S[\![\tau]\!]) \leftarrow g_{\mathrm{current}}$ ▷ Update the g-value.
                        $G \leftarrow G \cup \{S[\![\tau]\!]\}$ ▷ Insert the successor state.
    $l \leftarrow l + 1$ ▷ Proceed to the next layer.
    $O \leftarrow \{S \in G \mid g^l(S) \times \eta(S) < \overline{\gamma}\}$ ▷ Prune states by the bound.
    if $|O| > b$ **then**
        $O \leftarrow$ Restrict(O, σ^l, $\hat{\sigma}_l$, b, p), complete $\leftarrow \bot$ ▷ Execute Algorithm 13.
if complete and $O \neq \emptyset$ **then**
    complete $\leftarrow \bot$ ▷ A better solution may exist.
return $\overline{\sigma}$, complete ▷ Return the solution.

Programming Description Language (DyPDL) (Section 4.1.4): in minimization, a state S is pruned if $g^l(S) \times \eta(S) \geq \overline{\gamma}$ where $g^l(S)$ is the path cost to reach S, $\eta(S)$ is the dual bound, a lower bound on the path cost from S, $\times$ is a binary operator in a monoid such as $+$ (Definition 21 in Section 4.1.2), and $\overline{\gamma}$ is the current best solution cost.

Even with pruning based on RLBs, constructing an exact DD is usually intractable, so DD-LNS compiles a restricted DD, which keeps only a subset of states of the exact DD. In constructing a restricted DD, DD-LNS keeps $\sigma^l(S)$, the best path to reach each state S in the l-th layer. When selecting states in the l-th layer, given a solution $\hat{\sigma}$, a state S is kept if $\sigma^l(S)_l = \hat{\sigma}_l$, i.e, the last transition to reach S is the same as the l-th transition of the current best solution. Otherwise, S is kept with the probability of p. This mechanism to guide the search using an existing solution can be viewed as a form of solution-guided search [24]. Let the number of states kept by the above criteria be K. If K is smaller than a parameter b, DD-LNS also keeps the best $b - K$ states from the remaining states according to a priority function. In practice, Gillard and Schaus [170] selects the best $b - K$ states minimizing the RLBs. We show pseudo-code for this procedure in Algorithm 13.

Since DD nodes are pruned and selected based on RLBs, the procedure of constructing a restricted DD can be considered beam search with randomization using a dual bound function for search

Algorithm 15 DD-LNS for minimization with a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$ with $\langle A, \times, \mathbf{1} \rangle$. An approximate dominance relation $\preceq_a$, a dual bound function η, the best solution $\overline{\sigma}$, a beam width b, and a probability p are given as input.

1: $d \leftarrow |\overline{\sigma}| - 2$, solved $\leftarrow \bot$ ▷ Initialize.
2: **while** the time limit is not reached and solved $= \bot$ **do**
3: $\hat{S} \leftarrow S^0[\![\overline{\sigma}_{:d}]\!]$ ▷ Decide the root node of the DD.
4: σ, exact $\leftarrow$ CompileRestrictedDD($\hat{S}$, $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$, $\preceq_a$, η, $\overline{\sigma}_{d+1:}$ b, p) ▷ Algorithm 14.
5: **if** $d = 0$ and exact **then**
6: solved $\leftarrow \top$ ▷ The optimality or infeasibility is proved.
7: **if** $\sigma \neq$ NULL **then**
8: $\overline{\sigma} \leftarrow \langle \overline{\sigma}_{:d}, \sigma \rangle$, $d \leftarrow |\overline{\sigma}| - 2$ ▷ Update the best solution.
9: **else**
10: **if** $d = 0$ **then**
11: $d \leftarrow |\overline{\sigma}| - 2$ ▷ Reset the neighborhood size.
12: **else**
13: $d \leftarrow d - 1$ ▷ Increase the neighborhood size.
14: **return** $\overline{\sigma}$, solved ▷ Return the solution.

guidance and pruning. Based on this interpretation, we present pseudo-code for the procedure to construct a restricted DD from a DyPDL model in Algorithm 14. In this interpretation, the open list O can be viewed as the set of DD nodes in the current layer. We have the following differences from the original implementation:

1. We generalize the cost expression of transition τ by defining $\mathsf{cost}_\tau(x, S) = w_\tau(S) \times x$, where w_τ is a numeric expression and $\times$ is a binary operator in a monoid. The original DD-LNS considers only $w_\tau(S) + x$.

2. We use an approximate dominance relation for pruning following Algorithms 9 and 10 in Sections 4.1.4 and 4.2.7 while the original implementation did not.

3. While the original DD-LNS assumes that all solutions have the same length, our algorithm does not.

We show pseudo-code for DD-LNS in Algorithm 15. DD-LNS decreases d by 1 if a better solution is not found with d, starting from $d = |\overline{\sigma}| - 2$ and restarting from $d = |\overline{\sigma}| - 2$ if $d = 0$, where $\overline{\sigma}$ is the current best solution. Therefore, in terms of LNS, DD-LNS removes a sequence of transitions $\overline{\sigma}_{d+1:}$ from the best solution in line 3 as a destroy heuristic, searches for a better solution by compiling a restricted DD in line 4 as a repair heuristic, and updates the current solution only if a new solution is better in line 8. DD-LNS terminates when the time limit is reached. In addition, DD-LNS terminates when it proves optimality or infeasibility (line 6); when $d = 0$ and exact $= \top$, the exact DD is constructed, and the state transition graph is exhausted.

Correctness

Algorithm 14 differs from Algorithm 10 in only the following two points: it searches for an $\hat{S}$-solution instead of S^0-solution, and it selects the best states to keep with a different criterion using Algorithm 13. Therefore, we can straightforwardly obtain the theoretical properties of Algorithm 14, which correspond to Theorems 16–18 for Algorithm 10 in Section 4.2.7.

Theorem 21. *Given a finite, acyclic, and monoidal DyPDL model (Definitions 14 and 15 in Section 3.1.1 and Definition 24 in Section 4.1.3), Algorithm 14 terminates in finite time.*

Theorem 22. *After line 11 of Algorithm 14, if $\overline{\sigma} \neq$ NULL, then $\overline{\sigma}$ is an $\hat{S}$-solution with $\overline{\gamma} = \mathsf{cost}_{\overline{\sigma}}(\hat{S})$.*

Theorem 23. *Let $\langle A, \times, \mathbf{1} \rangle$ be a monoid where $A \subseteq \mathbb{Q} \cup \{-\infty, \infty\}$ and A is isotone (Definitions 21 and 22 in Section 4.1.2). Given a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$ with $\langle A, \times, \mathbf{1} \rangle$ and $\overline{\gamma} \in A$, if an optimal solution exists for the minimization problem with the model, and Algorithm 14 returns $\overline{\sigma} \neq$ NULL and complete $= \top$, then $\overline{\sigma}$ is an optimal $\hat{S}$-solution. If Algorithm 14 returns $\overline{\sigma} =$ NULL and complete $= \top$, then there does not exist an $\hat{S}$-solution whose cost is less than $\overline{\gamma}$.*

For DD-LNS, we confirm that $\overline{\sigma}$ is a solution, and the optimality is proved if solved $= \top$.

Theorem 24. *In Algorithm 15, $\overline{\sigma}$ is a solution for the model.*

Proof. At the beginning, $\overline{\sigma}$ is a solution given as input, so the theorem holds. By line 3, $\hat{S}$ is reachable from the target state S^0 with $\overline{\sigma}_{:d}$ (Definition 8 in Section 3.1). In line 8, $\overline{\sigma}$ is updated to $\langle \overline{\sigma}_{:d}, \sigma \rangle$ if $\sigma \neq$ NULL. By Theorem 22, if $\sigma \neq$ NULL after line 4, σ is an $\hat{S}$-solution. By Lemma 1 in Section 3.1, a base state is reachable with $\langle \overline{\sigma}_{:d}, \sigma \rangle$, so it is a solution. □

Theorem 25. *Let $\langle A, \times, \mathbf{1} \rangle$ be a monoid where $A \subseteq \mathbb{Q} \cup \{-\infty, \infty\}$ and A is isotone. Given a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$ with $\langle A, \times, \mathbf{1} \rangle$, if an optimal solution exists for the minimization problem with the model, and DD-LNS returns $\overline{\sigma} \neq$ NULL and solved $= \top$, then $\overline{\sigma}$ is an optimal solution.*

Proof. If $d = 0$, $\hat{S} = S^0$ in line 3, σ returned by Algorithm 15 is a solution for the model. Therefore, the theorem is straightforward from Theorem 23. □

5.2 Large Neighborhood Beam Search

To develop new DIDP solvers using LNS and state space search, we start with a simple idea: given a solution path, $\langle (S^0, S^1, \sigma_1), ..., (S^{n-1}, S^n, \sigma_n) \rangle$, we remove a partial path $\langle (S^{i-1}, S^i, \sigma_i), ..., (S^{i+d-2}, S^{i+d-1}, \sigma_{i+d-1}) \rangle$ with length d and search for a better partial path from S^{i-1} to S^{i+d-1}. If we find a better solution path $\langle (S^0, S^1, \sigma_1), ..., (S^{i-1}, \hat{S}^i, \hat{\sigma}_i), ..., (\hat{S}^{i+\hat{d}-2}, S^{i+d-1}, \hat{\sigma}_{i+\hat{d}-1}), ..., (S^{n-1}, S^n, \sigma_n) \rangle$, whose length $\hat{d}$ can be different from d, we repeat this procedure with the new solution. While the overview of the algorithm is simple, there are design choices on how to select a partial path to remove and how to search for a better partial path. The novelty of our method compared to existing methods arises from such choices in addition to the fact that it is used for DIDP. First, we describe the modifications of beam search for DyPDL to search for a partial path from S^{i-1} to S^{i+d-1}. Then, we propose strategies to select a partial path to remove.

5.2.1 Beam Search in a Partial State Transition Graph

We want to find a path from S^{i-1} to S^{i+d-1} instead of from S^0 to a base state. We could modify line 8 of Algorithm 10 in Section 4.1.4 so that it checks if $S = S^{i+d-1}$ instead of $\exists B \in \mathcal{B}, S \models C_B$ (base case conditions). However, in DyPDL, it may not be desirable as shown in the following example.

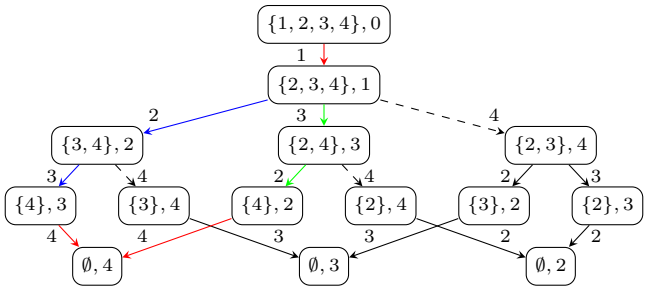


Figure 5.1: Partial state transition graph induced by the prefix $\langle 1 \rangle$ and the suffix $\langle 4 \rangle$ (highlighted in red) in Example 1. The current partial path is highlighted in blue and an alternative partial path is highlighted in green. Dashed transitions conflict with the suffix (explained in Section 5.2.2).

Algorithm 16 Rolling out a sequence of transitions σ from a state S for minimization with a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$ with $\langle A, \times, \mathbf{1} \rangle$.

1: **if** $\exists B \in \mathcal{B}, S \models C_B$ **then return** $\langle\rangle$ ▷ Return an empty sequence.
2: **for** $i \leftarrow 1, ..., |\sigma|$ **do**
3: **if** $\sigma_i \notin \mathcal{T}(S[\![\sigma_{:i-1}]\!])$ or $S[\![\sigma_{:i}]\!] \not\models \mathcal{C}$ **then return** NULL ▷ Check feasibility.
4: **if** $\exists B \in \mathcal{B}, S[\![\sigma_{:i}]\!] \models C_B$ **then return** $\sigma_{:i}$ ▷ Return the S-solution.
5: **return** NULL ▷ No base case is satisfied.

Example 1. Consider the following DP formulation, where $U \subseteq \{0, 1, 2, 3, 4\}$ is a set variable, $i \in \{0, 1, 2, 3, 4\}$ is an element variable, and c_{ij} for $i, j \in \{0, 1, 2, 3, 4\}$ is a constant.

$$\text{compute } V(\{1, 2, 3, 4\}, 0)$$
$$V(U, i) = \begin{cases} \min_{j \in U} c_{ij} + V(U \setminus \{j\}, j) & \text{if } U \neq \emptyset \\ 0 & \text{if } U = \emptyset. \end{cases}$$

Given a state S, each transition in the DyPDL model has precondition $j \in S[U]$ and effect $S[U] \setminus \{j\}$ on U for some j, and so we denote each transition by j. Each solution corresponds to a permutation of the transitions 1, 2, 3, and 4. A solution $\langle 1, 2, 3, 4 \rangle$ corresponds to a sequence of states $\langle (\{1, 2, 3, 4\}, 0), (\{2, 3, 4\}, 1), (\{3, 4\}, 2), (\{4\}, 3), (\emptyset, 4) \rangle$. Consider removing $\langle 2, 3 \rangle$ from the solution. We visualize the partial state transition graph in Figure 5.1. If an algorithm tries to find a path from $(\{2, 3, 4\}, 1)$ to $(\{4\}, 3)$, the original one, $\langle 2, 3 \rangle$, is the only path. However, a partial path $\langle 3, 2 \rangle$ from $(\{2, 3, 4\}, 1)$ to $(\{4\}, 2)$ also results in a valid solution $\langle 1, 3, 2, 4 \rangle$.

Considering the above example, instead of focusing on a partial path to a state, we focus on a partial path to a *suffix* of the solution path. Given a solution $\sigma = \langle \sigma_1, ..., \sigma_n \rangle$, if we remove a partial path $\sigma_{i:i+d-1} = \langle \sigma_i, ..., \sigma_{i+d-1} \rangle$, then $\sigma_{:i-1} = \langle \sigma_1, ..., \sigma_{i-1} \rangle$ is the prefix, and $\sigma_{i+d:} = \langle \sigma_{i+d}, ..., \sigma_n \rangle$ is the suffix. For a sequence of transitions $\hat{\sigma}$, we want to check if $\langle \sigma_{:i-1}; \hat{\sigma}; \sigma_{i+d:} \rangle$ is a valid solution. Therefore, for a state S found by a search algorithm, we perform a rollout of the suffix from S and check if each of the resulting states satisfies the state constraints and a base case (Algorithm 16). Since line 3 checks if each transition in the suffix is applicable and if the resulting state satisfies the state constraints, the returned value by Algorithm 16 is an S-solution if it is not NULL.

Algorithm 17 is a modified version of beam search that checks a solution using Algorithm 16. This algorithm takes a prefix prefix and a suffix suffix as input. In line 1, we denote the state resulting

Algorithm 17 Beam search for minimization in a partial state transition graph induced by prefix, suffix, and a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$ with $\langle A, \times, \mathbf{1} \rangle$. An approximate dominance relation $\preceq_a$, a dual bound function η, a primal bound $\overline{\gamma}$, and a beam width b are given as input.

1: $\hat{S} \leftarrow S^0[\![\mathsf{prefix}]\!]$ ▷ Apply the prefix.
2: **if** $\hat{S} \not\models \mathcal{C}$ **then return** NULL, $\top$
3: $\overline{\sigma} \leftarrow$ NULL, complete $\leftarrow$ NULL ▷ Initialize the solution.
4: $l \leftarrow |\mathsf{prefix}|$, $\sigma^l(\hat{S}) \leftarrow \mathsf{prefix}$, $g^l(\hat{S}) \leftarrow w_{\mathsf{prefix}}(S^0)$ ▷ Initialize the g-value.
5: $O \leftarrow \{\hat{S}\}$ ▷ Initialize the open list.
6: **while** $O \neq \emptyset$ and $\overline{\sigma} =$ NULL **do**
7: $G \leftarrow \emptyset$ ▷ Initialize the set of states.
8: **for all** $S \in O$ **do**
9: $\sigma^{\mathrm{suffix}} \leftarrow$ RolloutOut(suffix, S, $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$) ▷ Execute Algorithm 16.
10: **if** $\sigma^{\mathrm{suffix}} \neq$ NULL **then**
11: current_cost $\leftarrow g^l(S) \times \mathsf{cost}_{\sigma^{\mathrm{suffix}}}(S)$ ▷ Compute the solution cost.
12: **if** current_cost $< \overline{\gamma}$ **then**
13: $\overline{\gamma} \leftarrow$ current_cost, $\overline{\sigma} \leftarrow \langle \sigma^l(S); \sigma^{\mathrm{suffix}} \rangle$ ▷ Update the best solution.
14: **else**
15: **for all** $\tau \in \mathcal{T}(S) : S[\![\tau]\!] \models \mathcal{C}$ **do**
16: $g_{\mathrm{current}} \leftarrow g^l(S) \times w_\tau(S)$ ▷ Compute the g-value.
17: **if** $\nexists S' \in G$ such that $S[\![\tau]\!] \preceq_a S'$ and $g_{\mathrm{current}} \geq g^{l+1}(S')$ **then**
18: **if** $g_{\mathrm{current}} \times \eta(S[\![\tau]\!]) < \overline{\gamma}$ **then**
19: **if** $\exists S' \in S$ such that $S' \preceq_a S[\![\tau]\!]$ and $g_{\mathrm{current}} \leq g^{l+1}(S')$ **then**
20: $G \leftarrow G \setminus \{S'\}$ ▷ Remove a dominated state.
21: $\sigma^{l+1}(S[\![\tau]\!]) \leftarrow \langle \sigma^l(S); \tau \rangle$, $g^{l+1}(S[\![\tau]\!]) \leftarrow g_{\mathrm{current}}$ ▷ Update the g-value.
22: $G \leftarrow G \cup \{S[\![\tau]\!]\}$ ▷ Insert the successor state.
23: $l \leftarrow l + 1$ ▷ Proceed to the next layer.
24: $O \leftarrow \{S \in G \mid g^l(S) \times \eta(S) < \overline{\gamma}\}$ ▷ Prune states by the bound.
25: **if** $|O| > b$ **then**
26: $O \leftarrow$ the best b states in O, complete $\leftarrow \bot$ ▷ Keep the best b states.
27: **if** complete and $O \neq \emptyset$ **then**
28: complete $\leftarrow \bot$ ▷ A better solution may exist.
29: **return** $\overline{\sigma}$, complete ▷ Return the solution.

from applying the prefix to the target state by $\hat{S} = S^0[\![\mathsf{prefix}]\!]$ and initialize the open list O with $\hat{S}$ in line 5. In lines 9–13, the algorithm performs a rollout of the suffix from S and checks if it results in a better solution. Other parts are the same as Algorithm 10. Therefore, if $S^0[\![\mathsf{prefix}]\!]$ is reachable from S^0 with prefix, the first returned value by Algorithm 17 is a valid solution for the model if it is not NULL. Similar to Algorithm 10, Algorithm 17 can be easily adapted for maximization by replacing $<$ with $>$ and swapping $\geq$ with $\leq$.

We use this modified version of beam search in large neighborhood beam search (LNBS) as shown in Algorithm 18. In lines 3–5, LNBS selects parameters d, i, and b, which corresponds to a destroy heuristic. In line 6, LNBS performs beam search in the neighborhood as a repair heuristic. If an improving solution is found, LNBS updates the current solution $\overline{\sigma}$ in line 9. If the searched neighborhood is the original search space, i.e., $i = 1$ and $d = n$, and beam search proves the optimality or infeasibility, LNBS terminates in line 11. Therefore, if it is guaranteed to select $i = 1$ and $d = n$ with sufficiently large b given enough time, LNBS is guaranteed to find the optimal solution or prove the infeasibility, i.e., it is complete. In fact, CABS can be considered a configuration of LNBS, where $i = 1$, $d = n$, and b increases exponentially. DD-LNS can also be

Algorithm 18 LNBS for minimization with a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$ with $\langle A, \times, \mathbf{1} \rangle$. An approximate dominance relation $\preceq_a$, a dual bound function η, and a solution $\overline{\sigma}$ are given as input.

1: **while** the time limit is not reached **do**
2: $n \leftarrow |\overline{\sigma}|, \overline{\gamma} \leftarrow \mathsf{cost}_{\overline{\sigma}}(S^0)$
3: Select d such that $2 \leq d \leq n$
4: Select i such that $1 \leq i \leq n - d + 1$
5: Select beam width b
6: $\sigma, \mathrm{complete} \leftarrow$ BeamSearchPartial($\overline{\sigma}_{:i-1}$, $\overline{\sigma}_{i+d:}$, $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$, $\preceq_a$, η, $\overline{\gamma}$, b)
7: ▷ Execute Algorithm 17.
8: **if** $\sigma \neq$ NULL **then**
9: $\overline{\sigma} \leftarrow \sigma$ ▷ Update the best solution.
10: **if** $i = 1 \wedge d = n \wedge \mathrm{complete}$ **then**
11: **return** $\overline{\sigma}$, $\top$ ▷ The optimality is proved.
12: **return** $\overline{\sigma}$, $\bot$. ▷ Return the solution.

considered a configuration of LNBS, where i ranges from $n-2$ to 1, d is $n-i+1$, and b is fixed while beam search is extended with the randomization mechanism. We will describe the strategies that we use to select d, i, and b in Sections 5.2.3–5.2.5 after introducing a method to filter unnecessary transitions in Section 5.2.2.

Correctness

Algorithm 17 is a generalization of Algorithm 10. We present theorems for Algorithm 17, which correspond to Theorems 16 and 17 for Algorithm 10.

Theorem 26. *Given a finite, acyclic, and monoidal DyPDL model, if* prefix *and* suffix *have a finite number of transitions, Algorithm 17 terminates in finite time.*

Proof. Since prefix has a finite number of transitions, computing $\hat{S}$ in line 1 is done in finite time. Unlike Algorithm 10, Algorithm 17 uses Algorithm 16 to detect a solution. Since suffix has a finite number of transitions, Algorithm 16 terminates in finite time. Other parts are the same as Algorithm 10, so the theorem is proved by the proof of Theorem 16. □

Theorem 27. *Given* prefix *such that* $S^0[\![\mathsf{prefix}]\!]$ *is reachable from* S^0*, after line 13 of Algorithm 17, if* $\overline{\sigma} \neq$ *NULL, then* $\overline{\sigma}$ *is a solution with* $\overline{\gamma} = \mathsf{cost}_{\overline{\sigma}}(S^0)$.

Proof. By a similar argument to the proof of Theorem 17, in line 13, S is reachable from $\hat{S}$ with $\sigma^l(S)_{|\mathsf{prefix}|+1:}$. By line 1, $\hat{S}$ is reachable from S^0 with prefix. By Lemma 1, S is reachable from S^0 with $\sigma^l(S) = \langle \mathsf{prefix}; \sigma^l(S)_{|\mathsf{prefix}|+1:} \rangle$. Since Algorithm 16 checks if σ is an S-solution by following the definition (Definition 9 in Section 3.1), if $\sigma^{\mathsf{suffix}} \neq$ NULL is returned, it is an S-solution. By Lemma 1, a base state is reachable from S^0 with $\overline{\sigma} = \langle \sigma^l(S); \sigma^{\mathsf{suffix}} \rangle$, so $\overline{\sigma}$ is a solution for the model. □

For LNBS, we confirm that $\overline{\sigma}$ is a solution, and the optimality is proved if the second returned value is $\top$ in Algorithm 18.

Theorem 28. *In Algorithm 18,* $\overline{\sigma}$ *is a solution for the DyPDL model.*

Proof. At the beginning, $\overline{\sigma}$ is a solution given as input, so the theorem holds. In line 9, $\overline{\sigma}$ is updated to σ if $\sigma \neq$ NULL. In line 6, Algorithm 17 is given $\overline{\sigma}_{:i-1}$ as prefix. If $\overline{\sigma}$ is a solution, $S^0[\![\mathsf{prefix}]\!]$ is reachable from S^0. Therefore, by Theorem 27, if $\sigma \neq$ NULL after line 6, $\overline{\sigma} = \sigma$ is a solution. □

Theorem 29. *Let $\langle A, \times, \mathbf{1}\rangle$ be a monoid where $A \subseteq \mathbb{Q} \cup \{-\infty, \infty\}$ and A is isotone. Given a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C}\rangle$ with $\langle A, \times, \mathbf{1}\rangle$, if an optimal solution exists for the minimization problem with the model, and LNBS returns $\overline{\sigma} \neq$ NULL with $\top$, then it is an optimal solution.*

Proof. When $i = 1$ and $d = n$ in Algorithm 18, $\overline{\sigma}_{:i-1} = \overline{\sigma}_{i+d:} = \langle\rangle$. Then, Algorithm 17 is exactly the same as Algorithm 10, so the theorem follows from Theorem 18. □

5.2.2 Removing Conflicting Transitions

In Example 1, consider finding a partial path from a prefix $\langle 1\rangle$, which results in state $\hat{S} = (\{2, 3, 4\}, 1)$, to a suffix $\langle 4\rangle$ using beam search. In $\hat{S}$, three transitions 2, 3, and 4 are applicable. However, applying transition 4 does not lead to a feasible solution because it is already used in the suffix and cannot be applied twice: it requires $4 \in U$ and removes 4 from U, but no other transition adds 4 to U, so applying 4 makes the suffix inapplicable. Generalizing this example, if we know that a transition τ makes a transition τ' in the suffix inapplicable, then we can ignore τ when searching for a partial path. In particular, we focus on the effects of τ that add/remove an element to/from a set variable and the preconditions of τ' that require the element to be/not to be in that set variable.

Proposition 1. *Suppose that a DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C}\rangle$ has a set variable $U \in \mathcal{V}$ whose domain is 2^N, where N is a set of objects. There does not exist a solution $\langle \sigma_1, ..., \sigma_n\rangle$ such that any pair of $\tau = \sigma_i$ and $\tau' = \sigma_j$ for $1 \leq i < j \leq n$ satisfy either of the following conditions:*

1. *There exists $k \in N$ such that $\mathsf{eff}_\tau[U](S) = S[U] \setminus \{k\}$, $(k \in S[U]) \in \mathsf{pre}_{\tau'}$, and each $\tau'' \in \mathcal{T} \setminus \{\tau, \tau'\}$ does not change U or $\mathsf{eff}_{\tau''}[U](S) = S[U] \setminus \{l\}$ for some $l \in N$.*
2. *There exists $k \in N$ such that $\mathsf{eff}_\tau[U](S) = S[U] \cup \{k\}$, $(k \notin S[U]) \in \mathsf{pre}_{\tau'}$, and each $\tau'' \in \mathcal{T} \setminus \{\tau, \tau'\}$ does not change U or $\mathsf{eff}_{\tau''}[U](S) = S[U] \cup \{l\}$ for some $l \in N$.*

Proof. For the first case, τ removes k from U, and no other transition adds k to U. Since τ' requires k to be in U, once we apply τ, we cannot apply τ' later. The other case is proved similarly. □

Before starting beam search in a neighborhood, we remove a transition τ from the model if there exists a transition τ' in the suffix such that τ and τ' satisfy one of the conditions in Proposition 1. Detecting such a pair of transitions is done once before any search by checking the expression trees representing the preconditions and effects of transitions.

5.2.3 Bandit-Based Depth Selection

Selecting the depth of a neighborhood, d, in line 3 of Algorithm 18 is non-trivial. If d is too small, it is unlikely that an improving solution exists. However, if d is too large, each neighborhood search takes a long time. We want to select d such that the total cost improvement is maximized within the time limit.

We formulate the depth selection as a budgeted multi-armed bandit problem [38]. In a multi-armed bandit problem, in each *round*, we select one *arm* from a set of arms and obtain a reward associated with the arm, which is a random variable following an unknown probability distribution. The objective is to maximize the cumulative reward. In LNBS, since we want to maximize the cost improvement within the time limit, we consider a variant called the budgeted multi-armed bandit problem with continuous random costs [442]. In this problem, selecting an arm also incurs a cost, which is a random variable in $[0, 1]$. A reward is also assumed to be a random variable in $[0, 1]$. The objective is to maximize the cumulative reward within a given cost budget. We give a more formal definition with regard to our setting below.

We have the set of depths $D \subseteq \{2, ..., n\}$, corresponding to the set of arms. If we select a depth $d \in D$ and perform search in line 6 of Algorithm 18, we obtain a new solution σ by spending search time t. Our budget is the time limit T, and t is the cost. As the cost is assumed to take a value in $[0, 1]$, we divide the actual time by the time limit T. The reward is also assumed to be in $[0, 1]$, so we use relative cost improvement defined similarly to the primal gap. If the cost $\mathsf{cost}_\sigma(S^0)$ is better than the current best solution cost $\overline{\gamma}$, the reward is

$$r = \max\left\{1, \frac{|\overline{\gamma} - \mathsf{cost}_\sigma(S^0)|}{\max\{|\overline{\gamma}|, |\mathsf{cost}_\sigma(S^0)|\}}\right\}. \tag{5.1}$$

If no better solution is found, the reward is $r = 0$. We repeat rounds until reaching the time limit T. We do not know the reward r and time t before finishing a round, so we use random variables r_{dk} and t_{dk} representing the reward and time if depth d is used at round k. Let a be a strategy that selects a depth a_k at the round k. The number of rounds performed by a by the time limit, K_{aT}, is also a random variable. The objective is to find a strategy a that maximizes the total expected reward $\mathbb{E}\left[\sum_{k=1}^{K_{aT}} r_{a_k k}\right]$.

We use Budgeted-UCB [442] to address the problem formulated above. Intuitively, it tries to maximize the reward per cost (time in our case) while exploring arms (depths) that have not been selected frequently. At each round, if some depths in D have not been selected before, Budgeted-UCB selects one of them. Otherwise, let m_{dk} be the number of rounds where the depth d is selected up to round $k - 1$, and let $\bar{r}_{dk}$ and $\bar{t}_{dk}$ be the average reward and search time for d up to round $k - 1$. Budgeted-UCB selects the depth d that maximizes

$$\frac{\bar{r}_{dk}}{\bar{t}_{dk}} + \frac{\epsilon_{dk}}{\bar{t}_{dk}} + \frac{\epsilon_{dk}}{\bar{t}_{dk}} \frac{\min\{\bar{r}_{dk} + \epsilon_{dk}, 1\}}{\max\{\bar{t}_{dk} - \epsilon_{dk}, \lambda\}} \tag{5.2}$$

where $\epsilon_{dk} = \sqrt{\frac{2 \log (k-1)}{m_{dk}}}$ and λ is a positive lower bound of the search time of each round.

In practice, we initialize $D = \{2, 4, 8, ..., 2^a, n\}$, where n is the length of the initial feasible solution and a is the maximum integer such that $2^a < n$. If we get a solution whose length n' is different from n at round k, we replace n with n' in D using $m_{n',k+1} = m_{n,k+1}$, $\bar{r}_{n',k+1} = \bar{r}_{n,k+1}$, and $\bar{t}_{n',k+1} = \bar{t}_{n,k+1}$ and ignore depths greater than n' in D. If multiple depths have not been selected before or have the same value, we select the minimum depth among them. For λ, we use the time of the first round divided by 10 while there is no guarantee that it is a lower bound. Thus, the theoretical analysis of Budgeted-UCB studied in the original paper [442] does not necessarily apply to our setting. In addition, while Xia et al. [442] assumed that the pairs $\{(r_{dk}, t_{dk})\}_{k=1}^{\infty}$ are i.i.d., we do not have such a guarantee; for example, the relative cost improvement may become smaller

as we find better solutions.

5.2.4 Start Selection

Once LNBS determines the depth d to use, it selects a starting point i, which induces the prefix and the suffix, in line 4 of Algorithm 18. One possible strategy is to select i uniformly at random from 1 to $n-d+1$. However, if the partial path starting from i cannot be improved, i is not worth selecting. Let δ_{di} be the cost change by partial path $\sigma_{i:i+d-1}$ defined as

$$\delta_{di} = w_{\sigma_{:i+d-1}}(S^0) - w_{\sigma_{:i-1}}(S^0). \tag{5.3}$$

In minimization with a cost-algebraic DyPDL model, the path weight is non-decreasing and $\delta_{di} \geq 0$, so $\delta_{di} = 0$ means that the partial path $\sigma_{i:i+d-1}$ cannot be improved further.[2] Therefore, for such a DyPDL model, we ignore i with $\delta_{di} = 0$.

We consider two strategies to select i from the remaining options: sampling uniformly at random and using a probability distribution biased by δ_{di}. The second approach is motivated by an intuition that a larger δ_{di} indicates larger room for improvement in minimization. It selects i with a probability proportional to δ_{di}. As we explain in the next subsection, for each d and i, the beam width b_{di} is maintained. Since smaller b_{di} leads to a shorter search time, we discount the probability of selecting i by b_{di}. Concretely, given the depth d, we select the starting point i with the probability

$$p_{di} = \frac{\delta_{di}/b_{di}}{\sum_{j=1}^{n-d+1} \delta_{dj}/b_{dj}}. \tag{5.4}$$

If a DyPDL model is not cost-algebraic in minimization, a partial path cost can be negative. Therefore, in such a case, we consider the difference from the minimum partial path cost, i.e., $\delta_{di} - \min_{j=1}^{n-d+1} \delta_{dj}$, which is always nonnegative. For $i = \arg\min_{j=1}^{n-d+1} \delta_{dj}$, to avoid the zero probability to be selected, we use the second smallest value instead of δ_{di}. Overall, for minimization with a non-cost-algebraic model, we use the probability

$$p_{di} = \frac{\Delta_{di}}{\sum_{j=1}^{n-d+1} \Delta_{dj}} \tag{5.5}$$

where

$$\Delta_{di} = \begin{cases} 1/b_{di} & \text{if } \forall j = 1, ..., n-d+1, \delta_{di} = \delta_{dj} \\ (\min\{\delta_{dj} | j = 1, ..., n-d+1, \delta_{dj} > \delta_{di}\} - \delta_{di})/b_{di} & \text{else if } i = \arg\min_{j=1}^{n-d+1} \delta_{dj} \\ (\delta_{di} - \min_{j=1}^{n-d+1} \delta_{dj})/b_{di} & \text{else.} \end{cases} \tag{5.6}$$

Similarly, for maximization, we use Equation (5.5) with

$$\Delta_{di} = \begin{cases} 1/b_{di} & \text{if } \forall j = 1, ..., n-d+1, \delta_{di} = \delta_{dj} \\ (\delta_{di} - \max\{\delta_{dj} | j = 1, ..., n-d+1, \delta_{dj} < \delta_{di}\})/b_{di} & \text{else if } i = \arg\max_{j=1}^{n-d+1} \delta_{dj} \\ (\max_{j=1}^{n-d+1} \delta_{dj} - \delta_{di})/b_{di} & \text{else.} \end{cases} \tag{5.7}$$

[2] Theoretically, a better solution may be found with such i if the suffix is not empty because a partial path may change the state from which the suffix is applied, which may change the cost of the suffix.

5.2.5 Beam Width Selection

Given the depth d and the starting point i, LNBS selects a beam width b in line 5 of Algorithm 18. Here, we use a similar strategy to CABS: for each d and i, we initialize the beam width b_{di} to be 1 and update it to $2b_{di}$ after each round with d and i. If we find an improved solution in line 6, we reset $b_{d'i'} = 1$ only for d' and i' such that $i' > i$ or $i' + d' < i + d$; if $i' \leq i$ and $i' + d' \geq i + d$, the prefix and the suffix for the neighborhood induced by i' and d' do not change, and we know that a better partial path was not found with beam widths smaller than $b_{d'i'}$.[3] If the neighborhood is exhausted, i.e., complete $= \top$ in line 6, we ignore the combination of d and i in lines 3 and 4 until a new solution is found and b_{di} is reset to 1 ($\delta_{di} = 0$ is used in Equation (5.4), and $\Delta_{di} = 0$ is used in Equation (5.5)). If all starting points of a partial path with length d are ignored, d is ignored in the depth selection mechanism. Since the number of neighborhoods is finite, LNBS eventually exhausts all the neighborhoods and finds the optimal solution, which guarantees completeness.

Theorem 30. *Given a finite, acyclic, and monoidal DyPDL model and a sufficiently long time limit, LNBS with the depth, start, and beam width selection strategies described in Sections 5.2.3–5.2.5 terminates in finite time reaching line 11.*

Proof. In the next paragraph, we prove that given the current solution $\overline{\sigma}$, LNBS finds an improving solution or reaches line 11 after performing beam search finite times. By Theorem 26, each run of beam search terminates in finite time. Since there are a finite number of solutions for the finite and acyclic DyPDL model, after finding finitely many improving solutions, LNBS reaches line 11 in finite time.

Since the model is finite, the current solution $\overline{\sigma}$ has a finite number of transitions. There are a finite number of neighborhoods defined by d and i. The beam width b_{di} doubles after each run of beam search. When an improving solution is found after beam search with d and i, our claim holds. Otherwise, beam widths are kept in all other neighborhoods. As in the proof of Theorem 19 in Section 4.2.7, complete $= \top$ will be returned in line 6 after selecting the same d and i finitely many times without finding an improving solution, i.e., the neighborhood defined by d and i will be exhausted. The depth and start selection strategies always select a neighborhood that is not exhausted. Therefore, after performing finitely many runs of beam search, LNBS finds a better solution or exhausts the neighborhood with $i = 1$ and $d = n$ and reaches line 11. □

5.3 Experimental Evaluation

We compare LNBS with MIP, CP, CABS, and DD-LNS using the eleven problem classes used in Chapter 4: the traveling salesperson problem with time windows (TSPTW) [377], the capacitated vehicle routing problem (CVRP) [95], the multi-commodity pickup and delivery traveling salesperson problem (m-PDTSP) [208], the orienteering problem with time windows (OPTW) [238], the multi-dimensional knapsack problem (MDKP) [306, 249], bin packing [306], the simple assembly line balancing problem (SALBP-1) [374, 22], single machine total weighted tardiness ($1||\sum w_i T_i$) [123], talent scheduling [74], the minimization of open stack problem (MOSP) [446], and graph-clear [256]. We use the same instance sets and the same DyPDL, MIP, and CP models as Chapter 4 except for

[3] Theoretically, a better solution may be found with beam width smaller than $b_{d'i'}$ if the updated primal bound changes the search behavior.

the CP model of graph-clear: we use the CPS formulation by Morin et al. [317] since it achieves better solution quality than the CPN formulation [317] used in Chapter 4. All MIP and CP models are presented in Appendix B. The problem instances are converted to YAML-DyPDL files by Python scripts. All DyPDL models are presented in Section 3.3, and their YAML-DyPDL domain files are in Appendix A.2.

We evaluate two configurations, LNBS/uniform and LNBS/bias, with different start selection mechanisms introduced in Section 5.2.4. For minimization with a cost-algebraic DyPDL model, which is the case with problem classes except for OPTW and MDKP, LNBS/uniform and LNBS/bias ignore the starting point i of a partial path if $\delta_{di} = 0$ in Equation (5.3). Then, LNBS/uniform selects i uniformly at random, and LNBS/bias selects i using p_{di} in Equation (5.4). For OPTW and MDKP, LNBS/uniform selects i uniformly at random from all candidates, and LNBS/bias selects i using Equations (5.5) and (5.7). LNBS/uniform and LNBS/bias use Budgeted-UCB to select the depth as described in Section 5.2.3 and geometrically increase the beam width for each neighborhood as described in Section 5.2.5.

As described in Section 5.2.2, LNBS/uniform and LNBS/bias remove transitions that conflict with the suffix based on Proposition 1. In the DyPDL models for ten out of the eleven benchmark problem classes, Proposition 1 is applicable. In TSPTW, CVRP, m-PDTSP, OPTW, bin packing, talent scheduling, and MOSP (Sections 3.2.1, 3.3.1–3.3.3, 3.3.5, 3.3.8 and 3.3.9), each transition removes one element from a set variable, and no transition adds an element to the set variable. In SALBP-1 (Section 3.3.6), except for one transition, which does not change a set variable, each transition removes one element from the set variable. In $1||\sum w_i T_i$ and graph-clear (Sections 3.3.7 and 3.3.10), each transition adds one element to a set variable, and no transition removes an element from the set variable. In MDKP (Section 3.3.4), the DyPDL model has no set variables, so Proposition 1 is not applicable. In TSPTW, m-PDTSP, bin packing, $1||\sum_i w_i T_i$, talent scheduling, MOSP, and graph-clear, if a suffix has m transitions, LNBS removes m transitions from the model. In CVRP, for each element in a set variable, there are two transitions to remove it, so LNBS removes $2m$ transitions. Similarly, in OPTW, LNBS removes $3m$ transitions since there are three transitions to remove each element. In SALBP-1, LNBS removes $m - k$ transitions from the model, where k is the number of transitions in the suffix that does not change a set variable.

CABS, DD-LNS, and the LNBS configurations are implemented in didp-rs 0.7.0.[4] All solvers use the dual bound function defined in YAML-DyPDL as the heuristic function, i.e., beam search or a restricted DD keeps the best states according to $g(S) \times \eta(S)$, where $g(S)$ is the path weight to reach S, $\times \in \{+, \max\}$ is a binary operator, and η is the dual bound function. In LNBS configurations and DD-LNS, CABS is run first to find a feasible solution, and then LNBS and DD-LNS are run to improve the solution. For DD-LNS, we use $b = 1000$ and $p = 0.1$ following the original paper [170].

LNBS focuses on improving the solution quality and anytime performance. We use two metrics explained in Chapter 4: the primal gap, which is the relative difference between the solution cost achieved by a solver and the best-known solution cost, and the primal integral, which is the integral of the primal gap over time. For TSPTW, CVRP, and $1||\sum w_i T_i$, best-known solution costs are provided with the problem instances. For other problem classes, we use the best cost found by the solvers used in the evaluation. In addition, to investigate the disadvantage of LNBS in proving

[4]For LNBS/bias in OPTW and MDKP, didp-rs 0.7.2 is used since maximization is not supported in didp-rs 0.7.0. didp-rs 0.7.2 does not have other differences from 0.7.0 except for bug fixes irrelevant to the DyPDL models used in the evaluation.

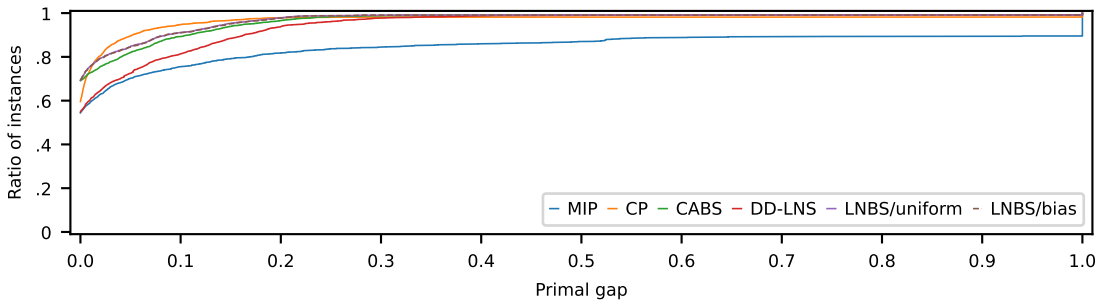


Figure 5.2: The ratio of instances against the primal gap averaged over all problem classes.

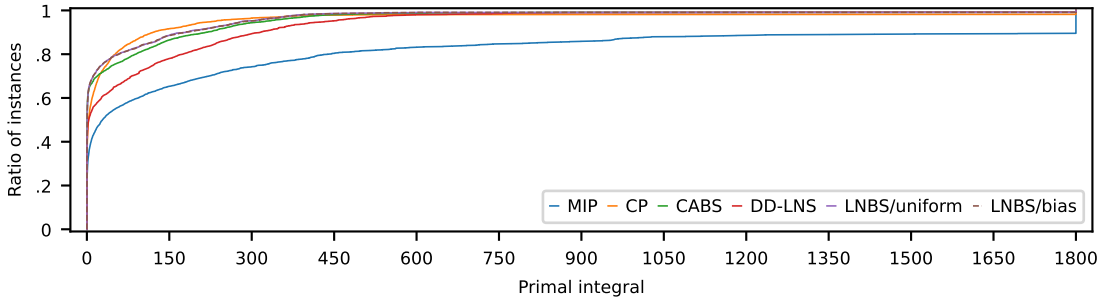


Figure 5.3: The ratio of instances against the primal integral averaged over all problem classes.

optimality, we evaluate coverage, the number of instances where optimality or infeasibility is proved within the time limit.

Our experimental setting is the same as Chapter 4: we use Python 3.10.2, didp-yaml 0.7.0 for DIDP, Gurobi 10.0.1 for MIP, and IBM ILOG CP Optimizer 22.1.0 for CP; for each run, we use an Intel Xeon Gold 6418 processor with an 8 GB memory limit, a 30-minute time limit, and a single thread. For LNBS and DD-LNS, for each problem instance, we take the median of 5 runs in terms of the primal gap using different random seeds.

5.3.1 Results

Figure 5.2 shows the cumulative ratio of instances against the primal gap, where the y-axis is averaged over all problem classes. The x-axis is the primal gap, and the y-axis is the ratio of instances where the primal gap is less than or equal to x. Similarly, Figure 5.3 shows the cumulative ratio of instances against the primal integral averaged over all problem classes. We show plots presenting the results for each problem class in Appendix D. LNBS/bias and LNBS/uniform have similar performance, so they overlap in the plot. The LNBS configurations outperform MIP, DD-LNS, and CABS, and are comparable with CP; they have a higher ratio of instances than CP when the primal gap or primal integral is less than a certain threshold.

Table 5.1 shows the average primal gap in each problem class, and Table 5.2 shows the average primal integral. The value of LNBS/uniform or LNBS/bias is in bold if it is better than the baselines (MIP, CP, CABS, and DD-LNS), and the best value is underlined.

The difference between LNBS/uniform and LNBS/bias is not large. LNBS/uniform has a smaller average primal gap in TSPTW, OPTW, MDKP, and $1||\sum w_i T_i$, and LNBS/bias has a smaller primal gap in CVRP. In all of these problem classes except for MDKP, the difference in the average

Table 5.1: Average primal gap of MIP, CP, CABS, DD-LNS, and LNBS in each problem class. The primal gap of LNBS/uniform or LNBS/bias is in bold if it is lower than the baselines (MIP, CP, CABS, and DD-LNS), and the lowest of the baselines is in bold if the LNBS configurations are worse. The lowest primal gap is underlined.

	MIP	CP	CABS	DD-LNS	LNBS/uniform	LNBS/bias
TSPTW (340)	0.2357	0.0259	0.0033	0.0095	**0.0017**	**0.0021**
CVRP (207)	0.5950	0.2534	0.1752	0.2501	**0.1626**	**0.1622**
m-PDTSP (1178)	0.0842	0.0125	**0.0019**	0.0100	0.0025	0.0024
OPTW (144)	0.2180	**0.0098**	0.0279	0.0608	0.0314	0.0318
MDKP (276)	**0.0000**	0.0058	0.1097	0.0889	0.1019	0.1036
Bin Packing (1615)	0.0417	**0.0015**	0.0017	0.0190	0.0023	0.0023
SALBP-1 (2100)	0.2697	0.0046	**0.0002**	0.0067	0.0021	0.0020
$1\|\|\sum w_iT_i$ (375)	0.0131	**0.0009**	0.0329	0.0384	0.0047	0.0051
Talent Scheduling (1000)	0.0395	0.0087	0.0167	0.0596	**0.0043**	**0.0042**
MOSP (570)	0.0356	0.0044	**0.0000**	0.0203	0.0002	0.0002
Graph-Clear (135)	0.1386	0.0151	**0.0000**	0.0009	**0.0000**	**0.0000**

Table 5.2: Average primal integral of MIP, CP, CABS, DD-LNS, and LNBS in each problem class. The primal integral of LNBS/uniform or LNBS/bias is in bold if it is lower than the baselines (MIP, CP, CABS, and DD-LNS), and the lowest of the baselines is in bold if the LNBS configurations are worse. The lowest primal integral is underlined.

	MIP	CP	CABS	DD-LNS	LNBS/uniform	LNBS/bias
TSPTW (340)	509.18	48.97	9.25	22.99	**6.38**	**6.90**
CVRP (207)	1147.11	482.89	333.68	467.61	**319.99**	**318.93**
m-PDTSP (1178)	177.82	26.04	5.25	20.45	5.47	**5.02**
OPTW (144)	448.17	**23.89**	65.67	142.16	73.76	74.00
MDKP (276)	**0.11**	15.87	201.73	171.09	189.52	192.69
Bin Packing (1615)	88.32	7.92	**4.90**	34.99	8.37	8.26
SALBP-1 (2100)	536.41	28.44	**1.93**	13.28	6.96	6.90
$1\|\|\sum w_iT_i$ (375)	64.01	**3.49**	71.21	78.56	12.57	13.23
Talent Scheduling (1000)	122.30	30.32	36.77	113.42	**11.01**	**11.01**
MOSP (570)	92.63	13.01	**0.31**	37.02	0.62	0.62
Graph-Clear (135)	337.66	44.34	**0.45**	2.51	0.51	0.53

primal gap is 0.0004. The difference is even smaller in m-PDTSP, SALBP-1, and talent scheduling (LNBS/bias is better by 0.00001), and there is no difference in bin packing, MOSP, and graph-clear. The average primal integral has a similar tendency.

Compared to CABS, LNBS/uniform and LNBS/bias achieve a better average primal gap and average primal integral in TSPTW, CVRP, MDKP, $1||\sum w_iT_i$, and talent scheduling. In $1||\sum w_iT_i$, while LNBS/uniform and LNBS/bias show a significant improvement from CABS (0.0047 and 0.0051 from 0.0329) outperforming MIP (0.0131), CP is still the best. However, in talent scheduling, LNBS/uniform and LNBS/bias outperform CP while CABS does not. Overall, the LNBS configurations are better than MIP and CP in seven problem classes, TSPTW, CVRP, m-PDTSP, SALBP-1, talent scheduling, MOSP, and graph-clear, while CABS is better in six problem classes excluding talent scheduling. CABS has a better average primal gap than LNBS/uniform and LNBS/bias in OPTW and SALBP-1. While CABS is also better than the LNBS configurations in m-PDTSP, bin packing, and MOSP, the differences are relatively small. Similarly, in terms of primal integral, CABS is better in bin packing and SALBP-1, and the differences in m-PDTSP, MOSP, and graph-clear are relatively small.

Table 5.3 shows the number of instances where the LNBS configurations have a better/same/worse primal gap, and Table 5.4 shows the number of instances where the LNBS configurations have a better/same/worse primal integral. In TSPTW, CVRP, MDKP, $1||\sum w_iT_i$, and talent scheduling, the LNBS configurations have a better average primal gap than CABS, and the number of instances

Table 5.3: Number of instances where LNBS has a better/same/worse primal gap than CABS. A value in 'Better'/'Worse' is in bold if it is larger than 'Worse'/'Better'.

	LNBS/uniform			LNBS/bias		
	Better	Same	Worse	Better	Same	Worse
TSPTW (340)	**22**	309	9	**21**	307	12
CVRP (207)	**132**	42	33	**132**	40	35
m-PDTSP (1178)	**69**	1091	18	**53**	1108	17
OPTW (144)	14	103	**27**	13	104	**27**
MDKP (276)	**257**	6	13	**240**	5	31
Bin Packing (1615)	9	1529	**77**	11	1525	**79**
SALBP-1 (2100)	3	1801	**296**	4	1802	**294**
$1\|\|\sum w_i T_i$ (375)	**78**	295	2	**78**	295	2
Talent Scheduling (1000)	**595**	385	20	**590**	389	21
MOSP (570)	0	568	**2**	0	568	**2**
Graph-Clear (135)	0	135	0	0	135	0

Table 5.4: Number of instances where LNBS has a better/same/worse primal integral than CABS. A value in 'Better'/'Worse' is in bold if it is larger than 'Worse'/'Better'.

	LNBS/uniform			LNBS/bias		
	Better	Same	Worse	Better	Same	Worse
TSPTW (340)	51	0	**289**	57	0	**283**
CVRP (207)	**130**	19	58	**136**	19	52
m-PDTSP (1178)	195	2	**981**	222	2	**954**
OPTW (144)	21	0	**123**	20	0	**124**
MDKP (276)	**250**	0	26	**226**	0	50
Bin Packing (1615)	50	0	**1565**	48	0	**1567**
SALBP-1 (2100)	191	0	**1909**	194	0	**1906**
$1\|\|\sum w_i T_i$ (375)	**228**	0	147	**223**	0	152
Talent Scheduling (1000)	**863**	0	137	**858**	0	142
MOSP (570)	122	0	**448**	110	0	**460**
Graph-Clear (135)	7	0	**128**	30	0	**105**

where they have a better primal gap is larger than the number of instances where they are worse. In terms of the primal integral, we observe the same tendency in CVRP, MDKP, $1||\sum w_i T_i$ and talent scheduling, but not in TSPTW. In m-PDTSP, although the LNBS configurations have a worse average primal gap than CABS, the number of instances where they have a better primal gap is larger than the number of instances where they are worse.

Table 5.5 shows the coverage. CABS is equal to or better than LNBS/uniform and LNBS/bias in all problem classes. LNBS proves optimality or infeasibility only when it selects the entire state transition graph as a neighborhood and exhausts it. In contrast, CABS always searches the entire state transition graph when b becomes large enough. Nevertheless, LNBS/uniform and LNBS/bias prove the optimality of more than 93% of instances that are optimally solved by CABS.

DD-LNS performs worse than CABS, LNBS/uniform, and LNBS/bias in all problem classes except for MDKP. Previous work has reported that DD-LNS is effective for TSPTW [170]. Note however that the DD-LNS results in Tables 5.1 and 5.2 are not the results reported by Gillard and Schaus. To validate that our implementation and experimental settings do not handicap DD-LNS, we compare the results of DD-LNS with those of the original paper for TSPTW.[5] Our DD-LNS implementation finds a better solution than the original in all instances with the time limit of 600 seconds used in the original paper. This difference is likely due to the difference between the DP models used by us and Gillard and Schaus. In TSPTW (Section 3.2.1), a solution is a tour that starts from a depot, visits each customer j within the time window $[a_j, b_j]$, and returns to the depot, and an optimal solution minimizes the total travel time. In both DP models, state variables are the

[5] `https://github.com/xgillard/ijcai_22_DDLNS/blob/main/results/tsptw/ddlns/results_w1000_t600.txt`

Table 5.5: Coverage of MIP, CP, CABS, DD-LNS, and LNBS in each problem class. The best coverage of the baselines (MIP, CP, CABS, and DD-LNS) is in bold if LNBS configurations are worse. The highest coverage is underlined.

	MIP	CP	CABS	DD-LNS	LNBS/uniform	LNBS/bias
TSPTW (340)	222	47	**259**	109	242	242
CVRP (207)	**27**	0	6	0	6	6
m-PDTSP (1178)	940	**1049**	1035	459	1035	1035
OPTW (144)	16	49	**64**	10	62	62
MDKP (276)	**165**	6	5	2	5	5
Bin Packing (1615)	1159	**1234**	1167	777	1140	1142
SALBP-1 (2100)	1423	1584	**1802**	1509	1703	1706
$1\|\|\sum w_i T_i$ (2100)	106	150	**288**	100	281	281
Talent Scheduling (1000)	0	0	**239**	0	236	237
MOSP (570)	231	437	**527**	353	524	524
Graph-Clear (135)	17	1	103	3	103	103

set of unvisited customers U, the current customer i, and the current time t, and each transition corresponds to visiting a customer. Each DP model has a dual bound (called RLB by Gillard and Schaus), a lower bound on the optimal solution cost. While Gillard and Schaus use a dual bound based on a minimum spanning tree, we use a simpler and looser one based on the minimum travel time between customers (see Section 3.2.1). However, we use information that was not considered by Gillard and Schaus. First, we use dominance between states based on the current time: a state S dominates another state S' if $S[U] = S'[U]$, $S[i] = S'[i]$, and $S[t] \leq S'[t]$. Furthermore, since the time to visit customer j is underestimated by $t + c^*_{ij}$, where c^*_{ij} is the shortest travel time from i to j, we define state constraints $\forall j \in S[U], t + c^*_{S[i],j} \leq b_j$. The dominance and the state constraints are useful to prune states, which potentially explains the performance gap. Another difference is whether considering the time window constraints at the depot or not. In the benchmark instances used above, a time window $[a_0, b_0]$ is defined for the depot. As we explained in Section 4.3.2, in the problem instances used in the evaluation, we can always return to the depot by the deadline b_0 after visiting the final customer, so we do not need to consider it. Gillard and Schaus explicitly model a required return to the depot within $[a_0, b_0]$ while we do not, but this difference does not change the set of feasible or optimal solutions.

5.3.2 Instance Set-Wise Comparisons in a Subset of Problems

As shown in Section 5.3.1, LNBS/uniform and LNBS/bias are significantly better than CABS in CVRP, MDKP, $1||\sum w_i T_i$, and talent scheduling and worse in OPTW and SALBP-1. However, the performance of CABS and the LNBS configurations are relatively close in other problem classes. To clearly illustrate the differences, we present instance set-wise comparison for each problem class where CABS and LNBS perform similarly, i.e., TSPTW, m-PDTSP, bin packing, MOSP, and graph-clear.

TSPTW

We show the coverage, average primal gap, and average primal integral in each instance set of TSPTW in Table 5.6. The LNBS/uniform and LNBS/bias show significant improvement over CABS in OhlmannThomas [328], where no instance is optimally solved by the DIDP solvers. In other instance sets, CABS and the LNBS configurations are similar. In the AFG set [10], CABS solves 45 out of 50 instances, LNBS/uniform and LNBS/bias solve 43, and CABS and LNBS/uniform

Table 5.6: Comparison of CABS and LNBS in each instance set of TSPTW. 'gap' is the average primal gap, 'integral' is the average primal integral, and 'c.' is the coverage. The value of LNBS/uniform or LNBS/bias is in bold if it is better than CABS, and the value of CABS is in bold if LNBS/uniform and LNBS/bias are worse. The best value is underlined.

	CABS			LNBS/uniform			LNBS/bias		
	gap	integral	c.	gap	integral	c.	gap	integral	c.
TSPTW Total (340)	0.0033	9.25	**259**	**0.0017**	**6.38**	242	**0.0021**	**6.90**	242
AFG (50)	0.0002	**1.28**	**45**	0.0002	1.53	43	0.0003	1.55	43
Dumas (135)	0.0000	**0.55**	135	0.0000	0.71	135	0.0000	0.70	135
GendreauDumasExtended (130)	0.0009	4.84	**79**	**0.0007**	**4.18**	64	0.0013	**4.78**	64
OhlmannThomas (25)	0.0392	95.07	0	**0.0188**	**58.20**	0	**0.0217**	**62.11**	0

Table 5.7: Comparison of CABS and LNBS in each instance set of m-PDTSP. 'gap' is the average primal gap, 'integral' is the average primal integral, and 'c.' is the coverage. The value of LNBS/uniform or LNBS/bias is in bold if it is better than CABS, and the value of CABS is in bold if LNBS/uniform and LNBS/bias are worse. The best value is underlined.

	CABS			LNBS/uniform			LNBS/bias		
	gap	integral	c.	gap	integral	c.	gap	integral	c.
m-PDTSP Total (1178)	**0.0019**	5.25	1035	0.0025	5.47	1035	0.0024	**5.02**	1035
class1 (248)	**0.0091**	24.35	145	0.0119	25.40	145	0.0116	**23.27**	144
class2 (720)	0.0000	0.16	720	0.0000	0.16	720	0.0000	0.16	720
class3 (210)	0.0000	0.13	170	0.0000	**0.11**	170	0.0000	**0.11**	**145**

have the same average primal gap. In the Dumas set [109], all instances are optimally solved by the DIDP solvers. In the GendreauDumasExtended set [160], CABS/uniform has a slightly better average primal gap and primal integral.

m-PDTSP

We show the evaluation results in each instance set of m-PDTSP in Table 5.7. All instance sets are proposed by Hernández-Pérez and Salazar-González [208]. CABS and the LNBS configurations achieve the same primal gap in all instances of class2 and class3. Although CABS has a better average primal gap than the LNBS configurations in class1, as shown in Table 5.3, the number of instances where the LNBS configurations have a better primal gap is larger than the number of instances where they are worse. Therefore, none of CABS and the LNBS configurations is a clear winner.

Bin Packing

We show the evaluation results in each instance set of bin packing in Table 5.8. CABS is equal or better than LNBS/uniform and LNBS/bias in the majority of the instance sets. In Falkenauer T and U [130], Scholl 2 [384], and Wäscher [441], CABS shows a clear advantage over the LNBS configurations. In Schwerin 2 [385] and Hard28 [382], CABS and the LNBS configurations achieve the same average primal gap, and CABS has a better primal integral. In Scholl 1, LNBS/uniform and LNBS/bias have a better average primal gap while CABS has a better average primal integral. In this instance set, CABS has a better primal gap than the LNBS configurations in 9 instances and a worse primal gap in 5 instances. In Schwerin 1 [385], while LNBS configurations are better than CABS on average, this result is due to only one instance: LNBS/uniform and LNBS/bias optimally

Table 5.8: Comparison of CABS and LNBS in each instance set of bin packing. 'gap' is the average primal gap, 'integral' is the average primal integral, and 'c.' is the coverage. The value of LNBS/uniform or LNBS/bias is in bold if it is better than CABS, and the value of CABS is in bold if LNBS/uniform and LNBS/bias are worse. The best value is underlined.

	CABS			LNBS/uniform			LNBS/bias		
	gap	integral	c.	gap	integral	c.	gap	integral	c.
Bin Packing Total (1615)	**0.0018**	**4.94**	**1167**	0.0023	8.41	1140	0.0023	8.30	1142
Falkenauer T (80)	**0.0073**	**19.62**	24	0.0085	47.25	24	0.0084	46.70	24
Falkenauer U (80)	**0.0005**	**5.26**	**42**	0.0025	9.89	37	0.0025	10.00	37
Hard28 (28)	0.0000	**0.12**	0	0.0000	0.20	0	0.0000	0.19	0
Scholl 1 (720)	0.0004	**1.23**	**540**	**0.0003**	1.75	530	**0.0003**	1.71	530
Scholl 2 (480)	**0.0033**	**8.34**	**392**	0.0046	13.00	380	0.0045	12.89	381
Scholl 3 (10)	0.0035	6.79	1	0.0035	8.05	1	**0.0018**	**5.58**	**2**
Schwerin 1 (100)	0.0021	**4.36**	96	**0.0016**	4.64	**97**	**0.0016**	4.51	**97**
Schwerin 2 (100)	0.0005	**2.18**	62	0.0005	3.86	62	0.0005	3.75	62
Wäscher (17)	**0.0070**	**22.32**	**10**	0.0115	33.72	9	0.0115	33.09	9

Table 5.9: Comparison of CABS and LNBS in each instance set of MOSP. 'gap' is the average primal gap, 'integral' is the average primal integral, and 'c.' is the coverage. The value of LNBS/uniform or LNBS/bias is in bold if it is better than CABS, and the value of CABS is in bold if LNBS/uniform and LNBS/bias are worse. The best value is underlined.

	CABS			LNBS/uniform			LNBS/bias		
	gap	integral	c.	gap	integral	c.	gap	integral	c.
MOSP Total (570)	**0.0000**	**0.31**	**527**	0.0002	0.62	524	0.0002	0.62	524
Challenge (46)	0.0000	0.02	45	0.0000	0.02	45	0.0000	0.02	45
Chu and Stuckey (200)	**0.0000**	**0.73**	**161**	0.0006	1.39	158	0.0006	1.36	158
Faggioli and Bentivoglio (300)	0.0000	0.01	300	0.0000	0.01	300	0.0000	0.01	300
SCOOP (24)	0.0000	**1.21**	21	0.0000	3.02	21	0.0000	3.05	21

solve one more instance than CABS with a better primal gap and primal integral. In other instances in Schwerin 1, LNBS configurations and CABS have the same primal gap, and CABS is better in the primal integral. Similarly, LNBS/bias optimally solves one instance of Scholl 3 [384] and achieves a better primal gap and primal integral than CABS. In other instances of Scholl 3, the LNBS configurations and CABS have the same primal gap, and CABS has a better primal integral. Overall, as shown in Table 5.3, while the LNBS configurations have a better primal gap and primal integral than CABS in a small number of instances, CABS is equal or better in a larger number of instances.

MOSP

We show the evaluation results in each instance set of MOSP in Table 5.9. The difference between CABS and the LNBS configurations is still not clear; they achieve the same primal gap in all instances of Challenge [402], Faggioli and Bentivoglio [129], and SCOOP.[6] In Chu and Stuckey[79], CABS has a better primal gap in two instances as shown in Table 5.3. The average primal integral is the same in Challenge and Faggioli and Bentivoglio, and the difference is small in Chu and Stucky and SCOOP.

[6] `https://cordis.europa.eu/project/id/32998`

Table 5.10: Comparison of CABS and LNBS in each instance set of graph-clear. 'gap' is the average primal gap, 'integral' is the average primal integral, and 'c.' is the coverage. The value of LNBS/uniform or LNBS/bias is in bold if it is better than CABS, and the value of CABS is in bold if LNBS/uniform and LNBS/bias are worse. The best value is underlined.

	CABS			LNBS/uniform			LNBS/bias		
	gap	integral	c.	gap	integral	c.	gap	integral	c.
Graph-Clear Total (135)	0.0000	**0.45**	103	0.0000	0.51	103	0.0000	0.53	103
Planar $n = 20$ (20)	0.0000	0.02	20	0.0000	0.02	20	0.0000	0.02	20
Planar $n = 30$ (20)	0.0000	**0.07**	20	0.0000	0.10	20	0.0000	0.10	20
Planar $n = 40$ (20)	0.0000	1.39	19	0.0000	**1.19**	19	0.0000	1.41	19
Random $n = 20$ (25)	0.0000	0.09	25	0.0000	0.13	25	0.0000	**0.05**	25
Random $n = 30$ (25)	0.0000	**0.07**	17	0.0000	0.11	17	0.0000	0.11	17
Random $n = 40$ (25)	0.0002	**1.08**	2	0.0002	1.49	2	0.0002	1.49	2

Table 5.11: Average primal gap, average primal integral, and coverage of MIP, CP, CABS, DD-LNS, and LNBS in large instances of m-PDTSP, MOSP, and Graph-Clear. The value of LNBS/uniform or LNBS/bias is in bold if it is better than the baselines (MIP, CP, CABS, and DD-LNS), and the best value of the baselines is in bold if LNBS configurations are worse. The best value is underlined.

Primal Gap	MIP	CP	CABS	DD-LNS	LNBS/uniform	LNBS/bias
m-PDTSP (240)	0.5774	0.1509	0.0744	0.1401	**0.0718**	**0.0701**
MOSP (760)	0.8810	0.0689	**0.0010**	0.0418	0.0027	0.0029
Graph-Clear (50)	0.5233	0.5290	**0.0012**	0.0769	0.0032	0.0040
Primal Integral						
m-PDTSP (1178)	1095.72	289.00	157.78	273.58	**153.63**	**151.97**
MOSP (570)	1601.08	152.92	**5.25**	75.65	10.15	10.51
Graph-Clear (135)	963.06	1268.39	**8.55**	138.74	16.22	17.07
Coverage						
m-PDTSP (240)	48	77	100	78	100	100
MOSP (760)	0	0	**154**	0	148	148
Graph-Clear (50)	0	0	0	0	0	0

Graph-Clear

We show the evaluation results in each instance set of graph-clear in Table 5.9. In graph-clear (Section 3.3.10), each instance is associated with a graph with n nodes. The instances are generated by us following Morin et al. [317] (see Section 4.3.2), using $n = 20, 30, 40$ for planar and random graphs. Similar to MOSP, the difference between CABS and the LNBS configurations is unclear: CABS, LNBS/uniform, and LNBS/bias have the same primal gap in all instances and a similar average primal integral in all instance sets.

5.3.3 Larger Instances for m-PDTSP, MOSP, and Graph-Clear

While we compare CABS, LNBS/uniform, and LNBS/bias in each instance set of TSPTW, m-PDTSP, bin packing, MOSP, and graph-clear, the conclusion is still not clear for m-PDTSP, MOSP, and graph-clear. In m-PDTSP, we observe that none of CABS and the LNBS configurations is a clear winner. In MOSP and graph-clear, CABS, LNBS/uniform, and LNBS/bias achieve the same primal gap in almost all instances. Therefore, we evaluate the DIDP solvers using larger instance sets of m-PDTSP, MOSP, and graph-clear. We describe instances used in this additional evaluation below.

In m-PDTSP (Section 3.3.2), a vehicle visits all customers, picks up some commodities at some customers, and delivers them to others. Each commodity has a weight and the total weight of

Table 5.12: Number of large instances of m-PDTSP, MOSP, and graph-clear where LNBS has a better/same/worse primal gap or primal integral than CABS. A value in 'Better'/'Worse' is in bold if it is larger than 'Worse'/'Better'.

	LNBS/uniform			LNBS/bias		
	Better	Same	Worse	Better	Same	Worse
Primal Gap						
m-PDTSP (240)	**71**	150	19	**88**	142	10
MOSP (760)	19	611	**130**	13	608	**139**
Graph-Clear (50)	11	10	**29**	7	10	**33**
Primal Integral						
m-PDTSP (240)	**143**	15	82	**150**	15	75
MOSP (760)	43	0	**717**	42	0	**718**
Graph-Clear (50)	2	0	**48**	3	0	**47**

commodities that a vehicle can carry is limited by the capacity. In the benchmark set for m-PDTSP, three types of instances are used: class1, class2, and class3 [208], and class1 instances are generated from instances of the sequential ordering problem (SOP) [11]. We generate larger class1 instances by using 30 SOP instances in TSPLIB[7] that were not used by the previous work. The original instances have at most 47 customers, and the new instances have 42 to 378 customers. We use the same methods as the previous work [208] with the maximum weight $q \in \{1, 5\}$ and the capacity $Q \in \{5q, 10q, 20q, 100q\}$, resulting in 240 instance in total.

In MOSP (Section 3.3.9), an instance is represented by a matrix, and the original set uses at most 125×125 matrices. We add instances using 150×150 to 1000×1000 matrices [64, 146].

For graph-clear, as explained in Section 5.3.2, each instance is associated with a graph. We generate 50 instances using random graphs with 100 and 200 nodes following Morin et al. [317].

We show the average primal gap, primal integral, and the coverage in Table 5.11. We also present the number of instances where the LNBS configurations have a better/same/worse primal gap or primal integral in Table 5.12. LNBS/uniform and LNBS/bias clearly outperform CABS in m-PDTSP in the primal gap and the primal integral, but CABS is better in MOSP and graph-clear. LNBS/uniform is slightly better than LNBS/bias in MOSP and graph-clear while LNBS/bias is better in m-PDTSP.

5.3.4 Ablation Study

The experimental results presented above show that there are no large differences on average between LNBS/uniform and LNBS/bias. To evaluate the importance of other components in LNBS, we consider two additional configurations: LNBS/conflicts does not remove transitions that conflict with the suffix (Section 5.2.2); LNBS/no-bandit selects the depth d of a neighborhood uniformly at random instead of using Budgeted-UCB. In both of them, other components are the same as LNBS/uniform. We evaluate these configurations using the eleven problem classes. For m-PDTSP, MOSP, and graph-clear, we use the larger instance sets introduced in Section 5.3.3. Tables 5.13–5.15 show the average primal gap, the average primal integral, and the coverage.

Compared to LNBS/uniform, LNBS/conflicts shows a better average primal gap and primal integral in m-PDTSP, OPTW, and MOSP. However, it is worse than LNBS/uniform in the remaining eight problem classes. In particular, LNBS/conflicts performs worse than baselines that are outper-

[7] `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/sop/`

Table 5.13: Average primal gap of MIP, CP, CABS, and LNBS configurations in each problem class. The primal gap of an LNBS configuration is in bold if it is lower than the baselines (MIP, CP, and CABS), and the lowest of the baselines is in bold if the LNBS configurations are worse. The lowest primal gap is underlined. For m-PDTSP, MOSP, and graph-clear, large instances introduced in Section 5.3.3 are used.

				LNBS/			
	MIP	CP	CABS	uniform	bias	conflicts	no-bandit
TSPTW (340)	0.2357	0.0259	0.0033	**0.0017**	**0.0021**	0.0034	0.0033
CVRP (207)	0.5950	0.2534	0.1752	**0.1626**	**0.1622**	**0.1666**	**0.1751**
m-PDTSP (240)	0.5774	0.1509	0.0744	**0.0718**	**0.0701**	**0.0699**	0.0745
OPTW (144)	0.2180	**0.0098**	0.0279	0.0314	0.0318	0.0352	0.0279
MDKP (276)	**0.0000**	0.0058	0.1097	0.1019	0.1036	0.1017	0.1097
Bin Packing (1615)	0.0417	**0.0015**	0.0017	0.0023	0.0023	0.0028	0.0017
SALBP-1 (2100)	0.2697	0.0046	**0.0002**	0.0021	0.0020	0.0022	0.0005
$1\|\|\sum w_i T_i$ (375)	0.0131	**0.0009**	0.0329	0.0047	0.0051	0.0209	0.0331
Talent Scheduling (1000)	0.0395	0.0087	0.0167	**0.0043**	**0.0042**	0.0122	0.0168
MOSP (760)	0.8810	0.0689	**0.0010**	0.0027	0.0029	0.0023	0.0011
Graph-Clear (50)	0.5233	0.5290	0.0012	0.0032	0.0040	0.0038	0.0012

Table 5.14: Average primal integral of MIP, CABS, CP, and LNBS configurations in each problem class. The primal integral of an LNBS configuration is in bold if it is lower than the baselines (MIP, CP, and CABS), and the lowest of the baselines is in bold if the LNBS configurations are worse. The lowest primal integral is underlined. For m-PDTSP, MOSP, and graph-clear, large instances introduced in Section 5.3.3 are used.

				LNBS/			
	MIP	CP	CABS	uniform	bias	conflicts	no-bandit
TSPTW (340)	509.18	48.97	9.25	**6.38**	**6.90**	10.45	9.39
CVRP (207)	1147.11	482.89	333.69	**319.99**	**318.93**	**325.02**	341.53
m-PDTSP (240)	1095.72	289.00	157.78	**153.63**	**151.97**	**150.63**	158.75
OPTW (144)	448.17	**23.89**	65.67	73.76	74.00	79.37	65.79
MDKP (276)	**0.11**	15.87	201.73	189.52	192.69	189.18	201.73
Bin Packing (1615)	88.32	7.92	**4.90**	8.37	8.26	8.85	5.34
SALBP-1 (2100)	536.41	28.52	**1.93**	6.96	6.90	7.36	2.77
$1\|\|\sum w_i T_i$ (375)	64.01	**3.49**	71.21	12.57	13.23	51.77	71.19
Talent Scheduling (1000)	122.30	30.32	36.77	**11.01**	**11.01**	29.42	37.52
MOSP (760)	1601.08	152.92	**5.25**	10.15	10.51	9.25	5.72
Graph-Clear (50)	963.06	1268.39	**8.70**	16.37	17.22	18.15	9.05

formed by LNBS/uniform. In TSPTW, LNBS/conflicts has a worse primal gap and primal integral than CABS. LNBS/conflicts also has a worse primal gap than MIP in $1||\sum w_i T_i$ and than CP in talent scheduling.

LNBS/no-bandit is similar to CABS: it has better coverage than LNBS/uniform in all problem classes and a better primal gap and primal integral in OPTW, bin packing, SALBP-1, MOSP, and graph-clear while LNBS/uniform has a better primal gap and primal integral in TSPTW, CVRP, m-PDTSP, MDKP, $1||\sum w_i T_i$, and talent scheduling. The average primal gap, the average primal integral, and the coverage of LNBS/no-bandit are similar to those of CABS. This result is probably because the largest depth, which makes LNBS the same as CABS, is more likely to be selected by uniform sampling.

In conclusion, removing transitions that conflict with the suffix improves the performance in eight out of the eleven problem classes. In particular, it is crucial to outperform baselines in three problem classes. Sampling a depth uniformly at random makes LNBS almost the same as CABS, so using Budgeted-UCB is essential to achieve different behavior.

Table 5.15: Coverage of MIP, CABS, CP, and LNBS configurations in each problem class. The coverage of an LNBS configuration is in bold if it is higher than the baselines (MIP, CP, and CABS), and the highest of the baselines is in bold if the LNBS configurations are worse. The highest coverage is underlined. For m-PDTSP, MOSP, and graph-clear, large instances introduced in Section 5.3.3 are used.

				LNBS/			
	MIP	CP	CABS	uniform	bias	conflicts	no-bandit
TSPTW (340)	222	47	259	242	242	235	259
CVRP (207)	**27**	0	6	6	6	6	6
m-PDTSP (240)	48	77	100	100	100	95	100
OPTW (144)	16	49	64	62	62	52	64
MDKP (276)	**165**	6	5	5	5	5	5
Bin Packing (1615)	1159	**1234**	1167	1140	1142	1140	1167
SALBP-1 (2100)	1423	1584	1802	1703	1706	1703	1802
$1\|\|\sum w_i T_i$ (375)	106	150	288	281	281	273	288
Talent Scheduling (1000)	0	0	239	236	237	189	**240**
MOSP (760)	0	0	**154**	148	148	103	151
Graph-Clear (50)	0	0	0	0	0	0	0

5.3.5 Analysis of Problem Characteristics

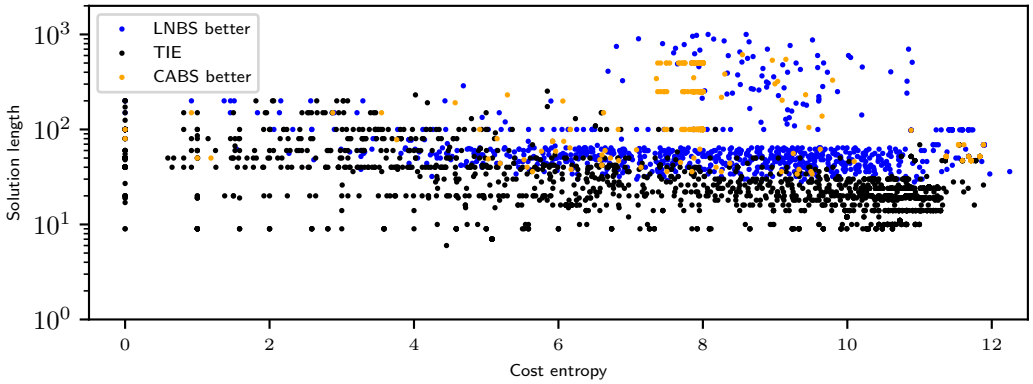
We have shown that LNBS/uniform and LNBS/bias are better than CABS in six problem classes, TSPTW, CVRP, (large instances of) m-PDTSP, MDKP, $1||\sum w_i T_i$, and talent scheduling. In contrast, the LNBS configurations are worse in OPTW, bin packing, SALBP-1, MOSP, and graph-clear. In this section, we investigate problem characteristics that may contribute to the performance difference between CABS and LNBS. We propose two hypotheses:

1. LNBS is better than CABS when path costs are diverse.
2. LNBS is better than CABS when search guidance is weak.

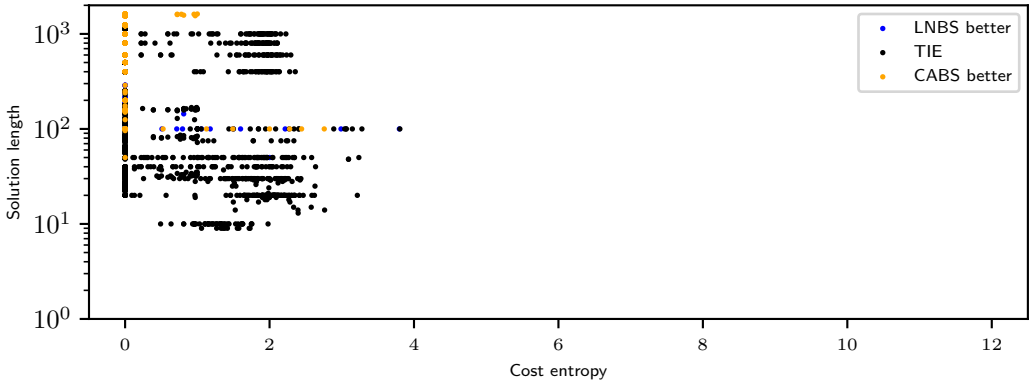
Diversity of Path Costs

One characteristic that might be beneficial for LNBS is the diversity of path costs. For example, in routing problems, i.e., TSPTW, CVRP, and m-PDTSP introduced in Sections 3.2.1, 3.3.1, and 3.3.2, where the LNBS configurations are better than CABS, a solution is a route visiting all customers, and its cost is the travel time of the route. In the DyPDL models for these problem classes, each transition corresponds to visiting one customer, and the cost of a partial path increases when a transition is applied. We expect that different partial solutions tend to have different costs; because the order in which customers are visited is the prime determinant of the path cost. It is also likely to be relatively easy to find improving partial paths because, unless the current partial path is optimal, better partial paths are included in a partial state transition graph with high density. In such a case, LNBS is likely to find a better partial path although it searches in a fraction of the partial state transition graph restricted by the beam width.

In contrast, in bin packing and SALBP-1 introduced in Sections 3.3.5 and 3.3.6, the problem is to pack items into capacitated bins (or schedule tasks in stations in SALBP-1) while minimizing the number of bins. In the DyPDL models, each transition packs one item into a bin, and the cost increases only when a new bin is opened. We expect that many partial paths tend to have the same cost, making it difficult to improve a solution by searching only a partial state transition graph. In the DyPDL models for MOSP and graph-clear, the cost is computed by taking the maximum weight



(a) Problem classes where partial path costs are diverse: TSPTW, CVRP, m-PDTSP, MDKP, $1||\sum w_i T_i$, and talent scheduling. All problem classes in this plot have instances with cost entropy larger than 6.



(b) Problem classes where partial path costs are not diverse: OPTW, bin packing, SALBP-1, MOSP, and graph-clear. All problem classes in this plot do not have instances with cost entropy larger than 4.

Figure 5.4: Entropy of the cost distribution over partial paths vs. the solution length in each problem instance. 'LNBS better' means LNBS/uniform finds a better solution, 'TIE' means that LNBS/uniform and CABS achieve the same solution cost, and 'CABS better' means that CABS finds a better solution.

of edges in a path, and so does not increase unless a new edge has a higher weight than the current maximum. Again, we expect that many partial paths share the same cost.

To quantitatively evaluate the diversity of partial path costs, we measure the diversity of costs in a partial state transition graph using entropy in information theory [390]. Given a solution for a DyPDL model $\sigma = \langle \sigma_1, ..., \sigma_n \rangle$, let $N_{di}(\sigma)$ be the set of solution paths whose prefix is $\langle \sigma_1, ..., \sigma_{i-1} \rangle$ and the suffix is $\langle \sigma_{i+d}, ..., \sigma_n \rangle$. Let $C = \{\mathsf{cost}_\psi(S^0) \mid \psi \in N_{di}(\sigma)\}$ be the set of the solution costs. Then, the entropy of the path costs is defined as follows:

$$H(N_{di}(\sigma)) = -\sum_{c \in C} \frac{|\{\psi \in N_{di}(\sigma) \mid \mathsf{cost}_\psi(S^0) = c\}|}{|N_{di}(\sigma)|} \log_2 \frac{|\{\psi \in N_{di}(\sigma) \mid \mathsf{cost}_\psi(S^0) = c\}|}{|N_{di}(\sigma)|}. \quad (5.8)$$

As this value gets larger, the cost distribution becomes more diverse, and we expect that LNBS will perform better than CABS. However, even if the entropy is large, if the problem itself is easy, both CABS and LNBS will find optimal or near-optimal solutions. To consider such cases, we also measure the length of the initial solution found by CABS.

We evaluate entropy and the length of the initial solution found for each problem instance used in Sections 5.3.1 and 5.3.3. We first run CABS until it finds a feasible solution and record the length of the solution. Then, we remove the first eight transitions from the solution and enumerate all feasible prefixes of the solution, i.e., we use $d = 8$ and $i = 1$. In Figure 5.4, we show a scatter plot of entropy and the solution length divided into two plots to emphasize the differences between the problem classes where partial path costs are diverse (TSPTW, CVRP, m-PDTSP, MDKP, $1||\sum w_i T_i$, and talent scheduling) and those where partial path costs are less diverse (OPTW, bin packing, SALBP-1, MOSP, and graph-clear). This classification is consistent with problem classes where LNBS is better/worse than CABS. With low entropy, CABS tends to be better. For higher entropy, the solution length makes differences: for short solutions, CABS and LNBS/uniform perform equally but for longer solutions, LNBS/uniform tends to perform better. This observation is consistent with our first hypothesis.

Strength of Search Guidance

Our second hypothesis is that LNBS performs better than CABS when search guidance is weak. We explain the intuition behind this hypothesis as follows. Beam search keeps the best b states according to the f-values. The f-value of state S is $f(S) = g(S) \times h(S)$, where $g(S)$ is the path weight to reach S, $h(S)$ is the estimated path cost by a heuristic function, and $\times \in \{+, \max\}$ is a binary operator. When f is weak, i.e., $f(S)$ is not well correlated with the optimal solution cost via S, beam search would make many mistakes in selecting the best states. In such a case, many short partial paths possibly have room for improvement, so LNBS would easily find better solutions. In contrast, if f is stronger, beam search makes fewer mistakes, and most partial paths do not have room for improvement. LNBS may become stuck in local minima, i.e., partial paths that cannot be improved further, while CABS will not since it always searches in the entire state transition graph.

To test this hypothesis, we evaluate CABS and LNBS/uniform with weaker heuristic functions in problem classes where CABS performs better than LNBS/uniform. Beam search uses the dual bound function η as the heuristic function h in our evaluation, i.e., $f(S) = g(S) \times \eta(S)$. Following Section 4.3.7, we use the zero dual bound function, which always returns 0 ($\eta(S) = h(S) = 0$). We compare CABS and LNBS/uniform with the zero dual bound function, called CABS/0 and LNBS/uniform/0, in bin packing and SALBP-1.

MOSP and graph-clear (Sections 3.3.9 and 3.3.10), where CABS is better than LNBS/uniform, cannot be used for this evaluation since their DyPDL models are already using the zero dual bound function. In these DyPDL models, the cost expression takes the maximum of edge weights ($\mathsf{cost}_\tau(x, S) = \max\{w_\tau(S), x\}$), so $f(S) = \max\{g(S), h(S)\} = \max\{g(S), 0\} = g(S)$. Since $f(S) = g(S)$ becomes exactly the same as the optimal solution cost via S at a point when a transition having the maximum weight is applied, we expect that beam search has strong search guidance even with the zero dual bound function.

In addition to the above problem classes, CABS is also better than LNBS/uniform in OPTW. In the DyPDL model for OPTW, since the objective is to maximize the nonnegative total profit, the zero dual bound function is not applicable. Therefore, we evaluate CABS and LNBS/uniform without the dual bound function, called CABS/blind and LNBS/uniform/blind. In these configurations, beam search is guided only by the g-value.

Table 5.16 presents the average primal gap and primal integral of CABS, LNBS/uniform, CABS/0,

Table 5.16: Average primal gap and primal integral of CABS and LNBS/uniform in each instance set of bin packing and SALBP-1 using the original and zero dual bound functions. 'gap' is the primal gap at the time limit, and 'integral' is the primal integral. The value is in bold if one is strictly better than the other.

	Original dual bound				Zero dual bound			
	CABS		LNBS/uniform		CABS/0		LNBS/uniform/0	
	gap	integral	gap	integral	gap	integral	gap	integral
Bin Packing Total (1615)	**0.0018**	**4.94**	0.0023	8.41	0.0058	**11.46**	**0.0054**	12.47
Falkenauer T (80)	**0.0073**	**19.62**	0.0085	47.25	0.0232	**44.77**	**0.0228**	55.19
Falkenauer U (80)	**0.0005**	**5.26**	0.0025	9.89	0.0078	15.95	**0.0066**	**15.78**
Scholl 1 (720)	0.0003	**1.23**	0.0003	1.75	0.0019	3.84	**0.0013**	**3.08**
Scholl 2 (480)	**0.0033**	**8.34**	0.0045	13.00	0.0075	**14.44**	**0.0072**	15.95
Scholl 3 (10)	0.0035	6.79	0.0035	8.05	0.0106	**19.43**	0.0106	20.41
Wäscher (17)	**0.0070**	**22.32**	0.0115	33.72	0.0445	**80.32**	0.0445	80.34
Schwerin 1 (100)	0.0021	**4.36**	**0.0016**	4.64	0.0063	**15.35**	0.0063	20.93
Schwerin 2 (100)	0.0005	**2.18**	0.0005	3.86	0.0045	**8.54**	0.0045	9.25
Hard28 (28)	0.0000	**0.12**	0.0000	0.20	0.0000	**0.12**	0.0000	0.17
SALBP-1 Total (2100)	**0.0002**	**1.92**	0.0021	6.96	0.0100	19.43	**0.0096**	**19.30**
Small $n = 20$ (525)	0.0000	0.01	0.0000	0.01	0.0000	0.01	0.0000	0.01
Medium $n = 50$ (525)	0.0000	0.06	0.0000	**0.05**	0.0002	**0.58**	0.0002	0.71
Large $n = 100$ (525)	**0.0007**	**1.88**	0.0009	2.37	0.0062	**14.01**	0.0062	14.27
Very Large $n = 1000$ (525)	**0.0002**	**5.74**	0.0074	25.38	0.0338	63.10	**0.0320**	**62.21**

Table 5.17: Number of instances where LNBS has a better/same/worse primal gap than CABS in bin packing and SALBP-1 with the original and zero dual bound functions. We compare LNBS/uniform with CABS and LNBS/uniform/0 with CABS/0. A value in 'Better'/'Worse' is in bold if it is larger than 'Worse'/'Better'.

	LNBS/uniform			LNBS/uniform/0		
	Better	Same	Worse	Better	Same	Worse
Primal Gap						
Bin Packing (1615)	9	1529	**77**	**50**	1545	20
SALBP-1 (2100)	3	1801	**296**	**210**	1825	65
Primal Integral						
Bin Packing (1615)	50	0	**1565**	104	0	**1511**
SALBP-1 (2100)	191	0	**1909**	291	0	**1809**

and LNBS/uniform/0 in each instance set of bin packing and SALBP-1. In bin packing, with the original dual bound function, CABS achieves a better primal gap in four out of the nine instance sets and is worse only in Schwerin 1. With the zero dual bound function, LNBS/uniform/0 has a better primal gap than CABS/0 in four out of the nine instance sets and an equal primal gap in other sets. We also present the number of bin packing instances where LNBS/uniform is better/same/worse than CABS and the number of instances where LNBS/uniform/0 is better/same/worse than CABS/0 in Table 5.17. The number of instances where LNBS/uniform/0 has a better primal gap than CABS/0 (50) is larger than the number of instances where LNBS/uniform/0 is worse (20). While CABS/0 is better than LNBS/uniform/0 in terms of the average primal integral, it is mainly because CABS/0 has a better primal integral than LNBS/uniform/0 in most instances where they achieve the same primal gap (1483 out of 1545). In 42 out of 50 instances where LNBS/uniform/0 has a better primal gap than CABS/0, LNBS/uniform/0 has a better primal integral.

Each instance set of SALBP-1 is introduced by Morrison, Sewell, and Jacobson [319] and is composed of instances with the same size; the Small instances have 20 tasks, the Medium instances have 50 tasks, the Large instances have 100 tasks, and the Very Large instances have 1000 tasks.

Table 5.18: Average primal gap and primal integral of CABS and LNBS/uniform in each instance set of OPTW with and without the dual bound functions. 'gap' is the primal gap at the time limit, and 'integral' is the primal integral. The value is in bold if one is strictly better than the other.

	Original dual bound				No dual bound			
	CABS		LNBS/uniform		CABS/blind		LNBS/uniform/blind	
	gap	integral	gap	integral	gap	integral	gap	integral
OPTW Total (144)	**0.0279**	**65.76**	0.0314	73.76	0.0024	**8.35**	**0.0023**	9.06
RS2006 Solomon (29)	**0.0000**	**2.53**	0.0011	2.71	0.0000	0.12	0.0000	0.12
RS2008 Solomon (29)	0.0028	**12.30**	**0.0026**	14.83	0.0000	**1.16**	0.0000	1.43
RS2008 Cordeau (10)	**0.0184**	**61.05**	0.0310	81.07	**0.0000**	**7.39**	0.0004	9.88
MG2009 Solomon (27)	**0.0918**	**190.04**	0.0958	203.96	0.0130	32.86	**0.0117**	**31.43**
MG2009 Cordeau (10)	**0.1008**	**219.81**	0.1152	254.59	**0.0000**	**8.83**	0.0008	16.48
VSVV2009 Solomon (29)	0.0028	**12.33**	**0.0026**	14.82	0.0000	**1.11**	0.0000	1.41
VSVV2009 Cordeau (10)	**0.0188**	**62.18**	0.0292	81.98	**0.0000**	**8.37**	0.0009	10.68

Table 5.19: Number of instances where LNBS has a better/same/worse primal gap than CABS in OPTW with and without the dual bound function. We compare LNBS/uniform with CABS and LNBS/uniform/blind with CABS/blind. A value in 'Better'/'Worse' is in bold if it is larger than 'Worse'/'Better'.

	LNBS/uniform			LNBS/uniform/blind		
	Better	Same	Worse	Better	Same	Worse
Primal Gap in OPTW (144)	14	103	**27**	8	126	**10**
Primal Integral in OPTW (144)	21	0	**123**	51	0	**93**

In SALBP-1, while CABS is better in Large and Very Large with the original dual bound function, LNBS/uniform/0 is better than CABS/0 in Very Large and achieves the same primal gap in other sets. This result is consistent with our observation in TSPTW that LNBS/uniform performs better than CABS in more difficult instances (Section 5.3.2). The number of instances where LNBS/uniform/0 has a better primal gap than CABS/0 (210) is larger than the number of instances where LNBS/uniform/0 is worse (65).

Table 5.18 and Table 5.19 show the results for CABS, LNBS/uniform, CABS/blind, and LNBS/uniform/blind in OPTW. There are seven instance sets, one introduced by Righini and Salani [356] (RS2006 Solomon), two introduced by Righini and Salani [357] (RS2008 Solomon and RS2008 Cordeau), two introduced by Montemanni and Gambardella [315] (MG2009 Solomon and MG2009 Cordeau), and two introduced by Vansteenwegen et al. [434] (VSVV2009 Solomon and VSVV2009 Cordeau). The instance sets with 'Solomon' in their names are based on the instances of the vehicle routing problem with time windows [102] by Solomon [405], and the instance sets with 'Cordeau' in their names are based on the instances of the multi-depot vehicle routing problem [316] by Cordeau, Gendreau, and Laporte [90]. CABS/blind and LNBS/uniform/blind achieve a better primal gap and primal integral than CABS and LNBS/uniform in all instance sets. This result shows that the overhead to compute the dual bound function in the original DyPDL model does not pay off in terms of finding a good solution. However, we observe that using the dual bound function is beneficial in terms of the average optimality gap (0.2696 for CABS, 0.2855 for LNBS/uniform, 0.5208 for CABS/blind, and 0.5278 for LNBS/uniform/blind). LNBS/uniform/blind achieves a worse average primal gap than CABS/blind in RS2008 Cordeau, MG2009 Cordeau, and VSVV2009 Cordeau and a worse average primal integral in all instance sets except for MG2009 Solomon. In MG2009 Solomon, LNBS/uniform/blind outperforms CABS/blind in the primal gap and primal integral. The number of instances where LNBS/uniform/blind is better than CABS/uniform/blind is less than the

number of instances where LNBS/uniform/blind is worse. However, compared to LNBS/uniform and CABS, in a larger number of instances, LNBS/uniform/blind achieves the same primal gap as CABS/blind and a better primal integral than CABS/blind. Overall, CABS/blind is still better than LNBS/uniform/blind in many cases, but the performance gap between them is closer than that of LNBS/uniform and CABS.

In summary, with the weaker heuristic functions, we observe that LNBS/uniform/0 (LNBS/uniform/blind) becomes better than or closer to CABS/0 (CABS/blind) in bin packing, SALBP-1, and OPTW. These results are consistent with our second hypothesis that LNBS performs better when search guidance is weaker. In bin packing, SALBP-1, and OPTW, as shown in Figure 5.4b, the cost entropy is relatively small. Therefore, the results in these problem classes are counterexamples for our first hypothesis that LNBS performs better when partial path costs are diverse.

Further investigation of the hypothesis, e.g., evaluating CABS and LNBS using heuristic functions with parametrized strength, is future work. Generalizing this analysis to other types of LNS is also an interesting direction. For example, for LNS in CP, we may analyze the performance of LNS with different strengths of value and variable selection heuristics, which guide the search; when these heuristics make fewer mistakes, many neighborhoods may not have room for improvement.

5.4 Discussion

Through the empirical evaluation, we have shown that LNBS/uniform and LNBS/bias perform better than CABS in six of the eleven problem classes. LNBS has three components to select a neighborhood: selecting the depth of the partial path, selecting the starting point of the partial path, and selecting the beam width. Given a neighborhood, LNBS removes conflicting transitions considering the suffix, the transitions after the selected partial paths. While we have presented the importance of the current configuration in the ablation study (Section 5.3.4), each component has room for improvement or alternative design choices. It may also be useful to exploit the problem characteristics that affect the relative performance of LNBS compared to CABS (Section 5.3.5), during search.

As shown in Section 5.3.4, removing conflicting transitions (Section 5.2.2) improves the performance in most problem classes. To remove conflicting transitions, we use Proposition 1, which focuses on transitions that add/remove one element to/from a set variable. While this method is applicable to ten out of the eleven problem classes evaluated, to handle broader classes of problems, we may want to generalize Proposition 1. A straightforward extension would be considering transitions that add/remove multiple elements to/from a set variable. Furthermore, considering numeric variables may also be possible. For example, if a numeric variable represents the remaining capacity, and each transition in the suffix decreases the value of the variable, we may exclude some transitions based on the capacity constraint.

The ablation study in Section 5.3.4 also shows that LNBS/no-bandit, which selects a depth uniformly at random, is not much different from CABS. In other words, the depth selection mechanism using Budgeted-UCB (Section 5.2.3) makes LNBS/uniform and LNBS/bias different from CABS. While using completely different strategies to select a depth might be possible as alternative choices, variants of Budgeted-UCB are worth considering: Budgeted-UCB uses Equation (5.2) to balance exploration and exploitation, so different behavior can be obtained by scaling each term, for example.

Compared to the depth selection mechanism, other components use relatively simple strategies in the current configurations. To select the starting point of a partial path, uniform sampling and cost-biased sampling (Section 5.2.4) are considered; however, they do not make much difference in practice. All evaluated LNBS configurations double the beam width after selecting a neighborhood, following CABS (Section 5.2.5). The current solution (σ in Algorithm 12) is updated only when an improving solution is found. For these components, similar to Budgeted-UCB, it may be possible to develop adaptive strategies that consider statistics so far, following previous work in LNS [368]. Furthermore, we may consider using other heuristic search algorithms than beam search, e.g., A* [190], to search for a better partial path.

In Section 5.3.5, we have observed that CABS performs better than LNBS when the search guidance is strong. While Budgeted-UCB may eventually converge to selecting larger depths in such a case, measuring the strength of the search guidance during search and explicitly switching to CABS or larger depths might be beneficial.

5.5 Related Work

We discuss the novelty of LNBS compared with existing methods. In particular, we consider related work in the literature from the following two perspectives: state space search algorithms that search in a neighborhood and LNS algorithms using multi-armed bandits.

5.5.1 State Space Search in a Neighborhood

There exist several state space search algorithms that improve a solution path by searching in a neighborhood. However, except for DD-LNS, they were not framed as LNS or applied to combinatorial optimization.

Ratner and Pohl [352] proposed local path A* (LPA*) and applied it to the 15-puzzle. They consider the shortest path problem in an unweighted graph where the length of the path is equivalent to the path cost. Similar to LNBS, LPA* tries to find a better partial path between two nodes on a given path, but they only consider partial paths with a fixed length d. Given a feasible path $\langle (S^0, S^1), ..., (S^{n-1}, S^n) \rangle$, LPA* maintains the starting point of a partial path i, which is initialized with 0. At each iteration, LPA* runs A* to find a shortest path from S^i to S^{i+d}. If the newly found partial path is shorter, then LPA* runs A* to update subsequent partial paths with length d: each partial path $\langle (S^j, S^{j+1}), ..., (S^{j+d-1}, S^{j+d}) \rangle$ for $j = i + d, i + 2d, ..., i + md$, where m is the maximum integer such that $i + (m + 1)d \leq n$, is replaced with the shortest partial path found by A*. If a better partial path from S^i to S^{i+d} is not found, LPA* updates i to $i + \delta$ and repeats the procedure, where δ is a constant parameter. Therefore, LPA* can be viewed as an instantiation of LNBS, where the size of a neighborhood is fixed, the starting point of a partial path is selected by the above strategy, and A* is used instead of beam search.

Ratner and Pohl [352] also proposed Joint, another heuristic search algorithm in a neighborhood. Joint first divides a feasible path into consecutive segments with length d, replaces them with shortest partial paths, and tries to improve partial paths connecting two segments. Given a feasible path $\langle (S^0, S^1)..., (S^{n-1}, S^n) \rangle$, each partial path $\langle (S^j, S^{j+1}), ..., (S^{j+d-1}, S^{j+d}) \rangle$ for $j = 0, d, 2d, ..., md$, where m is the maximum integer such that $(m + 1)d \leq n$, is replaced with the shortest partial path found by A*. The set of nodes called joints are initialized with $J = \{S^{j+d} \mid j = 0, d, 2d, ..., md\}$. At

each iteration, Joint selects a joint $S^i \in J$ with the minimum i and tries to improve a partial path from $S^{i-d/2}$ to $S^{i+d/2}$. Once S^i is selected, joints far from $S^{i+d/2}$ are removed. Concretely, a joint S^j with $j < i + d/2 - \delta$ is removed, where δ is a constant parameter. In practice, they used $\delta < d/2$, so the selected joint S^i is always removed. Joint replaces a partial path from $S^{i-d/2}$ to $S^{i+d/2}$ by the shortest path found by A*. If the newly found path is better, $S^{i-d/2}$ and $S^{i+d/2}$ are added to J. This procedure is repeated until J becomes empty. Similar to LPA*, Joint can be considered an instantiation of LNBS with a fixed neighborhood size, the special mechanism to select a starting point, and A* instead of beam search.

Iterative tunneling A* (ITA*) [149] searches in a neighborhood constructed by including states reachable from a given path with a limited number of edges. This neighborhood construction is different from LNBS, DD-LNS, LPA*, and Joint, which remove a partial path from a given path. Given a feasible path $\langle (S^0, S^1), ..., (S^{n-1}, S^n) \rangle$, ITA* runs A* from S^0 to S^n to find the shortest path in a neighborhood of the path, which grows iteratively. Intuitively, the neighborhood in iteration j contains states that are reachable from the states on the current path with at most j edges. Nodes on the current path, S^0, ..., S^n, are assigned an iteration number of 0. A successor state of a state whose iteration number is k is assigned an iteration number $k + 1$ when it is first generated. A* in the j-th iteration only expands states whose iteration number is less than or equal to j. Starting from $j = 1$, ITA* repeats this procedure until reaching the memory limit. ITA* was applied to the 48-puzzle and Rubik's Cube.

Plan Neighborhood Graph Search (PNGS) is a method to improve a plan in classical AI planning [324]. PNGS first constructs a neighborhood graph and then finds a shortest path in the graph. Similarly to ITA*, PNGS is different from LNBS in that it does not construct a neighborhood by removing a partial path. Given a feasible path $\langle (S^0, S^1), ..., (S^{n-1}, S^n) \rangle$, PNGS runs a state space search algorithm from each node S^j for $j = 0, ..., n$ until expanding L nodes. The neighborhood graph includes the expanded nodes and edges traversed by the search algorithm. PNGS runs a shortest path algorithm in the neighborhood graph to find a solution. An anytime version of PNGS iteratively runs this procedure while increasing L. The original paper used A* or A* together with backward breadth-first search to construct the neighborhood graph and Dijkstra's algorithm to find a shortest path in the neighborhood.

As explained in Section 5.1.1, DD-LNS [170] was applied to combinatorial optimization and can be considered a state space search algorithm and an instantiation of LNBS. Although it tries to find a better partial path starting from the middle of a given path, the suffix must be empty, unlike LNBS. In addition, DD-LNS uses a fixed beam width.

5.5.2 Multi-Armed Bandits for Large Neighborhood Search

The multi-armed bandit problem [364] is a decision-making problem with various application fields including clinical trials, recommendation systems, and algorithm selection [50]. In a multi-armed bandit problem, we are given a set of arms D. At each round k, we pull one arm $d \in D$ and obtain a reward r_{dk}, which is a random variable following an unknown probability distribution associated with the arm d. Let a_k be the arm selected by strategy a at round k, and $r_{a_k k}$ be the reward obtained at round k. The objective is to find a strategy to maximize the total expected reward over K rounds, $\mathbb{E}\left[\sum_{k=1}^{K} r_{a_k k}\right]$. In this paper, we used a variant called the budgeted bandit problem [423], where pulling arm a incurs the cost t_a, and the number of trials is limited by the budget T.

In particular, we used the budgeted bandit problem with continuous random costs [442], where t_a is a continuous random variable following an unknown distribution (Section 5.2.3).

Previous work has combined multi-armed bandits with LNS, but the budgeted bandit problem was not used. Hendel [205] used a multi-armed bandit algorithm with LNS in a general-purpose MIP solver. In MIP, a neighborhood corresponds to a subproblem, where a subset of integer decision variables are assigned fixed values [94, 40]. As a repair heuristic, LNS runs branch-and-bound for the subproblem where the number of nodes is limited by an adaptively adjusted parameter. Hendel formulated a problem to select a subproblem from multiple candidates generated by different destroy heuristics as a multi-armed bandit problem. Each subproblem corresponds to an arm, and the reward depends on the result of branch-and-bound. They used a reward function combining the following three reward functions, all of which take a value in $[0, 1]$: r^{sol} returns 1 if a better solution is found or optimality is proved and 0 otherwise; $r^{\text{gap}} = \frac{\overline{c}-c}{\overline{c}-\underline{c}}$, where $\overline{c}$ is the current primal bound, $\underline{c}$ is the current dual bound, and c is the updated primal bound; r^{fail} returns 1 if a better solution is found or optimality is proved and returns a value negatively proportional to the number of explored branch-and-bound nodes otherwise. While r^{gap} is similar to our reward function in that it considers the cost improvement, they also incorporated the time taken by the subproblem in the reward function using r^{fail}, instead of separately considering it using the budgeted multi-armed bandit.

Chmiela et al. [75] also used multi-armed bandit with LNS in a general-purpose MIP solver. Differently from Hendel, they used a multi-armed bandit algorithm to select a primal heuristic from multiple candidates, some of which are LNS algorithms. Chmiela et al. used reward functions similar to Hendel, one of which is proportional to the cost improvement, and another is negatively proportional to the number of explored nodes.

Phan et al. [338] used multi-armed bandit in LNS for multi-agent path finding (MAPF), a problem in finding collision-free paths for multiple agents [407]. In their setting, the objective is to minimize the total delay, where the delay for each agent is the difference between the actual path length including waiting and the shortest path length ignoring collision. Given a set of paths for the agents, LNS selects a subset of the agents using a destroy heuristic and tries to improve the paths for them using a repair heuristic [290]. The repair heuristic is prioritized planning [401], which computes a path for each agent one by one using heuristic search in a space-time graph, considering already computed paths of other agents as obstacles. Thus, LNS for MAPF can be considered another form of a combination of LNS and state space search. Phan et al. used multi-armed bandit algorithms in two levels: selecting a destroy heuristic and selecting the number of agents for the destroy heuristic to select, i.e., the neighborhood size. Similar to our LNBS configuration, the number of agents is selected from $\{2^e \mid e \in 1, ..., E\}$, where E is a parameter. In both levels, the reward is defined as the cost improvement, $\overline{c} - c$, and the time taken by each arm is not considered.

5.6 Summary

We proposed large neighborhood beam search (LNBS), a state space search algorithm based on large neighborhood search (LNS) and beam search. Our configuration of LNBS exploits the multi-armed bandit problem and random sampling to select a neighborhood. As a domain-independent dynamic programming (DIDP) solver, LNBS finds better quality solutions on average than complete anytime beam search (CABS) in six out of the eleven benchmark problem classes. In particular, our analysis

suggests that LNBS performs better than CABS when the search guidance is weaker. A deeper investigation of the characteristics of the problems that make LNS effective in state space search and tree search is an interesting direction for future work. Based on such analysis, improving each component of LNBS may also be possible.

Chapter 6

Parallel Beam Search

In the previous two chapters, we developed domain-independent dynamic programming (DIDP) solvers and demonstrated their promising performance using a number of combinatorial optimization problem classes. However, unlike state-of-the-art general-purpose solvers for other paradigms, such as mixed-integer programming (MIP) [188] and constraint programming (CP) [277], the DIDP solvers cannot utilize multiple threads. To meet this standard, in this chapter, we develop multi-thread DIDP solvers.

The currently developed DIDP solvers are based on heuristic search algorithms studied in artificial intelligence (AI). In the AI community, parallelization of state space search algorithms, which include heuristic search algorithms, has been actively studied over the past three decades [263, 127, 303, 367, 448, 451, 436, 55, 253, 339, 235, 272, 271, 320, 322, 443, 395]. We leverage such previous work in the AI community to develop multi-thread DIDP solvers.

As a base solver, we focus on the state-of-the-art DIDP solver, complete anytime beam search (CABS) [447] (Section 4.2.7), which is based on beam search. We propose two types of parallel beam search algorithms using existing parallelization mechanisms for state space search. The experimental results show that our multi-thread DIDP solvers achieve significant performance improvement over sequential CABS. In addition, our solvers perform better and achieve a higher average speedup than commercial multi-thread MIP and CP solvers in multiple combinatorial optimization problem classes. Our multi-thread DIDP solvers are 9 to 36 times faster on average than sequential CABS using 32 threads; they sometimes achieve super linear speedups in two of the six problem classes tested. We also observe that the multi-thread solvers are slower than sequential CABS in some instances of these problem classes. Our experimental analysis suggests that a strong dual bound function makes parallel CABS diverge from sequential CABS while our solvers benefit from this phenomenon on average.

We introduce parallelization mechanisms for state space search in Section 6.1 and propose parallel beam search algorithms based on them in Section 6.2. We empirically evaluate the proposed algorithms in Section 6.3. Section 6.4 provides a review of parallel state space search algorithms, covering a broader range of work than Section 6.1. Finally, Section 6.5 summarizes the contribution of this chapter.

This work is based on a conference paper published in the *Proceedings of the AAAI Conference on Artificial Intelligence* [269]. We extend the paper with the following points:

- We present formal proofs of the correctness of the proposed algorithms.
- We include detailed experimental results.
- We provide a broader and deeper literature review of related work.

6.1 Parallel State Space Search

In this section, we introduce existing parallelization mechanisms for state space search related to our parallel beam search algorithms. First, we recap sequential state space search algorithms and explain shared memory and distributed environments. Then, we introduce three mechanisms: layer synchronization, a shared hash table for duplicate detection, and hash-based work distribution. We give a broader review of existing parallel state space search algorithms in Section 6.4.

6.1.1 Sequential State Space Search Algorithms

A state space search algorithm searches for a path in a state transition graph between two states, the initial state and a goal state. For the sake of simplicity, we assume minimization, i.e., the shortest path problem, in this chapter, but state space search algorithms can be adapted to maximization under some conditions as described in Section 4.1.4. As shown in Algorithm 9 of Section 4.1.4, a state space search algorithm maintains an open list that contains the set of candidate states to search. At each iteration, the algorithm expands one state from the open list, i.e., removes the state from the open list, generates its successor states, and inserts them into the open list. The order in which states are expanded from the open list depends on the specific algorithm. Breadth-first search (BrFS) searches layer by layer, i.e. expands all states having the same depth (the number of edges to reach the states) and then goes to the next depth. Best-first search (BFS) expands a state S that minimizes the priority value, denoted $f(S)$ (the f-value). For example, A* [190] (Section 4.2.1) is a BFS algorithm that uses $f(S) = g(S) + h(S)$, where $g(S)$ (the g-value) is the current best path cost to reach S, and $h(S)$ (the h-value) is an estimated path cost from S to a goal state by a heuristic function h. Beam search (Algorithm 10 in Section 4.2.7) can be viewed as a hybridization of BrFS and BFS. It searches layer by layer similar to BrFS but keeps only the best b states according to the f-values, where b is a parameter called *beam width*. A* and beam search are heuristic search algorithms since they use a heuristic function to guide the search. In contrast, BrFS does not use a heuristic function to decide what state to expand next. However, if an admissible heuristic function, which always underestimates the optimal path cost from a given state, and an upper bound on the optimal cost are given, BrFS can prune states based on f-values. This algorithm is called breadth-first heuristic search (BFHS) [450].

Typically, a state space search algorithm performs duplicate detection: if a generated state S (or a state dominating S) has already been expanded or is included in the open list with a better g-value, S is not added to the open list. The standard approach is to store generated states in a hash table, as described in Russell and Norvig [372]. A search algorithm checks if a generated state is already included in the hash table and stores it if not.

6.1.2 Shared Memory and Distributed Environments

Parallelization uses multiple cores simultaneously to accelerate an algorithm. There are mainly two types of environments for parallelization: shared memory and distributed [331]. In a shared memory environment, multiple threads execute an algorithm in parallel. The threads share the same memory, so communication between threads can be achieved by using shared data structures. In a distributed environment, multiple processes work together in parallel, but each process has its own memory and cannot directly access others'. Communication between processes is achieved by message passing, where a process sends data in its memory as messages to other processes. Typically, the message passing interface (MPI) [309] is used, which enables a process to send messages through network communication. Message passing can also be implemented for multiple threads in a shared memory environment, e.g., a thread sends a message by putting it on the message queue of another thread. Thus, parallel algorithms developed for distributed environments can usually be implemented in shared memory environments but not vice versa.

6.1.3 Layer Synchronization

Zhang and Hansen [448] proposed a parallel BFHS algorithm for a shared memory environment using layer synchronization: all threads expand states in the same layer in parallel and then proceed to the next layer. Layer synchronization is motivated by layered duplicate detection [450], which reduces memory consumption by storing only states in a small number of layers for duplicate detection. At the beginning of each layer, a single control thread distributes states in the layer to multiple worker threads. Each worker thread stores the states in its local open list and expands them. When a worker thread finishes expanding states, it needs to wait until other threads finish expansions. To reduce the waiting time, Zhang and Hansen introduced a dynamic load balancing mechanism, which allows a worker thread to send a subset of its open list to another worker thread that is waiting.

Frohner et al. [148] used layer synchronization to parallelize beam search in a shared memory environment. They parallelized a for loop to expand all states in the current layer using the Threads module of Julia,[1] which manages the assignments of threads to loop iterations. After generating states in the next layer, the f-value of the b-th best state is identified by a parallel counting sort algorithm, and states having worse f-values are removed.

6.1.4 Shared Hash Tables for Duplicate Detection

Parallel state space algorithms usually perform duplicate detection in parallel. One approach to parallelize duplicate detection in a shared memory environment is to use a single shared hash table. In this approach, multiple threads expand states and perform duplicate detection simultaneously in parallel with the shared hash table while avoiding contention using some mechanism. A simple approach is to make sure that only one thread accesses the hash table at a time using a mutually exclusive lock [436, 122].

Locking the whole hash table may be too restrictive; it is safe that different threads access different keys in the hash table simultaneously. Based on such ideas, concurrent data structures have been designed to be accessed by multiple threads in parallel. The parallel BFHS algorithm by Zhang and Hansen [448] uses a concurrent hash table for duplicate detection. Inspired by the Java

[1] `https://docs.julialang.org/en/v1/manual/multi-threading/`

concurrency package,[2] their concurrent hash table is divided into multiple segments, each of which is a hash table with a lock. A key is uniquely assigned to a segment based on the hash value. Different threads can mutate different segments in parallel, and multiple threads can read values from the same segment in parallel as long as no thread is mutating that segment. The parallel beam search algorithm by Frohner et al. [148] also employs a concurrent hash table, implemented as an array of linked lists with a lock for each linked list.

A concurrent hash table can also be implemented without locks. A lock-free data structure is a concurrent data structure that does not use locks and guarantees that at least one thread completes its operation after a finite number of time steps [206]. Typically, lock-free data structures are implemented with atomic compare-and-swap operations. Lock-free hash tables were used for duplicate detection in BFS by previous work [55, 271].

6.1.5 Hash-Based Work Distribution

An alternative approach to parallelize duplicate detection is hash-based work distribution [127], designed for distributed environments. Hash-based work distribution combines parallel duplicate detection with static load balancing. Each process has its local open list and expands states from it. However, the generated states are not necessarily inserted into the local open list. When a state is generated, it is sent to its owner process, uniquely determined by the hash value of the state. Given the hash value $\textsc{hash}(S)$, the owner of state S is process $i \in \{1, ..., k\}$ with $i = (\textsc{hash}(S) \mod k) + 1$. Since the same state is always sent to the same process, duplicate detection is performed by each process independently using a local hash table. When receiving a state, each process checks the local hash table and inserts the state into its local open list if there is no duplicate. If the hash function uniformly distributes states to hash values, the workload is uniformly distributed to all processes.

In the original paper by Evett et al. [127], message passing in hash-based work distribution was performed synchronously: a sender process waits until the message is received by the receiver process. Romein et al. [367] introduced asynchronous message passing: a sender process does not wait until the message is received. On the receiver side, each process periodically checks if there are incoming messages and receives them if they exist. Kishimoto, Fukunaga, and Botea [253] proposed hash-distributed A* (HDA*), a distributed parallel A* algorithm using hash-based work distribution with asynchronous message passing.

6.2 Parallel Beam Search for DIDP

Using the ideas presented in Section 6.1, we propose two parallel beam search algorithms: shared beam search (SBS) and hash-distributed beam search (HDBS). Both algorithms use layer synchronization (Section 6.1.3), i.e., all threads expand states in the same layer in parallel. We have the same motivation as Zhang and Hansen [448] to use layer synchronization: our sequential CABS solver for DIDP consumes less memory as it keeps only states in the current layer, which results in the superior performance to other DIDP solvers as we observed in Section 4.3.5. We present the parallel beam search algorithms after explaining our baseline implementation of sequential beam search for DIDP.

[2] `https://gee.cs.oswego.edu/dl/concurrency-interest/index.html`

6.2.1 Sequential Beam Search Implementation for DIDP

The pseudo-code of beam search for a Dynamic Programming Description Language (DyPDL) model is presented in Algorithm 10 of Section 4.2. For each state in the open list, beam search generates successor states (line 13) and inserts them into the set G (line 20) if they are not dominated by a state in G having an equal or smaller g-value (line 15). If a successor state dominates a state S' in G having an equal or larger g-value (line 17), S' is removed from G (line 18). These procedures work as duplicate detection since a state dominates itself. After expanding all states in G, the best b states minimizing the f-values are kept (line 24). In practice, we use $f(S) = g(S) \times h(S)$, where $\times \in \{+, \max\}$ is a binary operator in a monoid such that the cost expression of each transition τ is represented as $\mathsf{cost}_\tau(x, S) = w_\tau(S) \times x$ using a numeric expression w_τ (Definition 21 in Section 23).

In DyPDL, a state is defined by state variables (Definitions 1 and 2 in Section 3.1). In our modeling language, YAML-DyPDL (Section 3.2), an approximate dominance relation (Definition 17 in Section 3.1.2) is defined by resource variables (Definition 28 in Appendix A.1.1). A state S dominates another state S', denoted by $S' \preceq_a S$, if the values of resource variables in S are better than those of S'. In addition, a dual bound function η, which returns a dual bound (lower bound for minimization) on the optimal S-solution cost given a state S, can be defined in a YAML-DyPDL model. Our current DIDP solvers developed in Chapters 4 and 5 use $h(S) = \eta(S)$ and thus $f(S) = g(s) \times \eta(S)$ if η is defined. Otherwise, they do not use h, i.e., $f(S) = g(S)$.

In our implementation, a data structure called a *search node* has a pointer to a state and its g-h-, and f-values. The set G is represented by a hash table where the key is the values of non-resource variables and the hash-table entry is a list of search nodes. When a state is generated, a new search node for the state is created, and compared with existing search nodes in the corresponding hash-table entry. If the new node dominates an existing node in the list, it replaces the dominated one. Otherwise, the new node is appended to the list if it is not dominated.

To save memory and computation, selecting the best b states is performed incrementally. Search nodes are stored in a binary heap in descending order of f-values, and ties are broken by h-values. When a new search node is generated, if the number of states in G is equal to b, and the f- and h-values are worse than or equal to those of the top of the binary heap, then the node is discarded. Otherwise, we perform duplicate detection as described in the previous paragraph. A search node also has a binary flag indicating if the state is included in G. When state S' is removed from G due to dominance, the flag in the search node for S' is set to be false, and a counter tracking the number of states in G is decremented. The dominated search node is removed as soon as it becomes the top of the binary heap. After dominance detection, if G still contains b states, the top of the binary heap is removed before inserting the new search node into the binary heap.

As described in Section 4.3.1, we compute the dual bound on the optimal cost using Theorem 20 in Section 4.2.7. After expanding all states in the current layer (just after line 22 in Algorithm 10), we compute $\underline{\gamma}^O = \min_{S \in O} g^l(S) \times \eta(S)$. We also maintain $\underline{\gamma}^D$, the minimum $g^m(S) \times \eta(S)$ of search nodes that are not dominated but discarded due to the beam width up to the current layer. We assume $\underline{\gamma}^O = \infty$ if $O = \emptyset$ and $\underline{\gamma}^D = \infty$ if no state is discarded. In the actual implementation, we incrementally update $\underline{\gamma}^O$ and $\underline{\gamma}^D$ when a search node is generated or removed from the binary heap. Using these values, the dual bound on the optimal cost is computed as $\min\{\overline{\gamma}, \underline{\gamma}^O, \underline{\gamma}^D\}$, where $\overline{\gamma}$ is a primal bound (upper bound on the optimal cost for minimization).

6.2.2 Shared Beam Search (SBS)

We propose shared beam search (SBS), a parallel beam search algorithm using a shared hash table for duplicate detection. We use a concurrent hash table, DashMap,[3] in Rust. While a lock-free hash table such as LeapFrog[4] could be a choice, we select DashMap since it has better performance than lock-free hash tables when entries are frequently inserted in parallel.[5] DashMap has an architecture similar to the concurrent hash table used by Zhang and Hansen [448]; it is divided into multiple segments, and each segment has a lock. Multiple threads can access the same segment if all of them are reading. If one thread is mutating a segment, other threads cannot read or mutate that segment. Regarding this property, in our implementation, a thread first checks if a search node is dominated by another search node in the hash table by just reading it. If not, a thread acquires the lock of the corresponding segment for writing, checks dominance again since a new node might have been inserted before acquiring the lock, and then inserts the search node if not dominated.

We show pseudo-code of SBS in Algorithm 19. Similar to sequential beam search, SBS has a single open list O and maintains the best solution found so far ($\overline{\sigma}$) and a flag indicating the completeness of search (complete). The set G, which stores successor states, is divided into multiple subsets $G_1, ..., G_K$, corresponding to segments in the concurrent hash table. SBS checks base states (lines 6–9) before expanding states. Then, SBS expands states in the open list O and generates successor states (lines 11–22). These for loops (lines 6–9 and lines 11-22) are performed in parallel by a thread pool using Rayon,[6] a multi-threading library in Rust. Assignments of the threads to loop iterations are managed by Rayon. Lines 8–9 are performed in two phases: take the minimum of current_cost in parallel using Rayon and then update $\overline{\sigma}$ to the corresponding solution.

While the sequential implementation incrementally selects the best b states from the generated states, parallelizing this step is not straightforward. In our implementation of SBS, for the open list O, we use an array of search nodes instead of a binary heap. If $|O| > b$, search nodes are sorted in parallel by the f- and h-values, and the best b nodes are selected (line 26). For this step, we use the parallel sorting algorithm provided by Rayon. If the flag of a search node indicates that the state is not included in G, that search node is removed before sorting. The difference between incremental selection in sequential beam search and sorting in SBS may change the search behavior as a tie can be broken differently when multiple states have the same f- and h-values.

Correctness

In Section 4.2.7, we proved several properties for beam search: it terminates in finite time given a finite and acyclic DyPDL model (Theorem 16); the return value is a feasible solution if it is not NULL (Theorem 17); it proves optimality given a sufficiently large beam width (Theorem 18); the dual bound on the optimal path cost can be computed by taking the minimum $g^l(S) \times \eta(S)$ over states in the open list O and discarded states so far (Theorem 20). We prove the same properties for SBS, confirming that the features of SBS do not affect the assumptions used in the original proofs for sequential beam search.

[3] `https://crates.io/crates/dashmap`
[4] `https://docs.rs/crate/leapfrog`
[5] `https://github.com/robclu/conc-map-bench`
[6] `https://crates.io/crates/rayon`

Algorithm 19 SBS for minimization with a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$ with $\langle A, \times, \mathbf{1} \rangle$. An approximate dominance relation $\preceq_a$, a dual bound function η, a primal bound $\overline{\gamma}$, and a beam width b are given as input.

1: **if** $S^0 \not\models \mathcal{C}$ **then return** NULL, $\top$
2: $\overline{\sigma} \leftarrow$ NULL, complete $\leftarrow \top$ ▷ Initialize the solution.
3: $l \leftarrow 0$, $\sigma^0(S^0) \leftarrow \langle\rangle$, $g^l(S^0) \leftarrow \mathbf{1}$ ▷ Initialize the g-value.
4: $O \leftarrow \{S^0\}$ ▷ Initialize the open list
5: **while** $O \neq \emptyset$ and $\overline{\sigma} =$ NULL **do**
6: **for all** $S \in O$ with $\exists B \in \mathcal{B}, S \models C_B$ in parallel **do**
7: current_cost $\leftarrow g^l(S) \times \min_{B \in \mathcal{B}: S \models C_B} \mathsf{cost}_B(S)$ ▷ Compute the solution cost.
8: **if** current_cost $< \overline{\gamma}$ **then**
9: $\overline{\gamma} \leftarrow$ current_cost, $\overline{\sigma} \leftarrow \sigma^l(S)$ ▷ Update the best solution.
10: $G_i \leftarrow \emptyset$ for $i = 1, ..., K$ ▷ Initailize the sets of states.
11: **for all** $S \in O$ with $\nexists B \in \mathcal{B}, S \models C_B$ in parallel **do**
12: **for all** $\tau \in \mathcal{T}(S)$ with $S[\![\tau]\!] \models \mathcal{C}$ **do**
13: $g_{\text{current}} \leftarrow g^l(S) \times w_\tau(S)$ ▷ Compute the g-value.
14: **if** $g_{\text{current}} \times \eta(S[\![\tau]\!]) < \overline{\gamma}$ **then**
15: determine segment j for $S[\![\tau]\!]$.
16: lock G_j
17: **if** $\nexists S' \in G_j$ such that $S[\![\tau]\!] \preceq_a S'$ and $g_{\text{current}} \geq g^{l+1}(S')$ **then**
18: **if** $\exists S' \in G_j$ such that $S' \preceq_a S[\![\tau]\!]$ and $g_{\text{current}} \leq g^{l+1}(S')$ **then**
19: $G_j \leftarrow G_j \setminus \{S'\}$ ▷ Remove a dominated state.
20: $\sigma^{l+1}(S[\![\tau]\!]) \leftarrow \langle \sigma^l(S); \tau \rangle$, $g^{l+1}(S[\![\tau]\!]) \leftarrow g_{\text{current}}$ ▷ Update the g-value.
21: $G_j \leftarrow G_j \cup \{S[\![\tau]\!]\}$ ▷ Insert the successor state.
22: unlock G_j
23: $l \leftarrow l + 1$ ▷ Proceed to the next layer.
24: $O \leftarrow \{S \in \bigcup_{i=1}^K G_i \mid g^l(S) \times \eta(S) < \overline{\gamma}\}$ ▷ Prune states by the bound.
25: **if** $|O| > b$ **then**
26: $O \leftarrow$ the best b states in O, complete $\leftarrow \bot$ ▷ Keep the best b states.
27: **if** complete and $O \neq \emptyset$ **then**
28: complete $\leftarrow \bot$ ▷ A better solution may exist.
29: **return** $\overline{\sigma}$, complete ▷ Return the solution.

Theorem 31. *Given a finite, acyclic, and monoidal DyPDL model (Definitions 14 and 15 in Section 3.1.1 and Definition 24 in Section 4.1.3), SBS terminates in finite time.*

Proof. First, we clarify that a deadlock never happens: if a thread locks G_j in line 16, the thread unlocks it after a finite number of time steps in line 22. Therefore, as in the original proof for Theorem 16, it is sufficient to show that a successor state $S[\![\tau]\!]$ is inserted into G only if a new path to $S[\![\tau]\!]$ is found. The differences from sequential beam search are that G is divided into $G_1, ..., G_K$, the for loop in lines 11–22 is executed in parallel, and G_j is updated in parallel. Suppose that multiple paths to $S[\![\tau]\!]$ are found by different threads in the current layer. Since $S[\![\tau]\!]$ is assigned to a segment G_j according to the values of the non-resource variables, all threads access G_j. Only one thread can access G_j at a time due to the lock, so the threads check and potentially update G_j in some sequential order. Therefore, in the current layer, $S[\![\tau]\!]$ is inserted into G_j with the same g-value only once. If $S[\![\tau]\!]$ was included in G_j in a previous layer, as explained in the original proof, we have found a longer path to $S[\![\tau]\!]$ in the current layer. □

Theorem 32. *In line 10 of Algorithm 19, if $\overline{\sigma} \neq$ NULL, then $\overline{\sigma}$ is a solution for the model with $\overline{\gamma} = \mathsf{cost}_{\overline{\sigma}}(S^0)$.*

Proof. For a successor state $S[\![\tau]\!]$, $\sigma^{l+1}(S[\![\tau]\!])$ and $g^{l+1}(S[\![\tau]\!])$ are updated one by one in line 20 since G_j is locked. For the parent state S, $\sigma^l(S)$ and $g^l(S)$ are not modified by any thread in the current iteration. By mathematical induction in the same way as the proof of Theorem 12, after line 20, $S[\![\tau]\!]$ is reachable from the target state with $\sigma^l(S)$, and the path weight of $\sigma^l(S)$ is equal to $g^l(S)$. In lines 8–9, as explained above, SBS first identifies the best solution cost $\overline{\gamma}$ and then updates $\overline{\sigma}$ to the corresponding $\sigma^l(S)$. When SBS finishes the loop and reaches line 10, by the same argument in the proof of Theorem 12, we can prove that $\overline{\sigma}$ is a solution. □

Theorem 33. *Let $\langle A, \times, \mathbf{1}\rangle$ be a monoid where $A \subseteq \mathbb{Q} \cup \{-\infty, \infty\}$ and A is isotone (Definitions 21 and 22 in Section 4.1.2). Given a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C}\rangle$ with $\langle A, \times, \mathbf{1}\rangle$ and $\overline{\gamma} \in A$, if an optimal solution exists for the minimization problem with the model, and SBS returns $\overline{\sigma} \neq$ NULL and complete $= \top$, then $\overline{\sigma}$ is an optimal solution. If SBS returns $\overline{\sigma} =$ NULL and complete $= \top$, then there does not exist a solution whose cost is less than $\overline{\gamma}$.*

Proof. Similar to the proof of Theorem 18, since complete $= \top$, we assume that we never reach lines 26 and 28. It is sufficient to prove the claim of Lemma 4 in Section 4.1.4: when a solution with a cost $\hat{\gamma}$ exists and $\overline{\gamma} > \hat{\gamma}$, the open list contains a state $\hat{S}$ such that there exists an $\hat{S}$-solution $\hat{\sigma}$ with $\mathsf{cost}_{\langle \sigma^l(\hat{S}); \hat{\sigma}\rangle}(S^0) \leq \hat{\gamma}$. While the set of states in the next layer, G, is divided into $G_1, ..., G_K$ in SBS, all generated and not dominated states are inserted into one of them, and the open list is updated to $\{\bigcup_{i=1}^{K} G_i \mid g^l(S) \times \eta(S) \leq \overline{\gamma}\}$ in line 24. Thus, it does not make a difference in the argument of the proof of Theorem 18. □

Theorem 34. *Let $\langle A, \times, \mathbf{1}\rangle$ be a monoid where $A \subseteq \mathbb{Q} \cup \{-\infty, \infty\}$ and A is isotone. Given a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C}\rangle$ with $\langle A, \times, \mathbf{1}\rangle$ and $\overline{\gamma} \in A$, let D_m be the set of states discarded in layer $m \leq l-1$ by line 26 of Algorithm 19. If an optimal solution for the minimization problem with the model exists and has the cost γ^*, just after line 24,*

$$\min\left\{\overline{\gamma}, \min_{S \in O} g^l(S) \times \eta(S), \min_{m=1,...,l-1} \min_{S \in D_m} g^m(S) \times \eta(S)\right\} \leq \gamma^*$$

where we assume $\min_{S \in O} g^l(S) \times \eta(S) = \infty$ if $O = \emptyset$ and $\min_{S \in D_m} g^m(S) \times \eta(S) = \infty$ if $D_m = \emptyset$.

Proof. For sequential beam search, the proof of Theorem 20 uses mathematical induction to show that if $\gamma^* < \overline{\gamma}$, a state $\hat{S}$ is included in $O \cup \bigcup_{m=1}^{l-1} D_m$, and there exists an $\hat{S}$-solution $\hat{\sigma}$ such that $\langle \sigma^m(\hat{S}); \hat{\sigma}\rangle$ is an optimal solution where $m \in \{1, ..., l\}$. Similar to the proof of Theorem 33, this property is not changed in SBS since generated successor states are inserted into one of $G_1, ..., G_K$, and the open list is updated to the union of them in line 24. □

In practice, to compute the dual bound, we need to compute $\underline{\gamma}^O = \min_{S \in O} g^l(S) \times \eta(S)$. In sequential beam search, we incrementally update $\underline{\gamma}^O$ when a search node is generated or removed from the binary heap. Since SBS generates search nodes in parallel and stores them in the array in an arbitrary order decided by Rayon, computing $\underline{\gamma}^O$ requires additional effort. However, if the array has more than b states, and the dual bound function is used as a heuristic function, we can obtain $\underline{\gamma}^O$ for free since we sort the array to select the best b states according to the f-values, where

$f(S) = g^l(S) \times h(S) = g^l(S) \times \eta(S)$. Accordingly, to compute $\underline{\gamma}^D$, the minimum $g^l(S) \times \eta(S)$ of discarded states in the current layer can be obtained by taking the $b+1$-th f-value of the sorted array in each layer. Therefore, SBS computes the dual bound on the optimal cost only when it reaches line 26.

Comparison to Previous Work

Conceptually, SBS is similar to the parallel beam search algorithm proposed by Frohner et al. [148]: both of them use layer synchronization, a concurrent hash table for duplicate detection, and parallel sorting. In terms of implementation, there are several differences. While Frohner et al. used Julia and its Threads module, we use Rust and the Rayon library for SBS. Moreover, while Frohner et al. implemented a concurrent hash table by themselves, we use DashMap, a library in Rust. For parallel sorting, Frohner et al. implemented a parallel counting sort algorithm, but we use the algorithm provided by Rayon in SBS. This design choice in SBS is because the parallel counting sort algorithm by Frohner et al. is tied to their randomization mechanism, which is not used in SBS. In their parallel counting sort algorithm, each thread counts the numbers of states in buckets associated with different ranges of f-values. They use floating point numbers for f-values, obtained by adding Gaussian noise to $g(S) + h(S)$, and each bucket is associated with range $[i, i+1)$ where i is an integer. In SBS, we do not use such a randomization mechanism and do not assume f-values to be continuous numbers.

6.2.3 Hash-Distributed Beam Search (HDBS)

As an alternative approach to SBS, we propose hash-distributed beam search (HDBS), which uses hash-based work distribution with layer synchronization. We propose two variants, HDBS1 and HBDS2, with different layer synchronization mechanisms. Before explaining these variants, we describe their common features. As explained in Section 6.1.5, in hash-based work distribution, each process has its local open list and a hash table for duplicate detection. Using asynchronous message passing, a process sends a generated state to its owner process that is uniquely determined by the state's hash value. In each layer, one process keeps its best b/k states, where k is the number of processes. This mechanism does not guarantee that the best b states in that layer are kept; if a process owns only states having large f-values, and another process owns only states having small f-values, HDBS may keep states that would have been discarded by sequential beam search and discard states that would have been kept.

HDBS1

HDBS1 takes a straightforward approach to layer synchronization: all processes are synchronized after expanding all states in a layer by broadcast. We present pseudo-code for HDBS1 in Algorithm 20. Each process i has an open list O, a set of successor states G, and a first-in-first-out message queue Q_i. While all variables are specific to process i, we use a subscript i for the message queue Q_i and data sent to process 1: solution_found$_i$, empty$_i$, complete$_i$, $\overline{\gamma}_i$, and $\underline{\gamma}_i$. When a successor state is generated, its owner is computed based on the hash value and it is sent to its owner (lines 20 and 21). As explained in Section 6.1.5, $\textsc{owner}(S) = (\textsc{hash}(S) \mod k) + 1$ where $\textsc{hash}(S)$ is the hash value of state S. The hash value is based on the values of non-resource variables so that dominance

Algorithm 20 HDBS1 for minimization with a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$ with $\langle A, \times, \mathbf{1} \rangle$. An approximate dominance relation $\preceq_a$, a dual bound function η, a primal bound $\overline{\gamma}$, and a beam width b are given as input.

1: **if** $S^0 \not\models \mathcal{C}$ **then return** NULL, ∞, $\top$
2: $Q_i \leftarrow []$ for $i = 1, ..., k$ ▷ Initialize the message queues.
3: **for** $i = 1, ..., k$ in parallel **do**
4: $l \leftarrow 0$, $\overline{\sigma} \leftarrow$ NULL, $O \leftarrow \emptyset$, complete$_i \leftarrow \top$, $\overline{\gamma}_i = \overline{\gamma}$, $\underline{\gamma}^D \leftarrow \infty$ ▷ Data for process i.
5: **if** $i =$ OWNER(S^0) **then** $\sigma^l(S^0) \leftarrow \langle\rangle$, $g^l(S^0) \leftarrow \mathbf{1}$, $O \leftarrow \{S^0\}$ ▷ Initalize the open list.
6: **loop**
7: $G \leftarrow \emptyset$, $c \leftarrow 0$, sent_all $\leftarrow \bot$ ▷ Data in the current layer.
8: **while** $c < k$ **do**
9: RECV_STATE$(Q_i, l, c, G, \overline{\gamma}_i)$ ▷ Execute Algorithm 21.
10: **if** $\exists S \in O$ **then**
11: $O \leftarrow O \setminus \{S\}$ ▷ Remove a state.
12: **if** $\exists B \in \mathcal{B}, S \models C_B$ **then**
13: current_cost $\leftarrow g^l(S) \times \min_{B \in \mathcal{B}: S \models C_B} \mathsf{cost}_B(S)$ ▷ Compute the solution cost.
14: **if** current_cost $< \overline{\gamma}_i$ **then**
15: $\overline{\gamma}_i \leftarrow$ current_cost, $\overline{\sigma} \leftarrow \sigma^l(S)$ ▷ Update the best solution.
16: **else**
17: **for all** $\tau \in \mathcal{T}(S)$ with $S[\![\tau]\!] \models \mathcal{C}$ **do**
18: $g_{\text{current}} \leftarrow g^l(S) \times w_\tau(S)$ ▷ Compute the g-value.
19: **if** $g_{\text{current}} \times \eta(S[\![\tau]\!]) < \overline{\gamma}_i$ **then**
20: $j \leftarrow$ OWNER$(S[\![\tau]\!])$ ▷ Determine the owner.
21: SEND$(Q_j, (S[\![\tau]\!], \langle \sigma^l(S); \tau \rangle, g_{\text{current}}, \eta(S[\![\tau]\!])))$ ▷ Send a message.
22: **else if** not sent_all **then**
23: sent_all $\leftarrow \top$, SEND(Q_j, NULL) for $j = 1, ..., k$ ▷ Notify that all states were sent.
24: $l \leftarrow l + 1$ ▷ Proceed to the next layer.
25: $O \leftarrow \{S \in G \mid g^l(S) \times \eta(S) < \overline{\gamma}_i\}$, $\underline{\gamma}_i \leftarrow \min\left\{\overline{\gamma}_i, \underline{\gamma}^D, \min_{S \in O} g^l(S) \times \eta(S)\right\}$
26: solution_found$_i \leftarrow \overline{\sigma} \neq$ NULL, empty$_i \leftarrow O = \emptyset$
27: send solution_found$_i$, empty$_i$, complete$_i$, $\overline{\gamma}_i$, and $\underline{\gamma}_i$ to process 1
28: **if** $i = 1$ **then** AGGREGATE$(\underline{\gamma})$ ▷ Execute Algorithm 22.
29: receive α, is_optimal, and $\underline{\gamma}$ from process 1 ▷ Receive a broadcasted message.
30: **if** $i = \alpha$ **then**
31: **return** $\overline{\sigma}$, is_optimal, $\underline{\gamma}$ ▷ Return the solution.
32: **else if** $\alpha \neq$ NULL **then**
33: break ▷ Terminate.
34: **if** $|O| > b/k$ **then**
35: $O \leftarrow$ the best b/k states in O, complete$_i \leftarrow \bot$ ▷ Keep the best b/k states.
36: $\underline{\gamma}^D \leftarrow \min\left\{\underline{\gamma}^D, \min_{S \in \text{discarded states from } O} g^l(S) \times \eta(S)\right\}$ ▷ Check discarded states.

can be detected in each process. The send operation is asynchronous, i.e., the process does not wait until the message is received. Each process checks its message queue in line 9 using Algorithm 21. This operation is also asynchronous; if the message queue is empty, the process does not wait for messages. If the message queue contains a message, the process receives a state and inserts it into G while pruning dominated states (line 6–12 of Algorithm 21). In Algorithm 20, for simplicity, a process sends a state to its owner by message passing even if the process itself is the owner. In practice, such a state is immediately processed without using message passing.

After expanding all states in the open list, each process notifies other processes that it has sent

Algorithm 21 Receive states from a message queue. A message queue Q_i, the index of the current layer l, a counter c, a set of states G, and a primal bound $\overline{\gamma}_i$ are given as input.

1: **while** Q_i has an incoming message **do**
2: message $\leftarrow$ RECV(Q) ▷ Receive a message.
3: **if** message = NULL **then**
4: $c \leftarrow c + 1$ ▷ Received all states from a process.
5: **else**
6: $(S, \sigma^{\text{current}}, g_{\text{current}}, \eta(S)) \leftarrow$ message ▷ Received a state.
7: **if** $g_{\text{current}} \times \eta(S) < \overline{\gamma}_i$ **then**
8: **if** $\nexists S' \in G$ such that $S \preceq_a S'$ and $g_{\text{current}} \geq g^{l+1}(S')$ **then**
9: **if** $\exists S' \in G$ such that $S' \preceq_a S$ and $g_{\text{current}} \leq g^{l+1}(S')$ **then**
10: $G \leftarrow G \setminus \{S'\}$ ▷ Remove a dominated state.
11: $\sigma^{l+1}(S) \leftarrow \sigma^{\text{current}}$, $g^{l+1}(S) \leftarrow g_{\text{current}}$ ▷ Update the g-value.
12: $G \leftarrow G \cup \{S\}$ ▷ Insert the state.

Algorithm 22 Aggregate information of the current layer. The current dual bound on the optimal cost $\underline{\gamma}$ is given as input.

1: receive solution_found$_j$, empty$_j$, complete$_j$, $\overline{\gamma}_j$, and $\underline{\gamma}_j$ from $j = 1, ..., k$
2: solution_found $\leftarrow \bigvee_{j=1}^{k}$ solution_found$_j$, empty $\leftarrow \bigwedge_{j=1}^{k}$ empty$_j$, complete $\leftarrow \bigwedge_{j=1}^{k}$ complete$_j$
3: is_optimal $\leftarrow$ empty $\wedge$ complete ▷ Check the optimality.
4: $\underline{\gamma} \leftarrow \max\left\{\underline{\gamma}, \min_{j=1,...,k} \underline{\gamma}_j\right\}$ ▷ Update the dual bound on the optimal cost.
5: $\alpha \leftarrow$ NULL ▷ Index of the process that should return the result.
6: **if** solution_found **then**
7: let $\alpha \in \arg\min_{j=1,...,k} \overline{\gamma}_j$ ▷ Process α found the best solution.
8: is_optimal $\leftarrow$ is_optimal $\vee\ \underline{\gamma} = \overline{\gamma}_\alpha$ ▷ Check the optimality using the bounds.
9: **else if** empty **then**
10: $\alpha \leftarrow 1$ ▷ There are no states to search, so let process 1 return NULL.
11: broadcast α, is_optimal, $\underline{\gamma}$

all states by sending a special message NULL (line 23). Here, the flag 'sent_all' is used to send the special message only once for each layer. Using c, each process maintains the number of processes from which it has received all successor states, which is incremented in line 4 of Algorithm 21. When c becomes the number of processes, k, a process has received all states in the next layer from all processes. Since the process also sends the special message to itself (again, such a message is immediately processed without message passing in practice), it also means that the process has expanded all states in the current layer. Therefore, the process exits the while loop when $c = k$ (line 8). Then, the process sends the status of the current layer to process 1 in line 27: solution_found$_i$ is a flag indicating if a solution is found (line 26); empty$_i$ is a flag indicating if the new layer is empty (line 26); complete$_i$ is a flag indicating if any state has been discarded due to the beam width so far (lines 4 and 35); $\overline{\gamma}_i$ is a primal bound, which is initialized with an input value $\overline{\gamma}$ (line 4) and updated when a better solution is found (line 15); $\underline{\gamma}_i$ is a candidate for the dual bound on the optimal cost, which we explain later. Process 1 aggregates the flags over all processes and produces solution_found, empty, and complete using Algorithm 22. If solution_found = $\top$, then process α, which has found the best solution, is identified in line 7. If empty = complete = $\top$, all states that are not pruned by dominance or bounds have been searched, so the solution is optimal, and the flag is_optimal becomes $\top$ (line 3). Process 1 broadcasts α and is_optimal, process α returns the solution with the

optimality flag (line 31 in Algorithm 20), and other processes exit the loop and terminate (line 33).

If solution_found = $\bot$ and empty = $\top$ is true, HDBS1 terminates without a solution. Algorithm 22 assigns 1 to α (line 10) and lets process 1 return NULL. If is_optimal = $\top$ in addition, no state has been discarded, indicating that there is no solution whose cost is better than the given primal bound. If solution_found = empty = $\bot$, then α remains NULL, and all processes search the next layer. Since broadcast is used, all processes are synchronized in line 29, which achieves layer synchronization.

While the computation of the dual bound using Theorem 20 in Section 4.2.7 is implicit in sequential beam search (Algorithm 10) and SBS (Algorithm 19), we explicitly describe it in Algorithm 20 since it requires message passing. Similar to the sequential beam search implementation described in Section 6.2.1, process i maintains $\underline{\gamma}^D$, the minimum $g^m(S) \times \eta(S)$ of discarded states due to the beam width (line 35). After process i expands all states in the current layer, in line 25, O becomes the set of states in the new layer. Following Theorem 20, process i computes $\underline{\gamma}_i = \min\left\{\overline{\gamma}_i, \underline{\gamma}^D, \min_{S \in O} g^l(S) \times \eta(S)\right\}$ using only local information. This value is not guaranteed to be the dual bound on the optimal cost since other processes may have states with smaller $g^l(S) \times \eta(S)$. Therefore, $\underline{\gamma}_i$ is sent to process 1, which takes the minimum of all processes, updates $\underline{\gamma}$ if a better bound is found (line 4 in Algorithm 22), and broadcasts it. If $\underline{\gamma} = \overline{\gamma}_\alpha$, an optimal solution is found, so is_optimal becomes $\top$ (line 8).

Correctness We prove the theoretical properties of HDBS1. We make the following assumption for message passing.

Assumption 1. A message arrives in finite time. When multiple messages are sent from one process to another, those messages arrive in the order of sending.

We prove that HDBS1 searches layer by layer as in sequential beam search.

Definition 27. In Algorithm 20, a process is in layer m when $l = m$. A message sent is in layer m if the process is in layer m when it sends the message.

Lemma 5. *In Algorithm 20, when all processes are synchronized in line 29, they are in the same layer.*

Proof. When a process reaches line 29 for the first time, $l = 1$ since l is initialized to be 0 in line 4 and is incremented by 1 in line 24. When a process reaches line 29 with $l = 1$, it waits until all other processes reach there with $l = 1$. Thus, the lemma holds for $m = 1$. Suppose that the lemma holds up to $l = m$. When all processes are synchronized in line 29 with $l = m$, if $\alpha \neq$ NULL, all processes terminate. Otherwise, all processes proceed to the next iteration of the loop. When each process reaches line 29 for the next time, due to line 24, $l = m + 1$, and thus the lemma holds for $l = m + 1$. Therefore, by mathematical induction, the lemma holds. □

Lemma 6. *In Algorithm 20, the following two conditions hold:*

- *During the while loop in lines 8–23, a process receives only messages in layer l.*
- *When a process exits the while loop, it has received all messages sent to it in layer l.*

Proof. When process i enters the while loop with $l = m$, we have the following assumption: no message in layer $l \leq m-1$ was sent to i, or all messages in layer $l \leq m-1$ sent to process i were received. By the assumption, all messages received during the while loop are in layer $l \geq m$. To send a message in layer $l > m$, a process needs to pass line 29 with $l = m+1$, which requires all processes to reach line 29 with $l = m+1$ by Lemma 5. However, since process i cannot reach line 29 with $l = m+1$ before exiting the while loop, process i in layer m cannot receive a message in a layer $l > m$. Therefore, the first condition holds for $l = m$.

When process i exits the while loop, i received k special messages, NULL, in layer m sent by line 23. To reach line 23, sent_all must be $\bot$, which becomes $\top$ after reaching line 23. Thus, in layer m, each process sent a special message only once, so process i must have received one special message from each process. When a process reaches line 23, $O = \emptyset$, so it has sent all successor states. By Assumption 1, process i needs to receive all successor states before receiving the special messages. Therefore, the second condition holds for $l = m$.

We proved that our assumption, no message in layer $l \leq m-1$ was sent to i, or all messages in layer $l \leq m-1$ sent to process i were received, implies the lemma for $l = m$. This assumption is valid for $m = 0$ since $l \leq -1$ never holds, and thus no message is sent with $l \leq -1$. When the lemma holds for $l = m$, the assumption holds for $l = m+1$ since process i has received all messages in layer m. By mathematical induction, the lemma holds for $l \geq 0$. □

Theorem 35. *In Algorithm 20, in lines 26–36 and lines 7–23, each state $S \in O$ satisfies $|\sigma^l(S)| = l$.*

Proof. We never reach lines 26–36 in Algorithm 20 with $l = 0$ since l is initialized to be 0 in line 2 and is incremented by 1 in line 24. In lines 7–23, if $l = 0$, then $O = \{S^0\}$ with $|\sigma^0(S^0)| = 0$ for process $\text{OWNER}(S^0)$ and $O = \emptyset$ for other processes. Since no state is added to O in lines 7–23, the theorem holds for $l = 0$. Suppose that the theorem holds up to $l = m$. In lines 7–23 with $l = m$, by Lemma 6, a process only receives messages in layer m. Each of the states received in line 21 is a successor state $S[\![\tau]\!]$ of $S \in O$ generated in the sender process, where $|\langle \sigma^m(S); \tau \rangle| = m+1$ since $|\sigma^m(S)| = m$ by the assumption. When receiving this state, $|\sigma^{\text{current}}| = m+1$ in line 6 of Algorithm 21, and thus $|\sigma^{m+1}(S[\![\tau]\!])| = m+1$ when it is inserted into G in line 12. After exiting the while loop, l becomes $m+1$, and O is updated to a subset of G in lines 25 of Algorithm 20. Each state $S \in O$ satisfies $|\sigma^{m+1}(S)| = m+1$. If all processes terminate in lines 30–33, we are done. Otherwise, all processes go to the next iteration of the loop. Since line 35 only removes states from O, the condition continues to hold until line 24, where l is incremented. Thus, the theorem holds for $l = m+1$. By mathematical induction, the theorem holds for $l \geq 0$. □

Now, we prove that HDBS1 terminates in finite time.

Lemma 7. *Given a finitely defined DyPDL model (Definition 11 in Section 3.1.1), in Algorithm 20, one iteration of the while loop in lines 8–23 finishes in finite time.*

Proof. By lines 34–35, when a process enters the while loop, O contains at most b/k states, so the process expands a finite number of states during the while loop. Since the model is finite, the number of transitions is finite, and each process generates a finite number of successor states and sends a finite number of messages in layer l. By Lemma 6, each process in layer l receives only messages in layer l, which are finitely many, so it executes finitely many loop iterations in Algorithm 21, and G always contains at most a finite number of states. Since G contains at most a finite number of states,

dominance detection in Algorithm 21 is done in finite time. Since the model is finitely defined, each operation with the model, e.g., checking base cases and generating successor states, is done in finite time. Sending a message in line 21 or 23 does not wait until the message is received. Thus, each iteration of the while loop is done in finite time. □

Lemma 8. *Algorithm 20 satisfies the following three conditions:*

- *When the first process enters the while loop in lines 8–23 with $l = m$, all processes will enter the while loop with $l = m$ in finite time.*
- *Once a process enters the while loop with $l = m$, it will expand all states in O, generate all successor states, send all messages in layer m, and receive all messages in layer m in finite time.*
- *When the last process enters the while loop with $l = m$, all processes are in the while loop with $l = m$ and will exit in finite time.*

Proof. Assume that the first condition holds up to $m = n$. For $m = 0$, this assumption is valid since once HDBS1 starts, all processes enter the while loop with $l = 0$ in finite time. When a process i enters the while loop with $l = n$, since all processes have not received the special message NULL in layer n from process i, by Lemma 6, all processes have not exited the while loop with $l = n$. Thus, when the last process enters the while loop with $l = n$, all processes are in the while loop with $l = n$.

When a process enters the while loop with $l = n$, O contains at most b/k states due to lines 34–35. Since each iteration of the while loop removes one state from O and takes finite time by Lemma 7, O will become empty in finite time. After expanding all states, O becomes empty, and the process sends special messages in layer n to all processes in line 23. Therefore, each process sends all messages in layer n in finite time after entering the while loop with $l = n$.

If a process j has already entered the while loop when i enters, j may have already sent some messages in layer n to process i, but they have not been received yet by Lemma 6. By Assumption 1, all messages arrive in finite time. By Lemma 7, each loop iteration is done in finite time, so process i reaches line 9 in finite time after a message arrives in the message queue Q_i. Therefore, process i receives all messages in layer n, including the special messages sent in line 23, from all k processes in finite time and exits the while loop. Thus, with the previous paragraphs, the second and third conditions are proved.

Once process i exits the while loop, it reaches line 29 in finite time and waits for other processes. Since i exited the while loop, all processes have at least entered the while loop with $l = n$, and all processes have already exited or will exit the while loop in finite time. Thus, all processes will be synchronized again in line 29 with $l = n + 1$ in finite time. Due to the broadcast in line 29, α is the same in all processes in line 30. If $\alpha \neq$ NULL, all processes terminate. Otherwise, all processes proceed to the next iteration of the loop and will enter the while loop with $l = n + 1$ in finite time. Therefore, the first condition holds for $m = n + 1$. By mathematical induction, the lemma holds for $m \geq 0$. □

Theorem 36. *Given a finite, acyclic, and monoidal DyPDL model, HDBS1 terminates in finite time.*

Proof. By Lemma 8, after all processes are synchronized in line 29 with $l = m$, all processes will terminate or be synchronized in line 29 with $l = m + 1$ in finite time. By Theorem 35, after line 25, O contains only states reached with l transitions. Since the model is finite and acyclic, all paths in the state transition graph have a finite number of transitions. Therefore, when l becomes a sufficiently large finite number, O becomes empty in all processes. When O is empty in all processes, by Algorithm 22, α becomes 1, and all processes terminate in lines 30–33. Therefore, HDBS1 terminates in finite time. □

We also confirm that HDBS1 is an anytime algorithm.

Theorem 37. *In line 15 of Algorithm 20, if $\overline{\sigma} \neq$ NULL, then $\overline{\sigma}$ is a solution for the model with $\overline{\gamma}_i = \mathsf{cost}_{\overline{\sigma}}(S^0)$.*

Proof. The proof is the same as the proof of Theorems 12 and 17 since parallelization does not change how σ^l and g^l are maintained: for each successor state $S[\![\tau]\!]$ of S, we set $\sigma^{l+1}(S[\![\tau]\!]) = \langle \sigma^l(S); \tau \rangle$ and $g^{l+1}(S[\![\tau]\!]) = g^l(S) \times w_\tau(S)$. These values are sent with the successor state by message passing. □

In Algorithm 20, we explicitly maintain $\underline{\gamma}$, the dual bound on the optimal solution cost, and claim is_optimal $= \top$ if $\underline{\gamma}$ coincides with the primal bound. Thus, before proving the optimality of HDBS1, we confirm that $\underline{\gamma}$ is indeed a valid dual bound. First, we prove a lemma similar to Lemma 4.

Lemma 9. *In Algorithm 20, suppose that a solution exists for the DyPDL model, and let $\hat{\gamma}$ be its cost. Just after line 29, there exists a process i where at least one of the following conditions is satisfied:*

- *complete$_i = \bot$.*
- *$\overline{\gamma}_i \leq \hat{\gamma}$.*
- *O contains a state $\hat{S}$ such that an $\hat{S}$-solution $\hat{\sigma}$ exists, and $\langle \sigma^l(\hat{S}); \hat{\sigma} \rangle$ is a solution for the model with $\mathsf{cost}_{\langle \sigma^l(\hat{S}); \hat{\sigma} \rangle}(S^0) \leq \hat{\gamma}$.*

Proof. Assume that there exists process i which satisfies the third condition when it reaches line 7 with $l = m$. This assumption is valid for $l = 0$ since $S^0 \in O$ in process OWNER(S^0). Consider that all processes have exited the while loop with $l = m$, updated l to $m + 1$, and passed line 29. If complete$_i = \bot$ in some process, the lemma continues to hold in subsequent iterations. Otherwise, no state has been discarded in line 35 so far in all processes. In layer m, by Lemma 8, all processes expanded all states in O and sent all generated successor states before exiting the while loop in line 8–23. In addition, all processes received all successor states before exiting the while loop. Thus, no successor state is lost. If $\hat{S}$ does not have successor states, it must be a base state, and process i found a solution whose cost is $\overline{\gamma}_i \leq \hat{\gamma}$, which is the second condition. Otherwise, since received successor states are handled in the same way as sequential beam search, by a similar argument to the proof of Theorem 18, there exists a process i that satisfies the third condition in the lemma. Therefore, the lemma holds for $l = m + 1$. If all processes terminate in lines 30–33, or complete$_i$ becomes $\bot$ after lines 34-36, we are done. Otherwise, no process has found a solution, O is not changed after line 29, and all processes proceed to line 7. Thus, there exists process i which satisfies the third condition of the lemma when it reaches line 7 with $l = m + 1$, and the lemma holds for $l = m + 2$. By mathematical induction, the lemma holds for $l \geq 1$. □

Theorem 38. *Let $\langle A, \times, \mathbf{1}\rangle$ be a monoid where $A \subseteq \mathbb{Q} \cup \{-\infty, \infty\}$ and A is isotone. Given a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C}\rangle$ with $\langle A, \times, \mathbf{1}\rangle$ and $\overline{\gamma} \in A$, if an optimal solution for the minimization problem with the model exists and has the cost γ^*, just after line 29 of Algorithm 20, $\underline{\gamma} \leq \gamma^*$.*

Proof. If $\overline{\gamma} \leq \gamma^*$, it trivially holds since $\underline{\gamma} \leq \underline{\gamma}_i \leq \overline{\gamma}_i \leq \overline{\gamma}$. We assume that $\gamma^* < \overline{\gamma}$. Suppose that no state has been discarded in line 35 so far in all processes, and so $\underline{\gamma}^D = \infty$ and $\text{complete}_i = \top$ in all processes. This assumption is valid when each process reaches line 25 for the first time. By Lemma 9, in some process i, $\overline{\gamma}_i \leq \gamma^*$, or O contains a state $\hat{S}$ such that an $\hat{S}$-solution $\hat{\sigma}$ exists, and $\langle \sigma^l(\hat{S}); \hat{\sigma}\rangle$ is a solution for the model with the cost γ^*. In the first case, the theorem continues to hold in subsequent iterations. In the second case, by a similar argument to the proof of Theorem 20,

$$\underline{\gamma}_i = \min\left\{\overline{\gamma}_i, \infty, \min_{S \in O} g^l(S) \times \eta(S)\right\} \leq g^l(\hat{S}) \times \eta(\hat{S}) \leq \gamma^*.$$

If process i with $\hat{S} \in O$ does not discard $\hat{S}$ in line 35, we can repeat the same argument in the next iteration. Otherwise, after line 36, $\underline{\gamma}_D \leq g^l(\hat{S}) \times \eta(\hat{S})$. Therefore, in the next iteration,

$$\underline{\gamma}_i = \min\left\{\overline{\gamma}_i, \underline{\gamma}_D, \min_{S \in O} g^l(S) \times \eta(S)\right\} \leq \underline{\gamma}_D \leq \gamma^*.$$

The above inequality will hold in subsequent iterations since $\underline{\gamma}_D$ never increases. □

Theorem 39. *Let $\langle A, \times, \mathbf{1}\rangle$ be a monoid where $A \subseteq \mathbb{Q} \cup \{-\infty, \infty\}$ and A is isotone. Given a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C}\rangle$ with $\langle A, \times, \mathbf{1}\rangle$ and $\overline{\gamma} \in A$, if an optimal solution exists for the minimization problem with the model, and HDBS1 returns $\overline{\sigma} \neq$ NULL and is_optimal $= \top$, then $\overline{\sigma}$ is an optimal solution. If HDBS1 returns $\overline{\sigma} =$ NULL and is_optimal $= \top$, then there does not exist a solution whose cost is less than $\overline{\gamma}$.*

Proof. We focus on cases where is_optimal $= \top$ is returned by HDBS1. In such cases, $\alpha \neq$ NULL and is_optimal $= \top$ just after line 29 of Algorithm 20. After executing Algorithm 22, at least one of the following conditions held: empty $=$ complete $= \top$ or $\underline{\gamma} = \overline{\gamma}_\alpha$.

Suppose that empty $=$ complete $= \top$. Then, $\text{complete}_i = \top$ and $O = \emptyset$ in all processes. If there exists a solution whose cost is $\hat{\gamma} < \overline{\gamma}$, by Lemma 9, $\overline{\gamma}_i \leq \hat{\gamma}$ in some process i. Since $\overline{\gamma}_i \leq \hat{\gamma} < \overline{\gamma}$, process i should have reached line 15, and $\text{solution\_found}_i = \top$. Due to line 7 of Algorithm 22, $\overline{\gamma}_\alpha \leq \overline{\gamma}_i \leq \hat{\gamma} < \overline{\gamma}$, and process α should have updated $\overline{\sigma}$ to a solution with the cost $\overline{\gamma}_\alpha$ by Theorem 37. Therefore, when empty $=$ complete $= \top$, if a solution whose cost is less than $\overline{\gamma}$ exists, HDBS1 returns a solution $\overline{\sigma} \neq$ NULL whose cost is at most $\hat{\gamma} < \overline{\gamma}$. If an optimal solution exists, HDBS1 returns an optimal solution. By contraposition, when empty $=$ complete $= \top$, if HDBS1 returns $\overline{\sigma} =$ NULL, there is no solution whose cost is less than $\overline{\gamma}$.

Suppose that $\underline{\gamma} = \overline{\gamma}_\alpha$. If there exists an optimal solution with the cost γ^* and $\overline{\sigma} \neq$ NULL, by Theorem 39, $\overline{\gamma}_\alpha = \underline{\gamma} \leq \gamma^*$. Since $\overline{\sigma} \neq$ NULL, process α should have reached line 15, and $\overline{\sigma}$ is a solution with cost $\overline{\gamma}_\alpha = \gamma^*$. If $\overline{\sigma} =$ NULL, process α has never reached line 15, and $\overline{\gamma}_\alpha = \overline{\gamma}$. In all processes, due to line 25,

$$\min_{S \in O} g^l(S) \times \eta(S) \geq \underline{\gamma} = \overline{\gamma}_\alpha = \overline{\gamma}.$$

Therefore, any solution extending $\sigma^l(S)$ for $S \in O$ has a higher cost than $\overline{\gamma}$. In all processes, due to

line 36, for each discarded state S in layer $m \in \{1, ..., l-1\}$, $g^m(S) \times \eta(S) \geq \underline{\gamma} = \overline{\gamma}$. Therefore, any solution extending $\sigma^m(S)$ for a discarded state S in layer m has a higher cost than $\overline{\gamma}$. Thus, there is no solution with a smaller cost than $\overline{\gamma}$. □

HDBS2

In HDBS1, each process needs to wait until all processes receive all states in the next layer. However, once a process i has expanded all states in layer m and received all generated states in layer $m+1$ from other processes, no more states will be added in layer $m+1$ for process i. Process i can expand states in layer $m+1$ as long as it does not send generated states in layer $m+2$ to another process j that is still waiting for states in layer $m+1$. For such process j, process i stores generated states in a buffer. Process j notifies process i once it proceeds to layer $m+1$, and then process i sends the buffered states to process j. We call this approach HDBS2.

We show pseudo-code for HDBS2 in Algorithm 23. Similar to Algorithm 20, while all variables are specific to each process, we use subscript i for message queues Q_i and R_i. In addition, we differentiate $\overline{\gamma}_i$, the primal bound in each process, from $\overline{\gamma}$, the primal bound given as input.

After expanding all states in the current layer and receiving all states in the next layer, process i notifies each process j that it has proceeded to the next layer by asynchronously sending a message via message queue R_j (line 43). As a receiver, process i maintains J, the set of processes from which i received the notification via R_i. Process i can immediately send states to process j if $j \in J$ (line 29). Otherwise, it stores states in a buffer P_j (line 31) and sends all of them once receiving the notification from j (line 15). For the first layer, HDBS2 sends dummy notifications (line 6).

The notification also works as termination detection as it carries the information used by Algorithm 22 in HDBS1: a flag indicating if a solution is found ($\overline{\sigma} \neq$ NULL), a flag indicating if the open list is empty ($O = \emptyset$), a flag indicating if any state is discarded so far (complete), the cost of the best solution in the previous layer found by that process ($\overline{\gamma}^{l-1}$), and a candidate for the dual bound on the optimal cost ($\underline{\gamma}^l$). Unlike HDBS1, where process 1 aggregates the information and broadcasts the result, in HDBS2, each process aggregates the information and decides whether to terminate using Algorithm 24. In this algorithm, α is the index of the process that should return the solution, and $\overline{\gamma}^{l-1}$ is the cost of the best solution in the previous layer. If solution_found$_j = \top$, process j has found a solution whose cost is $\overline{\gamma}_j^{l-1}$ in the previous layer. If $\overline{\gamma}_j^{l-1}$ is better, $\overline{\gamma}^{l-1}$ is updated, and α becomes j. If $\overline{\gamma}_j^{l-1} = \overline{\gamma}^{l-1}$, α is updated if $j < \alpha$ (line 6). In this way, α eventually becomes the same value in all processes regardless of the orders in which messages arrive, and all processes agree on which process returns the solution. Similarly to Algorithm 22, when no solution is found and empty $= \top$ holds after receiving all messages, process 1 returns NULL (line 34 in Algorithm 23). The primal bound $\overline{\gamma}_i$ is updated whenever a better solution is found (line 12) while the dual bound $\underline{\gamma}$ is updated only after receiving the information from all processes (line 13).

In line 32, before sending the special message NULL, process i makes sure that it has received the messages via the message queue R_i from all processes ($|J| = k$). Therefore, at this point, process i does not have any message in buffers $P_1, ..., P_k$.

Correctness We verify the completeness and optimality of HDBS2, proving similar theoretical claims to those for HDBS1. Our proofs are similar to the original ones, but there are some changes regarding the different layer synchronization mechanisms. First, we prove a lemma for HDBS2 that

Algorithm 23 HDBS2 for minimization with a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$ with $\langle A, \times, \mathbf{1} \rangle$. An approximate dominance relation $\preceq_a$, a dual bound function η, a primal bound $\overline{\gamma}$, and a beam width b are given as input.

if $S^0 \not\models \mathcal{C}$ **then return** NULL, $\top$
$Q_i, R_i \leftarrow []$ for $i = 1, ..., k$ ▷ Initialize the message queues.
for $i = 1, ..., k$ in parallel **do**
  $l \leftarrow 0$, $\overline{\sigma} \leftarrow$ NULL, $O \leftarrow \emptyset$, complete $\leftarrow \top$, $\overline{\gamma}_i \leftarrow \overline{\gamma}$, $\underline{\gamma} \leftarrow \eta(S^0)$, $\underline{\gamma}^D \leftarrow \infty$ ▷ Data for i.
  if OWNER$(S^0) = i$ **then** $\sigma^l(S^0) \leftarrow \langle\rangle$, $g^l(S^0) \leftarrow \mathbf{1}$, $O \leftarrow \{S^0\}$ ▷ Initialize the open list.
  SEND$(R_j, (i, \bot, O = \emptyset, \text{complete}, \overline{\gamma}_i, \underline{\gamma}))$ for $i = 1, ..., k$ ▷ Send the initial information.
  loop
    $G \leftarrow \emptyset$, $c \leftarrow 0$, sent_all $\leftarrow \bot$, $P_j \leftarrow []$ for $j = 1, ..., k$ ▷ Data in the current layer.
    $J \leftarrow \emptyset$, $\alpha \leftarrow$ NULL, empty $\leftarrow \bot$, $\overline{\gamma}^{l-1}, \overline{\gamma}^l, \underline{\gamma}^l \leftarrow \infty$ ▷ Initialize the information.
    while $c < k$ **do**
      RECV_INFO$(R_i, J, \alpha, \text{empty}, \text{complete}, \overline{\gamma}^{l-1}, \underline{\gamma}^l)$ ▷ Execute Algorithm 24.
      $\overline{\gamma}_i = \min\{\overline{\gamma}_i, \overline{\gamma}^{l-1}\}$ ▷ Update the primal bound.
      if $|J| = k$ **then** $\underline{\gamma} = \max\{\underline{\gamma}, \underline{\gamma}^l\}$ ▷ Update the dual bound.
      for all $j \in J$ such that P_j contains messages **do**
        SEND$(Q_j$, message) for message in P_j, $P_j \leftarrow []$ ▷ Send buffered messages.
      RECV_STATE$(Q_i, l, G, c, \overline{\gamma}_i)$ ▷ Execute Algorithm 21.
      if $\exists S \in O$ **then**
        $O \leftarrow O \setminus \{S\}$ ▷ Remove a state.
        if $\exists B \in \mathcal{B}, S \models C_B$ **then**
          current_cost $\leftarrow g^l(S) \times \min_{B \in \mathcal{B}: S \models C_B} \mathsf{cost}_B(S)$ ▷ Compute the solution cost.
          if current_cost $< \overline{\gamma}_i$ **then**
            $\overline{\gamma}_i, \overline{\gamma}^l \leftarrow$ current_cost, $\overline{\sigma} \leftarrow \sigma^l(S)$ ▷ Update the best solution.
        else
          for all $\tau \in \mathcal{T}(S)$ with $S[\![\tau]\!] \models \mathcal{C}$ **do**
            $g_{\text{current}} \leftarrow g^l(S) \times w_\tau(S)$ ▷ Compute the g-value.
            if $g_{\text{current}} \times \eta(S[\![\tau]\!]) < \overline{\gamma}_i$ **then**
              $j \leftarrow$ OWNER$(S[\![\tau]\!])$ ▷ Determine the owner.
              if $j \in J$ **then**
                SEND$(Q_j, (S[\![\tau]\!], \langle \sigma(S); \tau \rangle, g_{\text{current}}, \eta(S[\![\tau]\!])))$ ▷ Send a message.
              else
                PUSH$(P_j, (S[\![\tau]\!], \langle \sigma(S); \tau \rangle, g_{\text{current}}, \eta(S[\![\tau]\!])))$ ▷ Buffer a message.
      else if not sent_all and $|J| = k$ **then**
        sent_all $\leftarrow \top$, SEND(Q_j, NULL) for $j = 1, ..., k$ ▷ Notify that all states were sent.
    if $i = \alpha \vee (\text{empty} \wedge i = 1)$ **then**
      return $\overline{\sigma}$, (empty $\wedge$ complete) $\vee\, \underline{\gamma} = \overline{\gamma}_i$, $\underline{\gamma}$ ▷ Return the solution.
    else if $\alpha \neq$ NULL $\vee$ empty **then**
      break ▷ Terminate.
    $l \leftarrow l + 1$ ▷ Proceed to the next layer.
    $O \leftarrow \{S \in G \mid g^l(S) \times \eta(S) < \overline{\gamma}_i\}$, $\underline{\gamma}^l \leftarrow \min\{\overline{\gamma}_i, \underline{\gamma}^D, \min_{S \in O} g^l(S) \times \eta(S)\}$
    if $|O| > b/k$ **then**
      $O \leftarrow$ the best b/k states in O, complete $\leftarrow \bot$ ▷ Keep the best b/k states.
      $\underline{\gamma}^D \leftarrow \min\left\{\underline{\gamma}^D, \min_{S \in \text{discarded states from } O} g^l(S) \times \eta(S)\right\}$ ▷ Check discarded states.
    SEND$(R_j, (i, \overline{\sigma} \neq \text{NULL}, O = \emptyset, \text{complete}, \overline{\gamma}^{l-1}, \underline{\gamma}^l))$ for $i = 1, ..., k$

is equivalent to Lemma 6 for HDBS1.

Algorithm 24 Receive the information of a layer. A message queue R_i, a set of process indices J, a process index α, flags empty and complete, the best solution cost found in the previous layer $\overline{\gamma}^{l-1}$, and a candidate for the dual bound on the optimal cost $\underline{\gamma}^l$ are given as input.

1: **while** R_i has an incoming message and $|J| < k$ **do**
2: $(j, \text{solution\_found}_j, \text{empty}_j, \text{complete}_j, \overline{\gamma}_j, \underline{\gamma}_j) \leftarrow \text{RECV}(R_i)$
3: $J \leftarrow J \cup \{j\}$
4: $\text{empty}_i \leftarrow \text{empty}_i \wedge \text{empty}_j$, $\text{complete} \leftarrow \text{complete} \wedge \text{complete}_j$
5: **if** solution_found_j and $(\overline{\gamma}_j < \overline{\gamma}^{l-1} \vee (\overline{\gamma}_j = \overline{\gamma}^{l-1} \wedge j < \alpha))$ **then**
6: $\overline{\gamma}^{l-1} \leftarrow \overline{\gamma}_j$, $\alpha \leftarrow j$
7: $\underline{\gamma}^l = \min\left\{\underline{\gamma}^l, \underline{\gamma}_j\right\}$

Lemma 10. *In Algorithm 23, the following two conditions hold:*

- *During the while loop in lines 10–33, a process receives only messages in layer l.*
- *When a process exits the while loop, it has received all messages sent to it in layer l.*

Proof. Following the proof of Lemma 6, when a process enters the while loop with $l = m$, we assume that no message in layer $l \leq m-1$ was sent to a process, or all messages in layer $l \leq m-1$ sent to that process were received. We prove that this assumption implies the lemma for $l = m$. By the assumption, all messages received by a process during the while loop are in layer $l \geq m$. If process i sends a message in layer $l > m$ to any other process j via the message queue R_j, i needs to exit the while loop with $l = m$ and reach line 38 to increase l. Suppose that i is the first process that exits the while loop with $l = m$. Then, process i only receives messages in layer m since all other processes have not reached layer $l > m$. To exit the while loop, process i needs to receive k special messages in layer m. Since one process sends only one special message in layer m to each process, process i needs to receive one special message from each process. When process j sends the special messages in line 33, it has already received k messages via the message queue R_j since $|J| = k$ is required by line 32. Each of k processes sends only one message in layer m to process j via R_j, so process j has received all k messages in layer m via R_j. Since line 1 in Algorithm 24 requires $|J| < k$, process j will not receive more messages via R_j after sending the special messages and before exiting the while loop. Therefore, when process i exits the while loop with $l = m$, in terms of the message queues $R_1, .., R_k$, all processes have received all k messages in layer m and will not receive messages in layer $l > m$ before exiting the while loop.

If process i sends a message in layer $l > m$ to process j via the message queue Q_j, process i needs to exit the while loop with $l = m$, enter the while loop with $l = m+1$, and reach line 15 or 29, which requires $j \in J$. To achieve $j \in J$, after entering the while loop with $l = m+1$, process i needs to receive a message from process j via the message queue R_i. By the previous paragraph, all messages in layer m via R_i were already received, so this message from process j should be in layer $l > m$. Therefore, process j should have exited the while loop with $l = m+1$ when process i reaches line 15 or 29, and process j in layer m does not receive messages in layer $m+1$ via Q_j. Thus, the first condition holds for $l = m$. The rest of the proof, i.e., proving the second condition for $l = m$ and the first condition for $l = m+1$, is the same as the last two paragraphs in the proof of Lemma 6. □

Since Theorem 35, which states that HDBS1 searches layer by layer, only depends on Lemma 6,

we can derive a corresponding theorem for HDBS2 from Lemma 10.

Theorem 40. *In Algorithm 23, in lines 40–43 and lines 8–33, each state $S \in O$ satisfies $|\sigma^l(S)| = l$.*

As in Theorem 36 for HDBS1, we prove that HDBS2 terminates in finite time given a finite and acyclic DyPDL model. We prove lemmas corresponding to Lemmas 7 and 8 first.

Lemma 11. *Given a finitely defined DyPDL model, in Algorithm 23, one iteration of the while loop in lines 10–33 finishes in finite time.*

Proof. While HDBS2 has additional operations in each iteration compared to HDBS1, except for line 11, which calls Algorithm 24, it is clear that all operations finish in finite time. In Algorithm 24, by Lemma 10, HDBS2 receives only messages in layer l. Since each process sends one message in layer l, Algorithm 24 receives at most k messages in each iteration. Therefore, line 11 in Algorithm 23 finishes in finite time. □

Lemma 12. *Algorithm 23 satisfies the following three conditions:*

- *When the first process enters the while loop in lines 10–33 with $l = m$, all processes will enter the while loop with $l = m$ in finite time.*
- *Once a process enters the while loop with $l = m$, it will expand all states in O, generate all successor states, send all messages in layer m, and receive all messages in layer m in finite time.*
- *When the last process enters the while loop with $l = m$, all processes are in the while loop with $l = m$ and will exit in finite time.*

Proof. Following the proof of Lemma 8, we assume that the first condition of the lemma holds up to $m = n$ and prove the second and third conditions for $l = n$ and the first condition for $l = n+1$. The assumption is valid for $n = 0$ as in the original proof. It is also straightforward that when the last process enters the while loop with $l = n$, all processes are in the while loop with $l = n$ since they have not received the special message NULL in layer n from the last process. If $n = 0$, all processes have reached line 6 and sent messages in layer 0 to all processes via message queues $R_1, ..., R_k$ before entering the while loop. If $n > 0$, all processes have reached line 43 and sent messages in layer n to all processes via message queues $R_1, ..., R_k$ before entering the while loop. In either case, all k messages in layer n have been sent to each process via $R_1, ..., R_k$. With a similar argument to the original proof, using Lemmas 10 and 11, we can prove that each process i receives the k messages in layer n via R_i in finite time after entering the while loop with $l = n$. Once receiving these k messages, $|J|$ becomes k, and process i will send all messages in layer n in finite time in lines 15, 29, and 33. Each process will also receive these messages in finite time and exit the while loop, so the second and third conditions hold.

When a process i exits the while loop with $l = n$, by Lemma 10, it has received all messages in layer n. As discussed in the previous paragraph, all processes have already exited or will exit the while loop with $l = n$ in finite time. Process i received one message in layer n from each process via R_i, and each process sent the same messages in layer n to all processes via $R_1, ..., R_k$. Therefore, in all processes, Algorithm 24 processes the same set of messages in layer n during the while loop with $l = n$. By lines 5–6 in Algorithm 24, regardless of the order in which the messages arrived, α is

uniquely determined. If $\alpha \neq \text{NULL}$ in process i, it holds in all processes. In such a case, all processes terminate, and process α returns a solution. Otherwise, $\alpha = \text{NULL}$, and all processes proceed to the next iteration. When the first process enters the while loop with $l = n + 1$, all processes will exit or have already exited the while loop with $l = n$ and will enter the while loop with $l = n + 1$ in finite time, so the first condition of the lemma holds for $l = n + 1$. □

Similarly to Theorem 36 for HDBS1, by using Theorem 40 and Lemma 12, we can prove that HDBS2 terminates given a finite and acyclic DyPDL model.

Theorem 41. *Given a finite, acyclic, and monoidal DyPDL model, HDBS2 terminates in finite time.*

We also omit the proof for the following theorem as it is exactly the same as that of Theorem 37.

Theorem 42. *In line 22 of Algorithm 23, if $\overline{\sigma} \neq \text{NULL}$, then $\overline{\sigma}$ is a solution for the model with $\overline{\gamma}^l = \mathsf{cost}_{\overline{\sigma}}(S^0)$.*

Finally, we show theorems corresponding to Theorems 38 and 39.

Lemma 13. *In Algorithm 23, suppose that a solution exists for the DyPDL model, and let $\hat{\gamma}$ be its cost. In each layer, there exists a process i where at least one of the following conditions is satisfied when it exits the while loop in 10–33:*

- *complete $= \bot$.*
- *$\overline{\gamma}_i \leq \hat{\gamma}$.*
- *G contains a state $\hat{S}$ such that an $\hat{S}$-solution $\hat{\sigma}$ exists, and $\langle \sigma^{l+1}(\hat{S}); \hat{\sigma} \rangle$ is a solution for the model with $\mathsf{cost}_{\langle \sigma^{l+1}(\hat{S}); \hat{\sigma} \rangle}(S^0) \leq \hat{\gamma}$.*

Proof. The proof is straightforward from the proof of Lemma 9 while we use G instead of O and $l + 1$ instead of l in the third condition. We use Lemma 12 instead of Lemma 8 used in the original proof. □

Theorem 43. *Let $\langle A, \times, \mathbf{1} \rangle$ be a monoid where $A \subseteq \mathbb{Q} \cup \{-\infty, \infty\}$ and A is isotone. Given a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$ with $\langle A, \times, \mathbf{1} \rangle$ and $\overline{\gamma} \in A$, if an optimal solution for the minimization problem with the model exists and has the cost γ^*, in lines 4–43, $\underline{\gamma} \leq \gamma^*$.*

Proof. The theorem holds just after line 4 since $\underline{\gamma} = \eta(S^0) \leq \gamma^*$. Only line 13 updates $\underline{\gamma}$. In line 39, O is updated to $\{S \in G \mid g^l(S) \times \eta(S) \leq \overline{\gamma}_i\}$, and $\underline{\gamma}^l$ is computed. By a similar argument to the proof of Theorem 38, by Lemma 13, there exists a process j such that $\underline{\gamma}^l \leq \gamma^*$ holds. Let this $\underline{\gamma}^l$ be $\underline{\gamma}^l_j$. When process i reaches line 13 with $|J| = k$, it has received all messages in layer l via the message queue R_i using Algorithm 24. Therefore, $\underline{\gamma}^l$ in process i becomes at most $\underline{\gamma}^l_j$, and updating $\underline{\gamma}$ to $\underline{\gamma}^l$ does not break the condition. □

Theorem 44. *Let $\langle A, \times, \mathbf{1} \rangle$ be a monoid where $A \subseteq \mathbb{Q} \cup \{-\infty, \infty\}$ and A is isotone. Given a monoidal DyPDL model $\langle \mathcal{V}, S^0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$ with $\langle A, \times, \mathbf{1} \rangle$ and $\overline{\gamma} \in A$, if an optimal solution exists for the minimization problem with the model, and HDBS2 returns $\overline{\sigma} \neq \text{NULL}$ and is_optimal $= \top$, then $\overline{\sigma}$ is an optimal solution. If HDBS2 returns $\overline{\sigma} = \text{NULL}$ and is_optimal $= \top$, then there does not exist a solution whose cost is less than $\overline{\gamma}$.*

Proof. Following the proof of Theorem 39 for HDBS1, we consider is_optimal = $\top$ is returned and focus on two cases: empty = complete = $\top$ or $\underline{\gamma} = \overline{\gamma}_i$ in line 35.

Suppose that empty = complete = $\top$. Then, when each process reached line 43 in the current layer, $O = \emptyset$. Let the current layer be m. No state was inserted into G in the while loop in layer m, and $G = \emptyset$. If there exists a solution whose cost is $\hat{\gamma} < \overline{\gamma}$, by Lemma 13, $\overline{\gamma}_j \leq \hat{\gamma}$ in some process j. Since $\overline{\gamma}_j$ was initialized to be $\overline{\gamma} > \hat{\gamma}$, it was updated by line 12 or line 22. If $\overline{\gamma}_j$ was updated by line 12 in layer $n \leq m$, by Algorithm 24, some process β found a solution with the cost $\overline{\gamma}^{n-1} = \overline{\gamma}_j \leq \hat{\gamma}$ in layer $n-1$. In this case, in layer n, solution_found$_\beta \neq$ NULL in Algorithm 24 and $\alpha \neq$ NULL, so all processes would not proceed to layer $n+1$. Thus, $n = m$, and the solution was found in the previous layer $m-1$. If $\overline{\gamma}_j$ was updated by line 22, process j found a solution with the cost $\overline{\gamma}_j$. In layer m, since $O = \emptyset$ when process j entered the while loop, $\overline{\gamma}_j$ was not updated by line 22. Thus, the solution was found in the previous layer $m-1$. By Algorithm 24, process $i = \alpha$ has found the best solution in layer $m-1$, so $\overline{\gamma}_\alpha^{m-1} \leq \overline{\gamma}_j^{m-1} = \overline{\gamma}_j$. Process i returns a solution with the cost $\overline{\gamma}_i^{m-1} \leq \overline{\gamma}_j \leq \hat{\gamma} < \overline{\gamma}$. Therefore, when empty = complete = $\top$, if a solution whose cost is less than $\overline{\gamma}$ exists, HDBS2 returns a solution whose cost is at most $\hat{\gamma} < \overline{\gamma}$. If an optimal solution exists, HDBS2 returns an optimal solution. By contraposition, when empty = complete = $\top$, if HDBS2 returns $\overline{\sigma} =$ NULL, there is no solution whose cost is less than $\overline{\gamma}$.

Suppose that $\underline{\gamma} = \overline{\gamma}_i$. If there exists an optimal solution with the cost γ^* and $\overline{\sigma} \neq$ NULL, by a similar argument to the previous paragraph, process i has reached line 22 in the previous layer $m-1$, and $\overline{\gamma}^{m-1} = \overline{\gamma}_i^{m-1} = \overline{\gamma}_\alpha^{m-1}$ in Algorithm 24. At that time, $\overline{\gamma}_i$ was updated to the cost of $\overline{\sigma}$. If process i reached line 22 also in layer m, $\overline{\sigma}$ was updated to a new solution, and $\overline{\gamma}_i$ was updated to the cost of the solution. Therefore, when process i returns the solution $\overline{\sigma}$, $\overline{\gamma}_i$ is the cost of the solution. Since $\overline{\gamma}_i \geq \gamma^*$ and $\overline{\gamma}_i = \underline{\gamma} \leq \gamma^*$ by Theorem 43, $\overline{\gamma}_i = \gamma^*$, and $\overline{\sigma}$ is an optimal solution. If $\underline{\gamma} = \overline{\gamma}_i$ and $\overline{\sigma} =$ NULL, by a similar argument to the proof of Theorem 39, there is no solution with a smaller cost than $\overline{\gamma}$. □

Comparison to Previous Work

HDBS is based on hash-based work distribution with asynchronous message passing, which was proposed by previous work [127, 367]. While hash-based work distribution has been used for BFS algorithms including A* [127, 253, 254, 235, 272] and iterative deepening A* (IDA*) [258] with a transposition table [353], it has not previously been used to parallelize beam search. Barnat, Brim, and Chaloupka [18] used an idea similar to hash-based work distribution with layer synchronization to parallelize BrFS for model checking. A high-level overview of their algorithm is the same as hash-based work distribution: each process expands a state from its open list and sends a successor state to its owner process using message passing. However, the owner process is determined by a mechanism specific to model checking instead of a hash function. Their layer synchronization mechanism is similar to HDBS1: all processes are synchronized after expanding all states in the current layer.

6.3 Experimental Evaluation

We implement multi-thread DIDP solvers using CABS with SBS (CASBS), HDBS1 (CAHDBS1), and HDBS2 (CAHDBS2) on top of didp-rs 0.4.0[7] with Rust 1.65.0. For SBS, we use DashMap 5.4.0 and Rayon 1.7.0. For the number of segments in the concurrent hash table, we follow the default of DashMap 5.4.0: 4 times the number of threads, i.e., $K = 4k$. We use Fx Hash from rust-hash 1.1.0[8] for the hash function of HDBS. While HDBS can be implemented for a distributed environment, we focus on a shared memory environment and use threads instead of processes. We use Crossbeam Channel 0.5.8[9] for message passing and bus 2.4.0[10] for broadcast in HDBS1.

We experimentally compare the multi-thread DIDP solvers with the single-thread CABS solver, a commercial multi-thread MIP solver (Gurobi 10.0.1), and a commercial CP solver (IBM ILOG CP Optimizer 22.1.0). To evaluate the performance of the solvers, we use three metrics in Section 4.3.3: the coverage, the optimality gap, and the primal integral. Coverage is the number of optimally solved instances. The optimality gap is the relative difference between primal and dual bounds obtained by a solver. The primal integral is the integral of the primal gap, the relative difference between the primal bound and the best-known solution cost, over time. In addition, we evaluate the speedup achieved by the multi-thread solvers.

All experiments are run on a machine with 2 Intel Xeon Gold 6148 CPUs (40 cores in total). While we used 8 GB memory limit for each run in the previous chapters (Chapters 4 and 5), since multi-thread solvers can use multiple CPU cores, we give all available memory, 188 GB, to each run as a memory limit. Due to the limitation of the available computational resources, we use a shorter time limit, 5 minutes instead of 30 minutes for both single-thread and multi-thread solvers. In addition, we focus on representative problem classes from those used in the previous chapters. First, we select problem classes where we have a strong motivation to use CABS. In the experiment comparing the sequential solvers (Section 4.3.5), there are four problem classes where CABS has the highest coverage, lowest optimality gap, and lowest primal integral, outperforming MIP and CP: the traveling salesperson problem with time windows (TSPTW), the simple assembly line balancing problem (SALBP-1), the minimization of open stacks problem (MOSP), and graph-clear. We use these four problem classes in our evaluation. In addition, we use problems where using MIP or CP is better than DIDP: the capacitated vehicle routing problem (CVRP) and bin packing. For CVRP, we focus on A, B, E, F, and P instance sets from CVRPLIB [425] since no sequential solver could optimally solve any instance in other instance sets even with the 30-minute time limit, and thus we are unable to measure the speedup.

Since this chapter is based on the work [269] done before the previous chapters, there are other differences in the experimental settings. As mentioned above, our implementation is based on didp-rs 0.4.0, which is older than the version (0.7.0) used in Chapters 4 and 5. While the multi-thread DIDP solvers are also implemented in didp-rs 0.7.0, we have not re-run experiments with it due to the limitation of computational resources. In terms of algorithms, the only difference between the two versions is the computation of the dual bound on the optimal cost in beam search. As explained in Section 6.2.1, beam search computes the dual bound on the optimal cost by maintaining $\underline{\gamma}^O$, the minimum $g^l(S) \times \eta(S)$ of the states in the open list, and $\underline{\gamma}^D$, the minimum $g^m(S) \times \eta(S)$ of discarded

[7] https://github.com/domain-independent-dp/didp-rs/releases/tag/parallel-aaai24
[8] https://crates.io/crates/rustc-hash/1.1.0
[9] https://crates.io/crates/crossbeam-channel/0.5.8
[10] https://crates.io/crates/bus/2.4.0

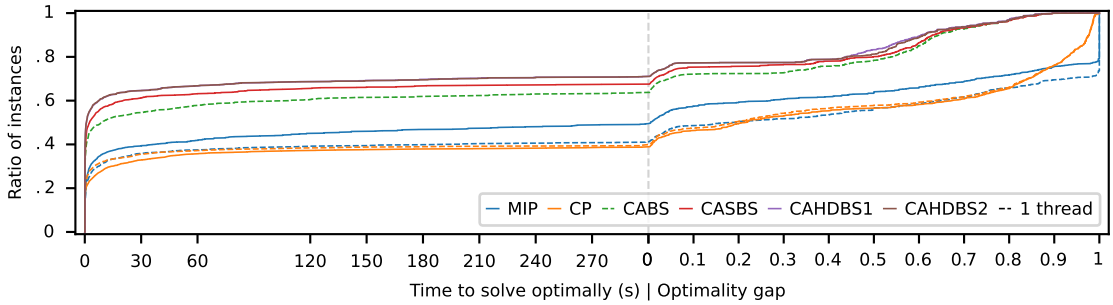


Figure 6.1: The ratio of the coverage against time and the ratio of instances against the optimality gap averaged over all problem classes using 1 thread and 32 threads.

states in layers $m = 1, ..., l-1$. As explained in Sections 6.2.2 and 6.2.3, SBS and HDBS also perform similar computation. In our implementation of sequential beam search and SBS based on didp-rs 0.4.0, we only maintain $\underline{\gamma}^O$ and compute the dual bound on the optimal cost only when no state is discarded, i.e., $\underline{\gamma}^D = \infty$. Since we do not obtain the dual bound once beam search discards a state, it potentially results in a worse optimality gap than the newer version. Similarly, in HDBS1, Algorithm 22 computes $\underline{\gamma}$ only when complete $= \top$. In HDBS2, Algorithm 24 computes $\underline{\gamma}^l$ only when complete $= \top$. Note that in Algorithm 23, when $|O| > b/k$, HDBS2 sends the flag complete after line 41, where complete becomes $\bot$. However, $\underline{\gamma}^l$ was computed before discarding states in line 39, so HDBS2 could actually compute the dual bound on the optimal cost using $\underline{\gamma}^l$ if that is the first time to reach line 41. As a result, in terms of the dual bound on the optimal cost, HDBS2 can be worse than sequential beam search and HDBS1. We emphasize that these implementations in the older version give the same disadvantages to sequential beam search, SBS, and HDBS1 and a larger disadvantage to HDBS2. The performance of the parallel DIDP solvers can be improved by using the newer version.

Another difference is that we use DIDPPy (Section 3.2.3) as the modeling interface for the DyPDL models instead of didp-yaml used in the previous chapters. For SALBP-1, bin packing, MOSP, and graph-clear, we use DyPDL models equivalent to those implemented in YAML-DyPDL. For TSPTW and CVRP, there is a minor difference. In the DyPDL models for TSPTW and CVRP, a solution corresponds to a tour visiting all customers starting from and returning to the depot. In the DyPDL models presented in Section 3.3, returning to the depot is represented by a base case whose cost is the distance between the current location and the depot. In the DyPDL models used in this chapter, returning to the depot is represented by a transition, which updates the current location to the depot and increases the cost by the distance. In the base case, whose cost is zero, all customers must be visited, and the current location must be the depot. For these DyPDL models, while we use DIDPPy, we present equivalent YAML-DyPDL domain files in Appendix A.2. For MIP and CP, we use the same model as Section 4.3, and the versions of Gurobi and CP Optimizer are also the same. We use Python 3.11.2 for all solvers with jemalloc 5.2.1 [126] as the memory allocator.

6.3.1 Results

First, we evaluate the multi-thread solvers using the metrics used for the sequential solvers in the previous chapters. Figure 6.1 shows the coverage and the optimality gap achieved with 1 thread and 32 threads. In the left-hand side, the x-axis is time in seconds, and the y-axis is the ratio of

Table 6.1: Coverage of multi-thread solvers with 32 threads and sequential solvers. The coverage of a multi-thread DIDP solver is in bold if it is higher than MIP, CP, and sequential CABS, and the higher of MIP and CP is in bold if there is no better DIDP solver. The highest coverage is underlined.

	MIP		CP		CABS	CASBS	CAHDBS1	CAHDBS2
#threads	1	32	1	32	1	32	32	32
TSPTW (340)	192	239	42	27	235	**260**	**262**	**262**
CVRP (90)	13	**29**	0	0	5	6	8	8
Bin Packing (1615)	1122	1192	1189	**1251**	1110	1077	1239	1239
SALBP-1 (2100)	1354	1397	1563	1581	1714	**1818**	**1824**	**1824**
MOSP (570)	216	238	421	397	507	**526**	**531**	**531**
Graph-Clear (135)	6	16	4	3	92	**103**	**113**	**113**

Table 6.2: Average optimality gap of multi-thread solvers with 32 threads and sequential solvers. The optimality gap of a multi-thread DIDP solver is in bold if it is lower than MIP, CP, and sequential CABS, and the lower of MIP and CP is in bold if there is no better DIDP solver. The lowest optimality gap is underlined.

	MIP		CP		CABS	CASBS	CAHDBS1	CAHDBS2
#threads	1	32	1	32	1	32	32	32
TSPTW (340)	0.3174	0.2128	0.7292	0.7572	0.1607	**0.1239**	**0.1170**	**0.1174**
CVRP (90)	0.8556	0.6778	0.9743	0.9736	0.5985	**0.5936**	**0.5747**	**0.5763**
Bin Packing (1615)	0.0491	0.0357	0.0082	**0.0044**	0.0060	0.0086	0.0054	0.0055
SALBP-1 (2100)	0.3130	0.2654	0.0166	0.0148	0.0077	**0.0062**	**0.0051**	**0.0051**
MOSP (570)	0.3768	0.3202	0.2135	0.2473	0.0777	**0.0591**	**0.0488**	**0.0515**
Graph-Clear (135)	0.6454	0.5880	0.4702	0.4655	0.1675	**0.1251**	**0.0786**	**0.0885**

coverage over the number of instances achieved with x seconds. In the right-hand side, the x-axis is the optimality gap, and the y-axis is the ratio of instance where the optimality gap is less than or equal to x. The y-axis is averaged over all problem classes. We present a plot presenting the result for each problem class in Appendix E, and Tables 6.1 and 6.2 summarize the results by presenting the coverage and the average optimality gap achieved by the time limit in each problem class.

The multi-thread DIDP solvers outperform multi-thread MIP and CP on average and in TSPTW, SALBP-1, MOSP, and graph-clear, problem classes where sequential CABS is better than sequential MIP and CP. MIP with 32 threads has the highest coverage in CVRP, and CP with 32 threads has the highest coverage and the lowest optimality gap in bin packing. In summary, parallelization does not change the ranking of the different solution paradigms on any problem class.

The multi-thread DIDP solvers show the performance improvement over the sequential CABS. In particular, CAHDBS1 and CAHDBS2 are better than CASBS. CAHDBS1 is slightly better than CAHDBS2 in the optimality gap on average and in all problem classes except for SALBP-1. To investigate this difference between CAHDBS1 and CAHBDS2, we evaluate the average primal gap in Table 6.3. In all problem classes, CAHDBS1 and CAHDBS2 achieve almost the same primal gap, i.e., they almost always find the same quality solution within the time limit. Therefore, the difference in the optimality gap comes from the difference in the dual bound, which is probably caused by our unideal implementations of HDBS1 and HDBS2 based on didp-rs 0.4.0 as described above.

CASBS solves fewer instances than sequential CABS in bin packing; while CASBS solves 69

Table 6.3: Average primal gap of multi-thread solvers with 32 threads and sequential solvers. The primal gap of a multi-thread DIDP solver is in bold if it is lower than MIP, CP, and sequential CABS, and the lower of MIP and CP is in bold if there is no better DIDP solver. The lowest primal gap is underlined.

	MIP		CP		CABS	CASBS	CAHDBS1	CAHDBS2
#threads	1	32	1	32	1	32	32	32
TSPTW (340)	0.3131	0.2101	0.0282	0.0246	0.0061	**0.0028**	**<u>0.0017</u>**	**<u>0.0017</u>**
CVRP (90)	0.2319	0.1142	0.0847	0.0506	0.0546	0.0552	**0.0402**	**<u>0.0401</u>**
Bin Packing (1615)	0.0470	0.0338	0.0057	**<u>0.0016</u>**	0.0028	0.0058	0.0025	0.0026
SALBP-1 (2100)	0.3123	0.2637	0.0110	0.0092	0.0021	**0.0012**	**<u>0.0003</u>**	**0.0003**
MOSP (570)	0.0595	0.0499	0.0082	0.0053	0.0003	0.0004	**<u>0.0000</u>**	**<u>0.0000</u>**
Graph-Clear (135)	0.2135	0.1873	0.0478	0.0403	0.0001	0.0001	**<u>0.0000</u>**	**<u>0.0000</u>**

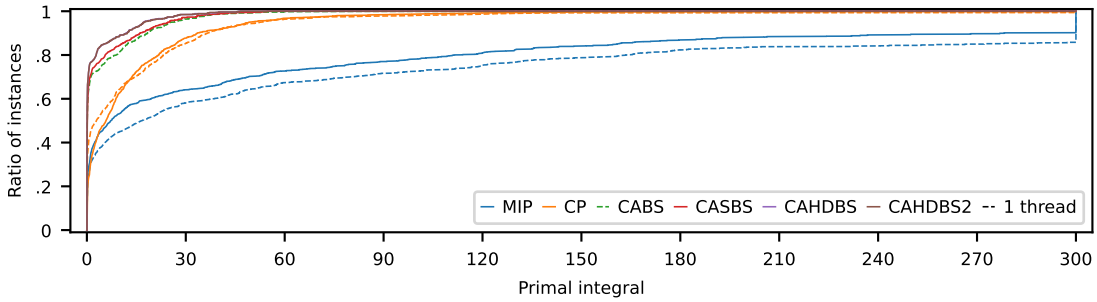


Figure 6.2: The ratio of instances against the primal integral averaged over all problem classes using 1 thread and 32 threads.

instances not solved by CABS, it fails to solve 102 instances solved by sequential CABS. We also observe that among bin packing instances solved by sequential CABS, CAHDBS1 fails to solve 47 instances, and CAHDBS2 fails to solve 52 instances. In other problem classes, the multi-thread DIDP solvers solve all instances solved by sequential CABS.

CP with 32 threads has lower coverage than sequential CP in TSPTW, MOSP, and graph-clear and higher optimality gap in TSPTW and MOSP, resulting in the inferior performance on average in Figure 6.1. However, as shown in Table 6.3, CP with 32 threads achieves better solution quality in these problem classes.

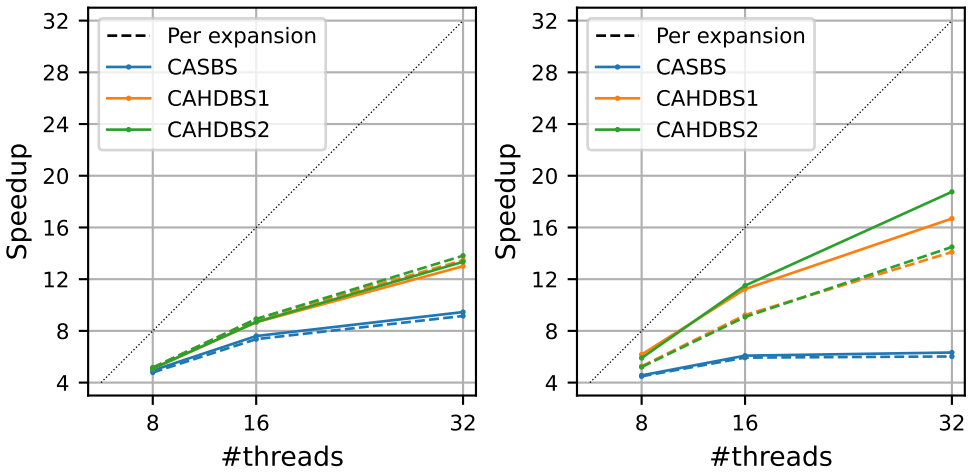
Figure 6.2 shows the ratio of instances against the primal integral averaged over all problem classes. We present a plot presenting the result for each problem class in Appendix E and show the average primal integral in each problem class in Table 6.4. CAHDBS2 and CAHDBS2 are better than CASBS on average and in all problem classes, and the difference between CAHDBS1 and CAHDBS2 is small. Again, while parallelization improves the performance of DIDP in most cases, the ranking is not changed.

6.3.2 Speedup

We now evaluate the speedup achieved by the multi-thread DIDP solvers. Focusing on TSPTW, SALBP-1, MOSP, and graph-clear, where we have a strong motivation to use CABS, we present the geometric mean speedup using 8, 16, and 32 threads in Figure 6.3. We take the geometric mean over instances optimally solved by the sequential solver in 10 to 300 seconds. As discussed

Table 6.4: Average primal integral of multi-thread solvers with 32 threads and sequential solvers. The primal integral of a multi-thread DIDP solver is in bold if it is lower than MIP, CP, and sequential CABS, and the lower of MIP and CP is in bold if there is no better DIDP solver. The lowest primal integral is underlined.

	MIP		CP		CABS	CASBS	CAHDBS1	CAHDBS2
#threads	1	32	1	32	1	32	32	32
TSPTW (340)	108.26	77.45	9.56	17.10	2.88	**1.48**	**1.08**	**<u>1.07</u>**
CVRP (90)	121.80	81.47	28.26	19.66	18.63	**18.49**	**<u>13.76</u>**	**13.80**
Bin Packing (1615)	19.26	15.45	4.55	**<u>1.33</u>**	4.91	3.61	1.77	1.78
SALBP-1 (2100)	99.29	87.96	16.53	15.96	1.96	**0.95**	**<u>0.46</u>**	**0.47**
MOSP (570)	26.37	22.89	4.19	4.85	0.21	0.32	**<u>0.09</u>**	**0.10**
Graph-Clear (135)	80.45	72.62	22.60	22.04	0.32	**0.22**	**<u>0.08</u>**	**<u>0.08</u>**



(a) TSPTW (b) SALBP-1

(c) MOSP (d) Graph-Clear

Figure 6.3: Geometric mean speedup in instances optimally solved by sequential CABS in 10-300 seconds. 'Per expansion' shows the speedup of search time per expansion.

Table 6.5: Speedup of multi-thread solvers with 32 threads against a sequential solver. 's.' shows the speedup of the search time, 's./e.' shows the speedup of the search time per expansion, and 'e.r.' shows the expansion ratio. All values are geometric mean over instances solved in 10-300 seconds by a sequential solver. The largest speedup among the DIDP solvers is in bold.

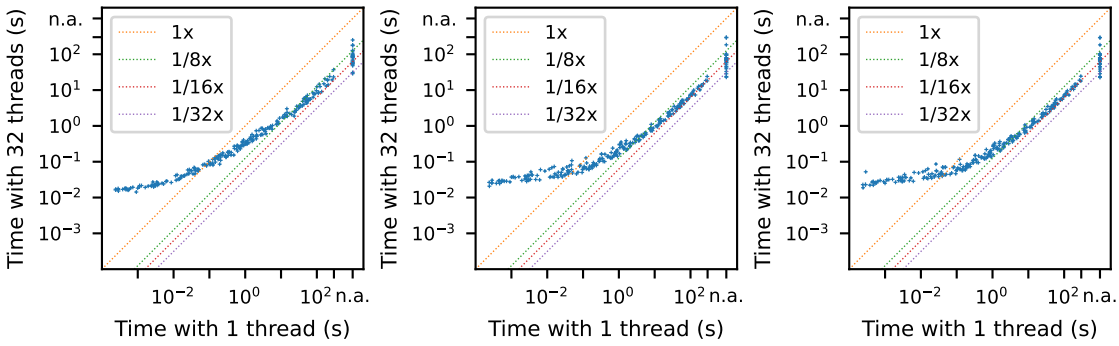
	MIP	CP	CASBS			CAHDBS1			CAHDBS2		
	s.	s.	s.	s./e..	e.r.	s.	s./e.	e.r.	s.	s./e.	e.r.
TSPTW (340)	4.2	0.1	9.5	9.2	0.968	13.0	13.5	1.036	**13.3**	**13.8**	1.035
CVRP (90)	5.3	-	5.2	5.2	0.983	**9.3**	**9.3**	0.999	**9.3**	**9.3**	1.000
Bin Packing (1615)	6.4	9.2	3.9	4.2	1.089	36.4	10.5	0.288	**39.6**	**11.0**	0.278
SALBP-1 (2100)	2.0	1.4	6.3	6.0	0.952	16.7	14.1	0.845	**18.8**	**14.5**	0.773
MOSP (570)	3.1	0.3	5.4	5.3	0.987	8.9	9.0	1.012	**9.0**	**9.1**	1.015
Graph-Clear (135)	2.0	3.2	6.0	6.0	0.999	**10.4**	**10.5**	1.011	10.3	**10.5**	1.011

in Section 6.3.1, the multi-thread DIDP solvers fail to solve some instances in bin packing. When taking the geometric mean, such instances are excluded. All multi-thread DIDP solvers continue to speed up as we increase the number of threads up to 32 threads. Comparing the different solvers, similar to other metrics presented above, CAHDBS2 and CAHDBS1 achieve better speedup than CASBS. CAHDBS2 is better than CAHDBS1 in TSPTW and SALBP-1, and the difference in MOSP and graph-clear is small. In Appendix E, we present the coverage over time and the distributions of the optimality gap and the primal integral in TSPTW, SALBP-1, MOSP, and graph-clear for CAHDBS2 with 8, 16, and 32 threads.

Table 6.5 presents the geometric mean speedup ('s.') using 32 threads in all problem classes. The multi-thread DIDP solvers achieve a higher speedup than MIP and CP in all problem classes. CP even shows slowdown in TSPTW and MOSP while the solution quality is improved in these problem classes as shown in Table 6.3. Note that a larger speedup does not necessarily mean better performance when the baseline sequential solvers are different. Indeed, using 32 threads does not change the ranking of the paradigms, as observed in Section 6.3.1. It does however show that the solver is making better use of the computational resources made available.

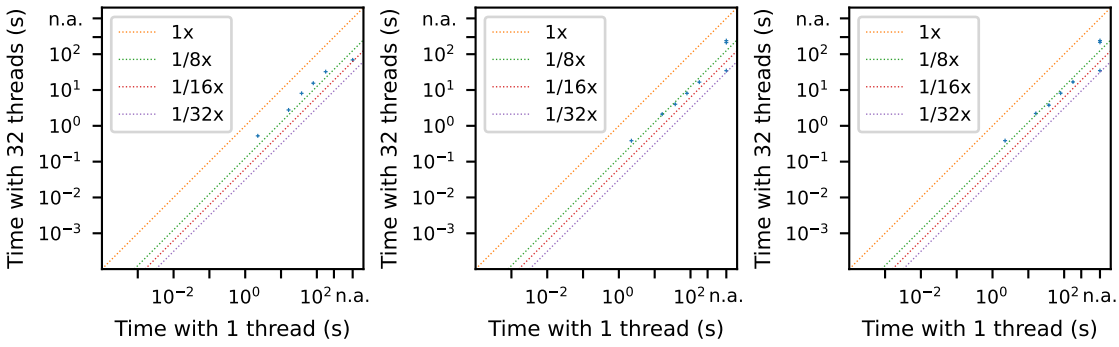
CAHDBS1 and CAHDBS2 achieve a super linear speedup in bin packing, which is likely to come from the different search behavior caused by parallelization. To normalize such effect, in Table 6.5, we evaluate the mean speedup per expansion ('s./e.'), the speedup of the average time taken to expand one state, and the mean expansion ratio ('e.r.'), the number of expansions by a multi-thread CABS divided by that of sequential CABS. We also present the speedup per expansion in Figure 6.3. We do not observe a super linear speedup per expansion. CASBS expands slightly fewer states than sequential CABS in all problem classes except for bin packing, where it expands more. For CAHDBS1 and CAHDBS2, in TSPTW, CVRP, MOSP, and graph-clear, the actual speedup is almost the same as the speedup per expansion. In bin packing and SALBP-1, CAHDBS1 and CAHDBS2 expand significantly fewer states than sequential CABS, which makes the actual speedup larger than the speedup per expansion. Such phenomenon is known as speedup anomaly in parallel branch-and-bound [284], and a similar behavior is observed in parallel BFS [272, 271]. Comparing CAHDBS1 and CAHDBS2 in bin packing and SALBP-1, CAHBDS2 expands fewer states than CAHDBS1, resulting in a better speedup. In terms of the speedup per expansion, CAHDBS2 is still slightly better than CADHBS1.

For a detailed evaluation of speedup, we compare the time taken by sequential and 32-thread



(a) CASBS (b) CAHDBS1 (c) CAHDBS2

Figure 6.4: Comparison of sequential and 32-thread CABS in time to optimally solve each instance of TSPTW. Unsolved instances are shown at 'n.a.'.

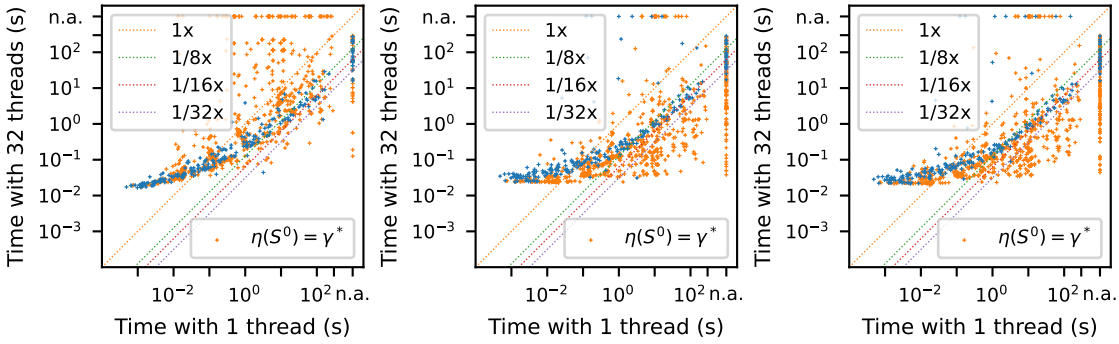


(a) CASBS (b) CAHDBS1 (c) CAHDBS2

Figure 6.5: Comparison of sequential and 32-thread CABS in time to optimally solve each instance of CVRP. Unsolved instances are shown at 'n.a.'.

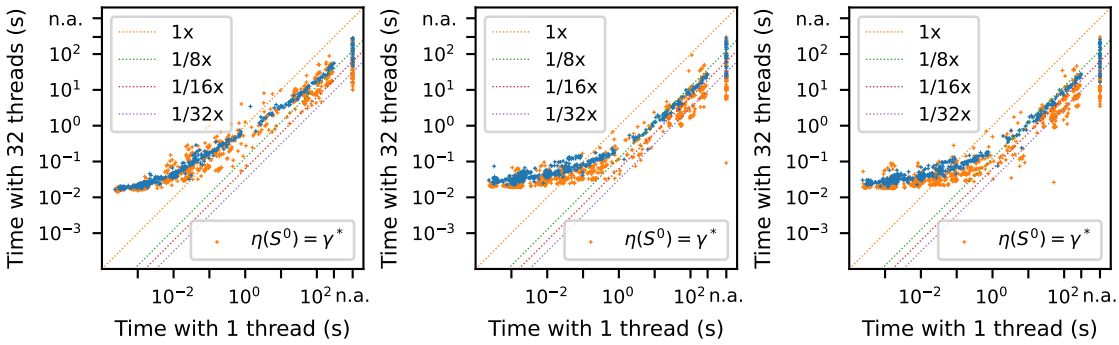
CABS to solve each instance in Figures 6.4–6.9. In these plots, each point represents one problem instance, the x-axis is the time taken by sequential CABS, and the y-axis is the time taken by a multi-thread solver. In all problem classes, the speedup tends to be larger as sequential CABS takes more time to solve. While CABS sometimes solves an instance in fewer than 0.01 seconds, the multi-thread solvers take at least 0.01 seconds to solve a problem instance, possibly due to the initialization of multiple threads.

For bin packing and SALBP-1, we confirm the tendency observed in Table 6.5. In bin packing, all multi-thread DIDP solvers show super linear speedups in a number of instances. At the same time, they show slowdowns in relatively difficult instances where CABS takes at least 1 second to solve. Moreover, as we mentioned in Section 6.3.1, the multi-thread solvers fail to solve some instances that are solved by sequential CABS. Therefore, while the multi-thread DIDP solvers, particularly CAHDBS1 and CAHDBS2, seem to benefit from the change in search behavior on average, they also suffer from it in some instances. In SALBP-1, CAHDBS1 and CAHDBS2 achieve super linear speedups in multiple instances, but they do not show slowdowns in difficult instances. For these observations, we hypothesize that the large diversity in the search time is due to the strong dual bound function used in the DyPDL models of bin packing and SALBP-1: in some instances of bin



(a) CASBS (b) CAHDBS1 (c) CAHDBS2

Figure 6.6: Comparison of sequential and 32-thread CABS in time to optimally solve each instance of bin packing. Unsolved instances are shown at 'n.a.'. '$\eta(S^0) = \gamma^*$' means instances where the dual bound value of the target state is equal to the optimal solution cost.

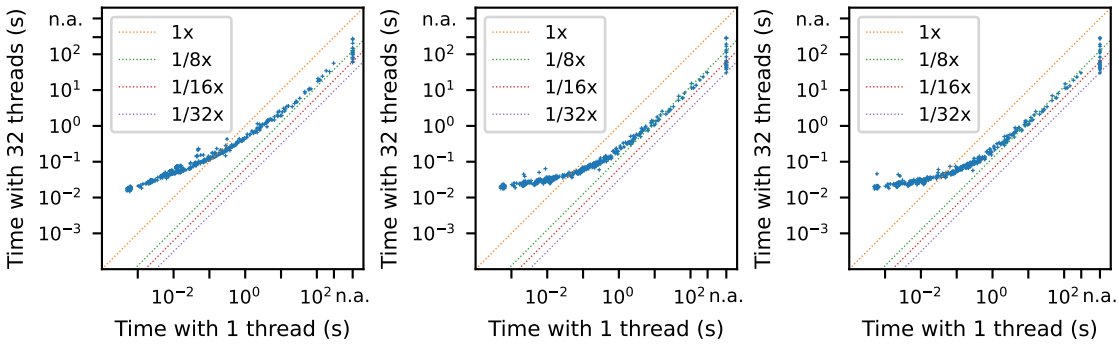


(a) CASBS (b) CAHDBS1 (c) CAHDBS2

Figure 6.7: Comparison of sequential and 32-thread CABS in time to optimally solve each instance of SALBP-1. Unsolved instances are shown at 'n.a.'. '$\eta(S^0) = \gamma^*$' means instances where the dual bound value of the target state is equal to the optimal solution cost.

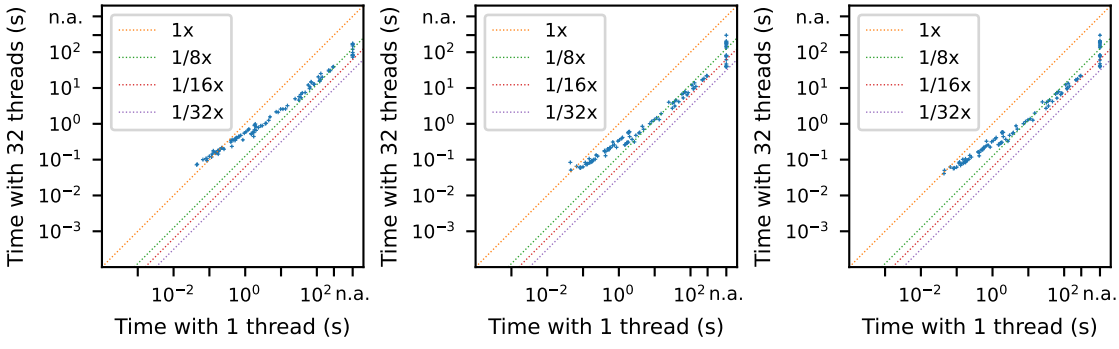
packing and SALBP-1, $\eta(S^0)$, the dual bound value of the target state, is equal to the optimal solution cost γ^*. In such a case, CABS proves the optimality as soon as it finds an optimal solution even if it discards some states due to the beam width. Therefore, the set of kept states, which can be changed by different tie-breaking in SBS or hash-based work distribution in HDBS, affects not only the solution quality but also the time to optimally solve an instance. To validate this hypothesis, for bin packing and SALBP-1, we plot instances with $\eta(S^0) = \gamma^*$ in a different color. In bin packing, a super linear speedup or slowdown occurs mostly in instances with $\eta(S^0) = \gamma^*$. The distribution of other instances is less diverse while there are several outliers. The tendency in SALBP-1 is similar.

We also compare the number of expansions in each instance of bin packing and SALBP-1 in Figures 6.10 and 6.11. The number of expansions by multi-thread CABS is correlated with that of sequential CABS, but the variance is large. In particular, in bin packing, multi-thread CABS sometimes expands thousands of times more states than sequential CABS and vice versa. However, if we focus on instances with $\eta(S^0) \neq \gamma^*$, the number of expansions by multi-thread CABS is close to that of sequential CABS in most cases. Therefore, we suspect that the speedup anomaly observed



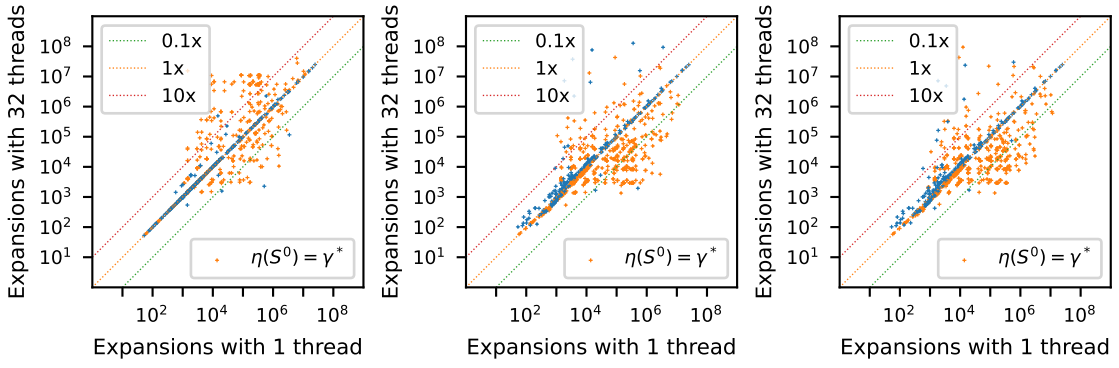
(a) CASBS (b) CAHDBS1 (c) CAHDBS2

Figure 6.8: Comparison of sequential and 32-thread CABS in time to optimally solve each instance of MOSP. Unsolved instances are shown at 'n.a.'.



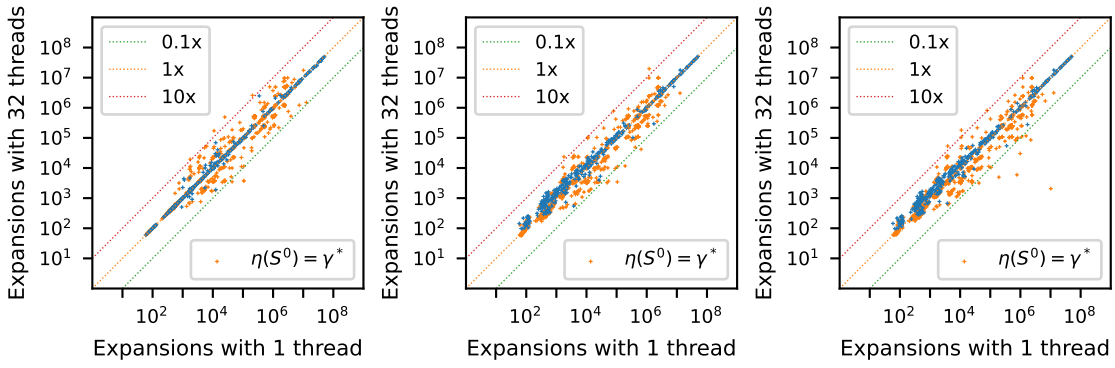
(a) CASBS (b) CAHDBS1 (c) CAHDBS2

Figure 6.9: Comparison of sequential and 32-thread CABS in time to optimally solve each instance of graph-clear. Unsolved instances are shown at 'n.a.'.



(a) CASBS (b) CAHDBS1 (c) CAHDBS2

Figure 6.10: Comparison of sequential and 32-thread CABS in the number of expanded states to optimally solve each instance of bin packing. '$\eta(S^0) = \gamma^*$' means instances where the dual bound value of the target state is equal to the optimal solution cost.



(a) CASBS (b) CAHDBS1 (c) CAHDBS2

Figure 6.11: Comparison of sequential and 32-thread CABS in the number of expanded states to optimally solve each instance of SALBP-1. '$\eta(S^0) = \gamma^*$' means instances where the dual bound value of the target state is equal to the optimal solution cost.

in bin packing and SALBP-1 is caused by the strong dual bound function, which makes finding an optimal solution sufficient to prove the optimality without exhaustive search. Indeed, in problems whose objective is to find a single feasible solution rather than optimization, such as satisfiability, constraint satisfaction, and satisfycing heuristic search, it is known that small perturbations in search can result in significantly different performance [179, 180, 82], and parallel algorithms sometimes suffer from it [272, 271]. While HDBS happens to benefit from it on average, previous work explicitly exploited this structure by using randomized algorithms in sequential and parallel solvers [181, 178, 83].

6.4 Related Work

To provide a deeper context for the work in this chapter, we review existing parallel state space search algorithms focusing on graph search algorithms, i.e., algorithms that try to find a path in a state transition graph while performing duplicate detection. As described in Section 6.1.2, there are mainly two types of environments for parallel algorithms: shared memory and distributed environments [331]. We introduce parallel state space algorithms developed for these two environments. While there are parallel search algorithms that use other environments, e.g., external memory search [111] and search on graphics processing units (GPUs) [120, 409, 452], we do not discuss such algorithms as they are not directly related to our approach. We also present parallel search algorithms that try to keep similar search behavior to their sequential counterparts to reduce search overhead, including those coming from speedup anomalies. Finally, we discuss future directions given the existing work. We summarize our parallel beam search algorithms and existing algorithms introduced in this section in Tables 6.6.

6.4.1 Parallel Search in Shared Memory Environments

We classify parallel state space search algorithms in shared memory environments into three approaches: parallel search with shared data structures, parallel structured duplicate detection, and parallel edge evaluation. In the first two approaches, multiple threads expand states in parallel. In

Table 6.6: Parallel state space search algorithms in the literature. 'DD' means duplicate detection, 'shared' means shared memory, 'TTP' means the traveling tournament problem, and 'alignment' means sequence alignment problems.

References	Paradigm	Algorithm	Open list	DD	Other features	Applications
SBS (Section 6.2.2)	shared	beam search	array	concurrent	layer synchronization parallel sort	DIDP
Vidal et al. [436]	shared	BFS	locked	locked	OpenMP	AI planning
El Baz et al. [122]	shared	BFS	locked	locked	OpenMP	AI planning
Zhang & Hansen [448]	shared	BFHS	distributed	concurrent	layer synchronization dynamic distribution	AI planning
Frohner et al. [148]	shared	beam search	array	concurrent	layer synchronization parallel counting sort	TTP
Kumar et al. [264]	shared	A*	concurrent	separate		TSP, VCP
Lock-free parallel A* [55]	shared	A*	lock-free	lock-free		pathfinding
Stivala et al. [408]	shared	memoization	-	lock-free		knapsack MOSP alignment
Zhou & Hansen [451]	shared	BFHS	PSDD	PSDD	layer synchronization	AI planning
PBNF [55]	shared	BFS	PSDD	PSDD		pathfinding puzzles AI planning
PA*SE [339]	shared	A*	locked	locked	no re-expansion	motion planning
ePA*SE [320, 443, 322]	shared	A*	locked	locked	no re-expansion parallel edge evaluation	motion planning
MPLP [321]	shared	A*	sequential	sequential	parallel edge evaluation	motion planning
PUHF [271, 395]	shared	GBFS	locked	lock-free	bounded search overhead	AI planning
HDBS (Section 6.2.3)	distributed	beam search	distributed	distributed	hash-based distribution async. message passing layer synchronization	DIDP
Evett et al. [127]	distributed	A*	distributed	distributed	hash-based distribution	puzzles
TDS [367]	distributed	IDA*	distributed	distributed	hash-based distribution async. message passing	puzzles
HDA* [253, 254, 235]	distributed	A*	distributed	distributed	hash-based distribution async. message passing abstract hashing	puzzles AI planning alignment pathfinding
Kumar et al. [264]	distributed	A*	distributed	separate	dynamic distribution	TSP, VCP
QE [110, 303]	distributed	A*	distributed	distributed	dynamic distribution local hashing	TSP
LG [272]	distributed	GBFS	distributed	distributed	hash-based distribution async. message passing	AI planning
Bergman et al. [34]	distributed	DD-B&B	centralized	separate	dynamic distribution	optimization

contrast, parallel edge evaluation uses multiple threads to expand a single state.

Parallel Search with Shared Data Structures

Typically, a state space search algorithm uses two data structures: a hash table for duplicate detection and an open list to store states to expand. One approach for parallelization in a shared memory environment is to use multiple threads to perform search while sharing the hash table and/or the open list. In doing so, we need to avoid contention, e.g., multiple threads mutating the same memory location at the same time. We review three mechanisms to avoid contention: mutually exclusive locks, concurrent data structures, and lock-free data structures.

Mutually Exclusive Locks A mutually exclusive lock ensures that only one thread executes a particular region of a program [331]. With mutually exclusive locks, a parallel algorithm can use the same data structures used in the corresponding sequential algorithm. For example, in OpenMP [70], only one thread can execute program lines inside the `critical` directive at the same time, and other threads need to wait until the thread finishes those lines. Vidal, Lucas, and Hamadi [436] and El Baz et al. [122] parallelized BFS using the `critical` directive for the open list and the hash table for duplicate detection.

Concurrent Data Structures Using a single lock can be too restrictive: it might be safe to allow different threads to access different regions of the same data structure. A concurrent data structure is designed so that multiple threads can access it simultaneously without contention. In Section 6.1.4, we explained concurrent hash tables for duplicate detection, which use multiple locks for different segments. These concurrent hash tables are used by parallel BFHS [448] and parallel beam search [148]. As explained in Section 6.1.3, the parallel BFHS algorithm uses distributed open lists, and the parallel beam search algorithm distributes states in the open list using the Thread module of Julia. In contrast, Kumar, Ramesh, and Rao [264] parallelized A* using a concurrent priority queue [323] for the shared open list. In this algorithm, each thread has its local hash table, and duplicate detection across multiple threads is not performed.

Lock-Free Data Structures Lock-free data structures [206] are concurrent data structures having a theoretical guarantee that at least one thread completes its operation in a finite number of steps, i.e., the whole system will make progress regardless of failures or delays of individual threads. Typically, a concurrent data structure uses atomic compare-and-swap operations instead of locks. Burns et al. [55] proposed lock-free parallel A*, which uses a lock-free priority queue [410] for the open list and a hash table implemented as an array of lock-free linked lists [189] for duplicate detection. Stivala et al. [408] parallelized a dynamic programming (DP) algorithm using a lock-free hash table [310]. In their algorithm, multiple threads perform recursion with memoization (Section 2.2.4) in parallel using a shared lock-free hash table to store states. While there is no open list, the recursive computation can be viewed as depth-first search. The work distribution is achieved by randomizing the order in which states are evaluated in each thread.

Parallel Structured Duplicate Detection

Sharing a single data structure by multiple threads requires synchronization, e.g., acquiring a mutually exclusive lock or performing an atomic operation, each time a thread accesses it. To reduce this overhead, parallel structured duplicate detection (PSDD) [451] uses multiple hash tables and avoids contention by exploiting problem structure. In PSDD, using problem-specific information, an abstract state transition graph is constructed, where nodes are abstracted states. An abstract state corresponds to a set of states, and each state is mapped to one abstracted state by an abstraction function. The abstract state transition graph has an edge from an abstract state A to B if there exists an edge from a state in A to a state in B in the original state transition graph. Since a state is uniquely mapped to an abstract state, for duplicate detection, checking states mapped to the same abstract state is sufficient. A set of states mapped to the same abstract state is called an $n$block, and one hash table is used for each $n$block. Multiple threads can expand states in different $n$blocks and perform duplicate detection in parallel if the generated successor states are mapped to different abstract states. PSDD was originally used to parallelize BFHS [451] with layer synchronization. In BFHS with PSDD, states in the current layer are partitioned into $n$blocks, and each thread acquires an $n$block and expands all states in it. To ensure that $n$blocks expanded by different threads in parallel do not interfere, the abstract state transition graph is shared with a lock, and the number of threads using each abstract state is maintained: an abstract state is used if an $n$block corresponding to that abstract state, its parent, or an abstract state having common successors is being expanded. Otherwise, the $n$block is free and can be expanded.

Burns et al. [55] proposed Parallel Best-NBlock-First (PBNF), a parallel BFS algorithm with PSDD. PBNF uses one priority queue as an open list for each $n$block. Similar to BFHS with PSDD, PBNF maintains the abstract state transition graph to identify free $n$blocks. A thread acquires a free $n$block that minimizes the best f-value of states in its open list, expands states from the open list in the best-first order, and releases the $n$block when the best f-value in the open list becomes worse than other free $n$blocks.

Parallel Edge Evaluation

In the algorithms presented above, multiple threads expand different states in parallel, and each thread sequentially generates all successor states of the expanded state. An alternative approach is to use multiple threads to generate successor states of a single state in parallel. Such algorithms are effective in problems where generating a successor state is expensive. Edge-Based Parallel A* for Slow Evaluations (ePA*SE) [320] stores pairs of a state and a transition in a shared open list instead of states. Each thread removes a pair from the open list and applies the transition to the state to generate a successor state. For duplicate detection, a single hash table is shared by all threads. ePA*SE is designed for robot motion planning, where evaluating the edge weight is expensive, and accessing the shared data structures is not a bottleneck [320]. Extensions of ePA*SE were developed by subsequent work [322, 443]. With a similar motivation to ePA*SE, Massively Parallel Lazy Planning (MPLP) [321] was developed for motion planning. Since evaluating the edge weight is expensive in motion planning, MPLP performs sequential BFS based on optimistic estimations of the edge weights. To eventually obtain the correct edge weights, MPLP evaluates the actual edge weights in parallel using multiple threads that are not used for the search.

6.4.2 Parallel Search in Distributed Environments

We classify distributed parallel state space search algorithms into two categories, focusing on load balancing mechanisms: hash-based work distribution and dynamic work distribution. In addition, we also present a parallelization of decision diagram-based branch-and-bound (Section 2.2.8), which can be considered a state space search algorithm.

Hash-Based Work Distribution

As described in Section 6.1.5, in hash-based work distribution [127], a state is uniquely assigned to its owner process based on the hash value. Each process performs search using a local open list and a local hash table for duplicate detection while sending generated states to their owner processes by message passing. If hash values are uniformly distributed, hash-based work distribution achieves good load balancing. Since the same state is always sent to the same process, it also achieves duplicate detection across different threads.

Hash-based work distribution was originally proposed by Evett et al. [127] to parallelize A*. In their algorithm, message passing is synchronous, i.e., each process waits until the sent message is received. In contrast, Romein et al. [367] introduced asynchronous message passing, where a process periodically receives a message from a message queue, and the sender process does not wait until the message is received. With this mechanism, Romein et al. [367] proposed transposition-driven scheduling (TDS), a parallelization of IDA* [258] with a fixed size hash table for duplicate detection

called a transposition table [353]. Kishimoto, Fukunaga, and Botea [253] proposed hash-distributed A* (HDA*), a parallel A* algorithm using hash-based work distribution and asynchronous message passing, and demonstrated that asynchronous message passing outperforms synchronous message passing.

Abstract Zobrist Hashing To achieve good load balancing, Romein et al. [367] and Kishimoto, Fukunaga, and Botea [253] used Zobrist hashing [453], which tends to uniformly distribute states to hash values. The drawback of such a hash function is the high frequency of communications: with k processes, $\frac{k-1}{k}$ generated states are sent to other processes on average. As we use more processes, the communication overhead increases. To reduce the number of states sent to other processes, Jinnai and Fukunaga [235] proposed Abstract Zobrist hashing, with which a successor state is more likely to be assigned the same hash value as its parent. Zobrist hashing takes the XOR of predefined random numbers for features of a state. For example, in the 15-puzzle, a state is represented by positions of fifteen tiles numbered from 1 to 15 in a 4×4 frame, so a feature can be a pair of a tile and its position in the frame. One random number is defined for each pair, and Zobrist hashing takes the XOR of the fifteen random numbers corresponding to the locations of the tiles in a state. In contrast, Abstract Zobrist hashing takes the XOR of random numbers for abstract features. In our example of the 15-puzzle, an abstract feature can be a row in which a tile is placed. In this way, Abstract Zobrist hashing still takes XOR of fifteen random numbers, but horizontally sliding a tile does not change the hash value. As in this example, if states tend to share abstract features with their successor states, generated states are more likely to be kept in the same process. With Abstract Zobrist hashing, communications become less frequent, but the state distribution becomes less uniform, so a good abstraction should balance this tradeoff. Jinnai and Fukunaga [235] manually designed abstract features for the sliding-tile puzzles and multiple sequence alignment. For classical AI planning, they developed methods to automatically construct abstract features given a problem instance. In their evaluation, HDA* with Abstract Zobrist hashing outperforms HDA* with Zobrist hashing in all problem classes.

Dynamic Work Distribution

While hash-distributed work distribution with asynchronous message passing is state of the art, early work investigated dynamic work distribution for distributed parallel A*. In such an algorithm, similar to hash-based work distribution, each process has an open list and independently performs search while dynamically distributing states. Unlike hash-based work distribution, dynamic work distribution requires a separate mechanism to detect duplicate states across different processes.

Kumar, Ramesh, and Rao [264] investigated three load balancing strategies for distributed parallel A*: random, ring, and blackboard strategies. In the random strategy, each process sends some of the generated states to randomly assigned processes. In the ring strategy, assuming a ring network topology where each process has neighbors, processes that can be communicated with with small latency, each process sends some of the generated states to its neighbors. The blackboard strategy tries to equalize the lowest f-value in the open list in each process. As discussed later in Section 6.4.3, if the lowest f-value in a process is higher than other processes, the process may expand states that are never expanded by sequential A*. The blackboard strategy is designed to reduce such unnecessary expansions. The blackboard strategy maintains a shared BLACKBOARD,

which stores states having low f-values. If a process has states whose f-values are significantly lower than the states in the BLACKBOARD, the process sends some of them to the BLACKBOARD. If a process has states whose f-values are significantly higher than the states in the BLACKBOARD, the process receives some states from the BLACKBOARD. For duplicate detection, Kumar, Ramesh, and Rao [264] used one hash table for each process, but duplicate states across different processes are not removed. In their evaluation using the traveling salesperson problem (TSP) and the vertex cover problem (VCP), the blackboard strategy outperforms the other two.

Dutt and Mahapatra [110] proposed quality equalizing (QE) balancing strategies for distributed parallel A* in the hypercube architecture, where each process has neighbors, similar to the ring topology. With the same motivation as the blackboard strategy, the QE strategy tries to equalize the lowest f-value in the open list in each process via the exchange of information about the open lists with neighboring processes. A process requests to receive states with lower f-values from its neighbors if the lowest f-value in the local open list is significantly higher than that of the neighbors. The same authors (Mahapatra and Dutt [303], in a different order) combined QE with duplicate detection of hash-based work distribution. In their algorithm called global hashing (GOHA), a generated state is first sent to its owner process for duplicate detection, but the state can be forwarded to another process by QE. To reduce communication overhead, Mahapatra and Dutt also proposed local hashing (LOHA), where the distributed hash table is partitioned by layers of states: each process has a hash table storing states in the same layer, and neighbors have hash tables for the adjacent layers. Therefore, each process sends generated successor states to its neighbor processes. This strategy is sufficient to detect duplicates in problems such as TSP, where duplicate states are always in the same layer. In their evaluation using TSP, LOHA outperforms GOHA.

Parallel Decision Diagram-Based Branch-and-Bound

In the operations research (OR) community, Bergman et al. [34] parallelized decision diagram-based branch-and-bound (DD-B&B) [35] in a distributed memory environment. As we discussed in Section 4.1, DD-B&B can be viewed as a state space search algorithm. In each iteration, DD-B&B selects a node from a relaxed decision diagram (DD) and constructs restricted and relaxed DDs to obtain primal and dual bounds. In the parallelized version, a single master process manages candidate nodes and dynamically distributes them to worker processes. Worker processes construct DDs from different nodes independently in parallel and send new DD nodes to the master process. This procedure can be viewed as a distributed parallel state space search algorithm with dynamic load balancing and without duplicate detection across different processes.

6.4.3 Reducing Search Overhead

As we empirically observed in Section 6.3.2, a parallel state space search algorithm may expand more or fewer states than its sequential counterpart, resulting in a slowdown or super linear speedup, since parallelization may change the search behavior. This phenomenon is known as speedup anomaly in parallel branch-and-bound [278]. For parallel A*, a super linear speedup or a significant slowdown was rarely reported, but an increase in the number of expansions, sometimes called search overhead, was observed by previous work [264, 110, 55, 253, 235]. The search overhead of parallel A* comes from three sources: expansions in the last f-layer, unnecessary expansions, and re-expansions. Given

an admissible heuristic function h, where $h(S)$ is a lower bound on the optimal path cost from S, A* never expands states having higher f-values than the optimal cost, and the first found solution is optimal [190, 98]. For states in the last f-layer, states whose f-values are the same as the optimal cost, the number of expansions depends on the tie-breaking strategy. Parallelization may change the tie-breaking, which may result in a different number of expansions in the last f-layer, similar to what we observed with parallel CABS in Section 6.3.2.

Moreover, due to the change in the expansion order, parallel A* can expand states whose f-values are higher than the optimal solution cost, which are never expanded by sequential A* with any tie-breaking strategy. The first found solution may no longer be the optimal solution, and parallel A* needs to wait until states having smaller f-values than the found solution cost are eliminated from the open list [264, 110, 55, 253, 235]. If the heuristic function is consistent, i.e., $h(S) \leq w(S, S') + h(S')$ where $w(S, S')$ is the weight of an edge (S, S') in the state transition graph, sequential A* never re-expands already expanded states. In contrast, due to the change of the expansion order, parallel A* may later find a better path to an already expanded state and re-expand it. As mentioned in Section 6.4.2, to reduce the search overhead, distributed parallel A* algorithms try to simulate the best-first expansion order by uniformly distributing states [253] or dynamically distributing states with low f-values [264, 110]. In a shared memory environment, Phillips, Likhachev, and Koenig [339] proposed Parallel A* for Slow Expansions (PA*SE), which never re-expands a state given a consistent heuristic. This property is guaranteed by a mechanism to make sure that an optimal path to reach a state is found before expanding it. Since this check is expensive, PA*SE is suited to problems where expanding a state is expensive, such as motion planning [339]. ePA*SE [320], the parallel A* algorithm for motion planning presented in Section 6.4.1, is an extension of PA*SE.

Kuroiwa and Fukunaga [272] evaluated HDGBFS, an adaptation of HDA* to greedy best-first search (GBFS) [107], which expands a state minimizing the h-value. They observed a large search overhead of HDGBFS compared to sequential GBFS in some problem instances and proposed Locally Greedy HDGBFS (LG), which reduces the search overhead using an ad hoc mechanism. In subsequent work [271], Kuroiwa and Fukunaga proved that under some assumptions, there is no theoretical upper bound on the search overhead of parallel GBFS algorithms that perform duplicate detection in parallel. They proposed Parallel Under High-water mark First (PUHF), which expands only states that could be expanded by sequential GBFS with some tie-breaking strategy. Similar to PA*SE, this property is guaranteed by a mechanism to check a state before expanding it using a sufficient criterion. Shimoda and Fukunaga [395] improved PUHF by using more permissive criteria.

6.4.4 Discussion

We introduced parallel state space search algorithms designed for different environments with different approaches. In this section, based on the existing work, we discuss future directions for parallel DIDP solvers.

In the current implementation of HDBS, we use Fx Hash in Rust, which uniformly distributes states to hash values. To reduce communication overhead in hash-based work distribution, Jinnai and Fukunaga [235] used a hash function that tends to assign the same hash value of a state to its successor states (Section 6.1.5). Developing such a hash function for DyPDL is a promising direction to improve HDBS.

Jinnai and Fukunaga [235] compared HDA* [253, 235], which uses hash-based work distribution,

and PBNF [55], which uses PSDD [451] (Section 6.4.1), in a shared memory environment. In their evaluation, HDA* with Zobrist hashing or Abstract Zobrist hashing performs better than PBNF in difficult instances of the sliding-tile puzzle. In contrast, PBNF is better than HDA* in grid pathfinding. Given this result, as an alternative to hash-based work distribution, PSDD is worth considering for parallel DIDP solvers in a shared memory environment.

While previous work in parallel BFS investigated mechanisms to achieve a similar expansion order to sequential BFS (Section 6.4.3), HDBS benefits from perturbations in the set of states kept in the open list on average. As in other fields [181, 178, 83], explicitly exploiting such perturbations in parallel and sequential DIDP solvers is an interesting direction.

As presented in Table 6.6, some of the introduced parallel state space search algorithms were used for AI planning. However, they have not been integrated with the mainstream AI planners such as Fast Downward [203][11] and ENHSP [379].[12] In contrast, commercial MIP and CP solvers such as Gurobi and CP Optimizer natively support multi-threading, as used in Section 6.3. In DIDP, following MIP and CP, parallel beam search algorithms are integrated into the release version of didp-rs. Since didp-rs is open-source, a researcher can implement new algorithms based on it, which possibly boosts research in parallel state space search.

6.5 Summary

We proposed parallel beam search algorithms and developed multi-thread domain-independent dynamic programming (DIDP) solvers by using them in complete anytime beam search (CABS). We investigated two ideas used to parallelize heuristic search algorithms in the AI community: concurrent data structures and hash-based work distribution. In our experimental evaluation, hash-distributed beam search (HDBS), which uses hash-based work distribution, outperforms shared beam search (SBS), which uses a concurrent data structure. Our multi-thread solvers based on HDBS achieve significant speedup and performance improvement over sequential CABS on average. Moreover, the multi-thread DIDP solvers outperform commercial multi-thread MIP and CP solvers in four out of six problem classes.

In some problem instances, we observed super linear speedups or slowdowns of the multi-thread solvers. This phenomenon seems to be caused by the strong dual bound function, which makes perturbations in the set of states kept by beam search lead to a significant difference in performance. While HDBS happens to benefit from it on average, explicitly exploiting this characteristic in sequential and parallel solvers might be beneficial.

A potential bottleneck of HDBS is frequent communications between threads. In parallel A* with hash-based work distribution, such communication overhead is reduced by using abstract hashing [235], which tends to assign the same process to a state and its successor states by exploiting problem structures. Developing abstract hashing for DyPDL is a promising direction to improve HDBS. As an alternative approach, parallel structured duplicate detection (PSDD) [451] may also be useful.

[11] `https://www.fast-downward.org/`
[12] `https://sites.google.com/view/enhsp/`

Chapter 7

Concluding Remarks

In this dissertation, we developed domain-independent dynamic programming (DIDP). DIDP enables a user to declaratively model and solve combinatorial optimization using dynamic programming (DP), which has been used as a problem-specific solving method in previous work. DIDP is domain independent in that it requires only a declarative mathematical model as input. Although such declarative problem-solving approaches for combinatorial optimization were achieved by existing model-based paradigms used in operations research (OR), they are based on constraint-based problem representations. In contrast, DIDP is a model-based paradigm using a state-based problem representation of DP. Our DIDP solvers use heuristic search, a class of algorithms studied in artificial intelligence (AI) to solve state-based problems such as AI planning. We demonstrated that DIDP outperforms existing model-based paradigms, mixed-integer programming (MIP) and constraint programming (CP), in multiple combinatorial optimization problem classes, which validates our thesis statement that DP can be used as a practical model-based paradigm for combinatorial optimization.

In Chapter 3, we developed the modeling formalism, Dynamic Programming Description Language (DyPDL), and its practical modeling language, YAML-DyPDL. DyPDL and YAML-DyPDL are inspired by modeling formalisms and languages for AI planning, but they also follow the approach of OR, which allows a user to investigate better optimization models by incorporating redundant information.

In Chapter 4, we developed general-purpose DIDP solvers using heuristic search algorithms. Our solvers exploit redundant information given in a DyPDL model. In addition, they meet the standard of general-purpose solvers for MIP and CP: they continuously improve the solution quality, eventually solve the problem optimally, and provide the optimality gap. In particular, complete anytime beam search (CABS), which iteratively executes beam search, achieved the best performance among the DIDP solvers.

In Chapter 5, we developed a DIDP solver using large neighborhood search (LNS), an algorithmic framework widely used in MIP and CP. We proposed large neighborhood beam search (LNBS), which combines LNS and beam search, and iteratively improves a partial path in the current solution path. Our DIDP solver based on LNBS uses multi-armed bandits to adaptively select the length of a partial path to improve.

In Chapter 6, we developed multi-thread DIDP solvers using parallel beam search algorithms. We

considered two approaches: shared beam search (SBS) based on a shared data structure and parallel sorting, and hash-distributed beam search (HDBS) based on hash-based work distribution with asynchronous message passing. Our multi-thread DIDP solvers based on HDBS achieve significant performance improvement and speedup over the sequential CABS solver.

7.1 Summary of Contributions

This dissertation contributes to two fields: combinatorial optimization, for which we developed solving methods, and heuristic search, with which we solved problems that heuristic search had not been previously used to solve. We summarize the contributions in the following lists.

7.1.1 Contributions to Combinatorial Optimization

1. We developed domain-independent dynamic programming (DIDP), a model-based paradigm for combinatorial optimization.
2. We developed Dynamic Programming Description Language (DyPDL), a declarative modeling language for DP designed for combinatorial optimization.
3. We developed didp-rs, a new open-source model-based solver for combinatorial optimization using DIDP. In terms of modeling, it provides a command line interface that takes DyPDL files as input in addition to a Python modeling library. In terms of solving, it provides exact, anytime, and multi-thread solvers based on heuristic search.
4. We developed the DyPDL models by adapting existing DP approaches to eleven combinatorial optimization problem classes. We also implemented the state-of-the-art MIP and CP models for these problem classes and developed improved CP models for the simple assembly line balancing problem to minimize the number of stations (SALBP-1) and talent scheduling.
5. We applied DP to the eleven combinatorial optimization problem classes through DIDP and compared its performance with state-of-the-art MIP and CP models solved by commercial MIP and CP solvers.
6. We updated the best-known solutions for two instances of the traveling salesperson problem with time windows (TSPTW).[1] We also closed seven instances of the multi-commodity pickup and delivery traveling salesperson problem (m-PDTSP) [270].

7.1.2 Contributions to Heuristic Search

1. We developed a generic heuristic search procedure to solve a class of DyPDL models exploiting dominance between states and a dual bound function. We provide theoretical proofs for its correctness, completeness, and optimality.
2. We developed a new open-source software platform (didp-rs) in which a researcher can implement and parallelize heuristic search algorithms.

[1] `https://lopez-ibanez.eu/tsptw-instances#traveltime`

3. We provide a benchmark set for heuristic search algorithms composed of eleven combinatorial optimization problem classes in the form of the DyPDL models.
4. We empirically evaluated existing heuristic search algorithms using the eleven combinatorial optimization problems.
5. We proposed large neighborhood beam search (LNBS), a new heuristic search algorithm using large neighborhood search (LNS) and multi-armed bandits.
6. We proposed shared-memory and distributed parallel beam search algorithms, which scale up to at least 32 threads.

7.2 Future Work

We present future research directions for DIDP. One of the future directions is to apply DIDP to more diverse problems. To widen the application fields or achieve better performance, we may need to extend the modeling formalism and language. The current DIDP solvers also have significant room for improvement.

7.2.1 Applications

Since DIDP is a model-based paradigm, it has great potential for applications. In particular, we consider three directions: modeling diverse optimization problems, hybridization with other paradigms, and an application to AI planning. In addition, investigating characteristics of models that can be efficiently solved by DIDP is also important.

Modeling Diverse Optimization Problems

The eleven combinatorial optimization problem classes used in this dissertation are studied in the OR literature: as mentioned in Section 4.3.2, MIP and CP models were previously developed for most of them, and the benchmark instances of problem classes except for graph-clear were previously published. Modeling and solving other well-known problems in OR can demonstrate the significance of DIDP. However, to investigate the flexibility and limitations of DIDP, modeling and solving more complex optimization problems, such as those arising from industry, is necessary. The work by Golestanian et al. [177] is one example of the latter. They formulated a pickup and delivery problem using an aircraft with a complicated capacity constraint: the cargo capacity can be increased or decreased during a route by removing or adding passenger seats. The problem is motivated by real-world air service for remote communities in Canada's north. Using didp-rs, they succeeded in formulating the problem as a DyPDL model and demonstrated superior performance to MIP and CP. To investigate the limitations of the current DIDP, we need to investigate more diverse and realistic problems.

Hybridization with Other Paradigms

Another potential application in combinatorial optimization is to combine DIDP with other model-based paradigms. In particular, we may use DIDP to solve the subproblems of a decomposed

optimization model in another paradigm or decompose a DP model and use another paradigm to solve the subproblems. For example, in column generation [101], a linear programming (LP) model is decomposed into a master problem and pricing subproblems. The subproblems are sometimes solved by labeling algorithms (Section 2.2.5), which are based on DP [103, 16, 299, 298, 373]. By using DIDP in column generation, a user just needs to declaratively define pricing subproblems rather than implementing problem-specific DP algorithms. To solve pricing problems efficiently, we may need to develop DIDP solvers that exploit techniques used in problem-specific labeling algorithms. It may also be possible to combine DIDP with other decomposition approaches such as Benders decomposition [30, 215].

Application to AI Planning

DIDP can solve numeric AI planning: we showed that a numeric planning problem can be automatically compiled into a DyPDL model in Section 3.1.1. Moreover, we may manually develop better DyPDL models for planning problems, possibly by incorporating redundant information. As discussed in Section 2.3.3, such an approach is different from the standard convention in the AI planning community, which uses the same model for a planning problem to compare the performance of different domain-independent planners. However, this approach is useful to solve application problems. Indeed, developing better planning models has been studied in CP [12] and logic programming [19].

Investigation of Model Characteristics

In this dissertation, we showed that DIDP is better than MIP and CP in some problem classes but worse in others. The characteristics of problems and their models that make differences in different paradigms is an interesting research question. Answering this question would help practitioners in selecting tools to tackle their problems. Moreover, through this investigation, we may find better modeling strategies for DIDP and other paradigms or improved solving algorithms to address the weaknesses of a particular framework. As described in Section 4.3.5, we have an insight on this question: the DIDP solvers perform better than the MIP and CP solvers in problems where constraints such as time window constraints and precedence constraints can be used to limit the size of the state transition graph. A deeper analysis of problem characteristics, e.g., evaluating the solvers using synthetic problems with varying tightness of constraints, is one of our future directions. Similar analyses were previously conducted in multiple problems including NP-complete problems such as the traveling salesperson problem (TSP) [71], satisfiability (SAT) [387], constraint satisfaction problems (CSPs) [404, 345], classical planning [57, 359], and satisficing heuristic search [84].

7.2.2 Improvements in Modeling

DyPDL and YAML-DyPDL have significant room for improvement. In particular, we discuss the limitations of modeling dominance in DyPDL and YAML-DyPDL and dual bound functions in YAML-DyPDL. We also consider the possibility of extending DyPDL to stochastic DP.

Refinement of the Dominance Definition in DyPDL

In the theoretical formalism, DyPDL, a state dominates another if it leads to a better or equal solution with a smaller or equal number of transitions (Definition 16 in Section 3.1.2). The latter

requirement, a smaller or equal number of transitions, is necessary to ensure that a state does not dominate its successors and descendants. However, this requirement might be too restrictive: a state S may dominate another state S' if S leads to a better but longer solution than S' as long as S' is not reached from S by that solution. Refining the definition of dominance considering such a condition is desirable for exploiting dominance in a broader class of DyPDL models.

Extensions of Dominance in YAML-DyPDL

In YAML-DyPDL, dominance is defined by specifying preference, less is better or less is worse, for element and numeric variables (Definition 28 in Appendix A.1). We may want to define dominance based on set variables, e.g., one state S dominates another state S' if the value of the set variable in S is a superset or subset of that of S'. Moreover, similar to an approach used in a decision diagram (DD) solver [89] (Section 2.4.4), allowing a user to define dominance more flexibly, e.g., as a function that compares two states and returns one of them, is worth considering.

Improved Modeling of Dual Bound Functions in YAML-DyPDL

In terms of the dual bound, the current DyPDL models for the orienteering problem with time windows (OPTW) in Section 3.3.3 and the multi-dimensional knapsack problem (MDKP) in Section 3.3.4 are not ideal. The dual bound functions in these models are based on relaxing the problems to the 0-1 knapsack problem, where we select items included so that their total weight does not exceed the capacity of the knapsack while maximizing the total profit. The Dantzig upper bound [96] computes the upper bound on the maximum profit as follows: it sorts items by descending order of the profit per weight, includes as many items as possible following the order, and fractionally includes the item that does not fit in the remaining capacity. In our YAML-DyPDL files, we used relaxations of the Dantzig upper bound because we were unable to efficiently represent it: to model the Dantzig upper bound, we need to use one 'if-then-else' expression to check if including the first j items in the knapsack is possible for each $j = 1, ..., n$, where n is the number of items, resulting in a large expression tree. Improving the expressiveness and flexibility of YAML-DyPDL is necessary to address such an issue. Furthermore, it would be beneficial if a user could define dual bound functions as optimization models such as LP and MIP models.

Extension to Stochastic Dynamic Programming

In this dissertation, we focused on deterministic DP. Stochastic DP is another major branch of DP, which is typically represented by a Markov decision process (MDP) [27, 219] (Section 2.1.2). Extending DyPDL to model stochastic DP or MDPs is possible for future work. In this direction, we need to investigate the value of using DyPDL compared to the existing modeling languages for MDPs such as PPDDL [444] and Relational Dynamic Influence Diagram Langauge (RDDL) [375] developed in the AI planning community.

7.2.3 Improvements in Solving

In this section, we present future directions to improve the performance of DIDP solvers. We discuss six topics: memory-efficient solvers, dual bound and heuristic functions, DD-based branch-and-bound, primal heuristics, randomization, and massively parallel solvers.

Development of Better Memory-Efficient Solvers

In Section 4.3.5, we showed that CABS outperforms other DIDP solvers due to its memory efficiency: other solvers reach the 8 GB memory limit before the 30-minute time limit. However, CABS is less time efficient, e.g., it takes more time to solve the same number of problems as other solvers when memory is not a constraining resource. Developing faster memory-efficient DIDP solvers is an important direction for future work. For this direction, using memory-efficient heuristic search algorithms studied in the AI community [117, 118, 119] is a promising direction. Indeed, CABS uses one such technique, layered duplicate detection [450]. As another example, a transposition table, which stores a fixed number of states in memory [353], might be useful.

Development of Dual Bound and Heuristic Functions

The current DIDP solvers use the dual bound function defined in a DyPDL model for two purposes: pruning states that do not lead to better solutions and guiding the search as a heuristic function. As shown in Section 4.3.7, using a dual bound function significantly improves the performance of the DIDP solvers. While the current DIDP solvers use the dual bound function defined in a DyPDL model, a user may not always define a good dual bound function in a model. Moreover, a user may not define a dual bound function at all since it is not required. Therefore, we should develop methods to automatically obtain dual bound and heuristic functions, which are not necessarily the same. We discuss three topics related to this issue: domain-independent dual bound functions, domain-independent heuristic functions, and learning heuristic functions.

Domain-Independent Dual Bound Functions To automatically obtain dual bound functions, we are considering two approaches based on different fields: OR and AI. In OR, state space relaxation obtains a dual bound by solving a relaxed version of the DP formulation [76] (Section 2.2.7). While state space relaxation methods were used for specific combinatorial optimization problems in previous work [76, 1, 144, 46, 357, 355, 183, 16, 17, 365], we may develop dual bound functions for DyPDL by generalizing them. In AI, abstraction, an idea similar to state space relaxation as pointed out by Holte and Fan [214], is used to obtain admissible heuristic functions, which returns a lower bound on the shortest path cost from a state. In particular, methods to automatically construct such heuristic functions have been studied for domain-independent AI planning [112, 193, 204, 386]. In addition, admissible heuristic functions computed by LP and MIP have also been proposed [85, 47, 344, 231, 378, 340, 380, 276]. Adapting these approaches to DyPDL is another option to develop dual bound functions for DIDP.

Domain-Independent Heuristic Functions While a dual bound function can be used as a heuristic function, as discussed in Section 4.1.4, they are not necessarily the same. A dual bound function is admissible, but inadmissible heuristic functions can be beneficial to quickly find good solutions. To automatically obtain inadmissible heuristic functions, we may use approaches inspired by domain-independent AI planning, similar to the above-mentioned approach for dual bound functions. For example, as mentioned in Section 2.3.3, inadmissible heuristic functions for AI planning can be obtained from the delete relaxation [48], which ignores the effects of actions that negatively change a state. While the optimal solution cost for the delete relaxation is admissible, since optimally solving it is NP-hard [58], some heuristic functions inadmissibly estimate the optimal cost

using polynomial-time algorithms [48, 194, 213, 250]. Although generalizing the delete relaxation to DyPDL is not trivial, its fundamental idea of relaxing the part of a model and approximating the solution cost might be useful for DIDP. As an alternative approach, we may use restricted DDs (Section 2.2.8), which give an upper bound on the shortest path cost, to obtain inadmissible heuristic functions.

Learning Heuristic Functions Using machine learning to obtain a heuristic function is also a possible direction. As mentioned in Section 2.3.2, Cappart et al. [62] solved CP models based on DP formulations using a CP solver whose value ordering heuristic is obtained by reinforcement learning with the same DP formulations as the CP models. This framework can be combined with DIDP by replacing the CP solver with a DIDP solver and using reinforcement learning to obtain a heuristic function. Besides this approach, learning heuristic functions has been studied for state space search problems such as combinatorial puzzles [4, 329, 131] and domain-independent AI planning [393, 134, 135, 445, 133, 132, 200, 72].

Integration with Decision Diagram-Based Branch-and-Bound

As described in Sections 2.2.8 and 2.4.4, DD-based branch-and-bound can be used to solve a DP model, and a general-purpose DD solver, ddo, has been developed [171]. Using DD-based branch-and-bound as a DIDP solver is an interesting direction for future work. To use DD-based branch-and-bound, we need a merging operator to create relaxed DDs. Ddo requires a user to provide a merging operator as a part of the model. If we follow this approach, we need to extend DyPDL so that a user can define a merging operator in a model. An alternative approach is to develop a method to automatically create a merging operator, similar to the above-mentioned ideas of automatically obtaining dual bound and heuristic functions. In either approach, a merging operator can be used not only for DD-based branch-and-bound but also for dual bound functions since a relaxed DD provides a dual bound. Indeed, relaxed DDs have been used to obtain an admissible heuristic function in classical planning [66].

Development of Efficient Primal Heuristics

In Chapter 5, we proposed LNBS as a primal heuristic, a method to obtain high-quality solutions quickly. One approach to developing a better primal heuristic is to improve LNBS. While LNBS outperforms CABS in six out of the eleven problem classes, CABS is better in the other five problem classes. As discussed in Section 5.3.5, the experimental result suggests that LNBS tends to perform better than CABS when a heuristic function is not informative. Given this observation, for example, developing a mechanism to detect the informativeness of a heuristic function and control the behavior of LNBS might be useful. Another possible approach is to develop different algorithms from LNBS. For example, as in constraint-based local search (CBLS) [431] (Section 2.3.2), integrating local search in DIDP might be possible. When we have multiple primal heuristics, as in MIP [75] (Section 5.5.2), selecting one of them using multi-armed bandits is worth considering.

Exploitation of Randomization

In Section 6.3.2, we observed that our multi-thread DIDP solvers sometimes achieve super linear speedups, benefitting from perturbations in the search behavior caused by parallelization. According to our experimental result, this behavior seems to be caused by a tight dual bound function, with which a solver proves optimality as soon as it finds an optimal solution. A similar phenomenon, i.e., large performance differences caused by small perturbations, is observed in SAT [179], CSPs [180], and satisficing heuristic search [82], whose objective is to find a single feasible solution. In these problems, solvers exploiting this phenomenon, e.g., by randomly restarting an algorithm with different parameters, were developed [181, 178, 83]. We may develop sequential and parallel DIDP solvers using such approaches.

Development of Massively Parallel Solvers

One of our parallel beam search algorithms, HDBS, is based on message passing. Therefore, it can be implemented in a distributed environment, where a parallel algorithm uses multiple processes that exchange data through network communications. Developing massively parallel optimization solvers that work in a large-scale distributed environment is beneficial to optimally solving difficult problem instances. For example, in previous work, an open-source MIP solver was parallelized with 80,000 processors [396, 397], a commercial MIP solver was parallelized with 43,344 processors [398], and a heuristic search algorithm was parallelized with 2,400 processors [253].

Massively parallel DIDP solvers are not necessarily restricted to beam search. The advantage of the DIDP solvers based on beam search is their memory efficiency as observed in Section 4.3.5. However, with a large amount of memory available in a distributed environment, heuristic search algorithms that are less memory-efficient but more time-efficient can be better choices. If we use hash-based work distribution (Section 6.1.5) to parallelize a heuristic search algorithm, abstract hashing, which reduces communications between processes by exploiting problem-specific structures [235] (Section 6.4.2), would be useful to achieve higher scalability.

Appendix A

Details of YAML-DyPDL

We present the syntax of YAML-DyPDL and the YAML-DyPDL domain files for the eleven combinatorial optimization problem classes for which we defined models in Chapter 3.

A.1 Syntax of YAML-DyPDL

In YAML-DyPDL files, the built-in types in YAML (i.e., maps, lists, strings, integers, floating-point numbers, and booleans) are used. Expressions and conditions are defined by strings. We summarize the notation for expressions in Table A.1 and explain their syntax in Section A.1.2. We divide numeric expressions and cost expressions into integer and continuous expressions, which should be processed using integers and floating-point numbers, respectively.

Table A.1: Notation for expressions in YAML-DyPDL.

Notation	Expression
$\langle eexpr \rangle$	element expression
$\langle sexpr \rangle$	set expression
$\langle iexpr \rangle$	integer expression
$\langle cexpr \rangle$	continuous expression
$\langle cond \rangle$	condition
$\langle icexpr \rangle$	integer cost expression
$\langle ccexpr \rangle$	continuous cost expression

In YAML-DyPDL, one problem instance is defined by two files, a domain file and a problem file. A domain file is a map that has keys listed in Table A.2. The value of the key cost_type defines whether the cost expression, the cost of a base case, and dual bound functions should be integer or continuous expressions. It is assumed to be integer by default. The value of the key objects is a list of strings, each of which is the name of an object type. Given the number of objects, n, which is defined in a problem file, we can use a set of objects indexed from 0 to $n-1$. For the name of the object type (or other entities described later), the names of the functions used in the syntax of expressions (Figure A.1) should be avoided. The key state_variables is required, and its value is a list of maps defining a state variable. We denote such a map by $\{variable\}$ and describe the details in Table A.3. The value of the key table is a list of maps defining a table of constants, denoted by

{*table*} and described in Table A.4. The key transitions is required, and its value is a list of maps defining a transition, denoted by {*transition*} and described in Table A.6. The key base_case is required, and its value is a list defining base cases. Each element of the list is a map denoted by {*base case*} described in Table A.9 or a list of conditions. If a list of conditions is given, the cost of the base case is assumed to be 0. Transitions and base cases can be defined in the problem file, and in such a case, they are optional in the domain file. The value of the key constraints is a list defining state constraints. Each element of the list is a map denoted by {*forall*} or a condition. The value of the key dual_bounds is a list of numeric expressions defining dual bound functions. If cost_type is integer (continuous), it must be an integer (a continuous) expression. The value of the key reduce is either min or max, specifying minimization or maximization. Minimization is assumed by default.

Table A.2: Keys of a domain file.

Key	Value	Requirement
cost_type	'integer' or 'continuous'	optional, 'integer' by default
objects	list of strings	optional
state_variables	list of {*variable*}	required
tables	list of {*table*}	optional
transitions	list of {*transition*}	required if not defined in the problem file
base_cases	list of {*base case*} and lists of ⟨*cond*⟩	required if not defined in the problem file
constraints	list of {*forall*} and ⟨*cond*⟩	optional
dual_bounds	list of ⟨*iexpr*⟩ or ⟨*cexpr*⟩	optional
reduce	'min' or 'max'	optional, 'min' by default

In Table A.3, we show the keys of {*variable*}. The key name defines the name, which is required. The key type defines the type of the state variable. For type, by using integer and continuous, YAML-DyPDL differentiates integer and continuous variables in numeric variables. The key object is required for element and set variables, and its value must be one of the names of the object types defined with the key objects. Element, integer, and continuous variables are resource variables if preference is defined. Resource variables are used to define an approximate dominance relation as explained in Section A.1.1.

Table A.3: Keys of {*variable*}.

Key	Value	Requirement
name	string	required
type	'element', 'set', 'integer', or 'continuous'	required
object	string, the name of a defined object	required for element and set variables
preference	'greater' or 'less'	optional for element and numeric variables

In Table A.4, we show the keys of {*table*}. The key name, which defines the name, and the key type, which defines the type, are required. The table is an element table if type: element, a set table if type: set, an integer table if type: integer, a continuous table if type: continuous, and a boolean table if type: bool. If the key args is not defined, the table is 0-dimensional and defines one constant. Otherwise, the value of the key args must be a list of the names of the object types defined with the key objects. If there are n elements in the list, the table is n-dimensional. An n-dimensional table associates an n-tuple of the indices of objects with a constant. The key default defines the default value of the table, whose value is a constant depending on the type as shown in Table A.5.

For element, integer, and continuous, the value is assumed to be 0 if the key is omitted. The value is assumed to be false for bool and an empty list for set. For a set table, the key object must be defined, whose value is the name of an object type defined with objects. Each value in the set table is restricted to be in 2^N where $N = \{0, ..., n-1\}$, and n is the number of the associated objects.

Table A.4: Keys of {*table*}.

Key	Value	Requirement
name	string	required
type	'element', 'set', 'integer', 'continuous', or 'bool'	required
args	list of strings, names of object types	optional
default	constant depending on 'type'	optional, Table A.5 defines the default value
object	name of an object type	required if 'type' is 'set'

Table A.5: Constants for different types.

Type	Value	Default
element	nonnegative integer	'0'
set	list of nonnegative integers	'[]'
integer	integer	'0'
continuous	floating-point number	'0.0'
bool	boolean (true or false)	'false'

In Table A.6, we show the keys of {*transition*}. The key name, which defines the name, is required. The key effect defines the effect of the transition using a map. In the map, each key is the name of a state variable defined with state_variable. The value must be an element expression if it is an element variable, a set expression if it is a set variable, an integer expression if it is an integer variable, and a continuous expression if it is a continuous variable. If the name of variable v is omitted in the map, the effect of that variable is assumed to be no change, i.e., $\mathsf{eff}_\tau[v](S) = S[v]$. The value of the key cost is an integer cost expression if cost_type is integer and a continuous cost expression if continuous. In the cost expression, we can use cost, which represents the cost of the successor state resulting from applying the transition. The key preconditions defines the preconditions of the transition, whose value is a list of {*forall*} described in Table A.8 and conditions. The value of forced is a boolean value (true or false), indicating whether the transition is a forced transition or not.

Table A.6: Keys of {*transition*}.

Key	Value	Requirement
name	string	required
parameters	list of {*parameters*}	optional
effect	map (keys: variables names, values: expressions)	required
cost	⟨*icexpr*⟩ or ⟨*ccexpr*⟩	optional, 'cost' by default
preconditions	list of {*forall*} and ⟨*cond*⟩	optional
forced	boolean	optional, 'false' by default

The key parameters shown in Table A.7 parametrizes the transition. The key name defines the name of the parameter, and the key object defines the associated object type or set variable. If the value of object is the name of an object type defined with objects, then one transition is defined for each index i of the objects. If the value is the name of a set variable U, one transition is defined

for each index i of the objects associated with the variable, and each transition has a precondition $i \in U$. In expressions and conditions in effects, the cost expression, and preconditions, the name of the parameter can be used as an element expression referring to the index i of the object. If multiple parameters are defined, then one transition is defined for each combination of indices. In the DyPDL formalism, forced transitions are totally ordered. In YAML-DyPDL, we use the order in which the transitions are defined. When multiple forced transitions are defined with parameters, they are associated with the indices of objects. We order such transitions by the lexicographic ascending order of the associated indices.

Table A.7: Keys of {*parameter*}.

Key	Value	Requirement
name	string	required
object	string, the name of an object type or a set variable	required

By using {*forall*}, we define the conjunction of multiple conditions. The condition described with the key condition is defined for each combination of indices of objects specified by the value of the key forall. In the condition, we can use the names of the parameters.

Table A.8: Keys of {*forall*}.

Key	Value	Requirement
forall	list of {*parameter*}	required
condition	⟨*cond*⟩	required

In Table A.9, we show the keys of {*base case*}. The key conditions defines a list of conditions in a base case, and the key cost defines the cost using a numeric expression.

Table A.9: Keys of {*base case*}.

Key	Value	Requirement
conditions	list of {*forall*} and ⟨*cond*⟩	required
cost	⟨*iexpr*⟩ or ⟨*cexpr*⟩	required

A problem file is a map described in Table A.10. Transitions, base cases, state constraints, and dual bounds can be also defined in a problem file, which allows problem-specific extensions of a DyPDL model. The value of the key object_numbers is a map whose keys are the names of the object types defined in a domain file. The values are positive integers defining the number of objects. The value of the key target is a map whose keys are the names of the state variables defined in a domain file, meaning the values of the state variables in the target state. The value is a constant depending on the type of a state variable as shown in Table A.5.

The value of the key table_values is a map whose keys are the names of the tables defined in a domain file. For a 0-dimensional table, i.e., a constant, the value is a constant depending on the type as shown in Table A.5. For a 1-dimensional table, the value is a map whose keys are nonnegative integers, and the values are constants. The nonnegative integer of a key must be less than n, the number of the associated objects specified by args in a domain file. For an m-dimensional table with $m > 1$, the value is a map whose keys are lists of nonnegative integers, and the values are constants.

The length of each list must be m. If a key is not defined, its value is assumed to be the value of default defined in a domain file.

Table A.10: Keys of a problem file.

Key	Value	Requirement
object_numbers	map (keys: names of object types, values: integers)	required if objects are defined
target	map (keys: variable names, values: constants)	required
table_values	map (keys: table names, values: maps or constants)	required if tables are defined
transitions	list of {*transition*}	optional
base_cases	list of {*base case*} and a list of ⟨*cond*⟩	optional
constraints	list of {*forall*} and ⟨*cond*⟩	optional
dual_bounds	list of ⟨*iexpr*⟩ or ⟨*cexpr*⟩	optional

A.1.1 Redundant Information in YAML-DyPDL

A user can define resource variables and dual bound functions in YAML-DyPDL. However, they must be consistent with the DyPDL model defined by the YAML-DyPDL files. First, we define the preference between states specified by the preference of the resource variables.

Definition 28 (Preference of states)**.** Given a pair of YAML-DyPDL files defining an optimization problem with a DyPDL model, let S and S' be states. The state S is *preferred* to S' iff $S[v] \geq S'[v]$ for each resource variable v where greater is preferred (preference: greater), $S[v] \leq S'[v]$ for each resource variable where less is preferred (preference: less), and $S[v] = S'[v]$ for each non-resource variable v.

We show that the preference is a preorder, which is one of the conditions of an approximate dominance relation (Definition 17 in Section 3.1.2).

Theorem 45. *Given a pair of YAML-DyPDL files defining an optimization problem with a DyPDL model, the preference between states is a preorder.*

Proof. For a state S, since $S[v] = S[v]$, S is preferred to S, so reflexivity holds. For three states S, S', and S'', suppose that S is preferred to S', and S' is preferred to S''. For resource variable v where greater is preferred, $S[v] \geq S'[v] \geq S''[v]$. For resource variable v where less is preferred, $S[v] \leq S'[v] \leq S''[v]$. For non-resource variable v, $S[v] = S'[v] = S''[v]$. Thus, S is preferred to S'', so transitivity holds. □

With the formal definition of the preference, we introduce the notion of the validity of the YAML-DyPDL files.

Definition 29 (Valid YAML-DyPDL)**.** A pair of YAML-DyPDL files defining an optimization problem with a DyPDL model is *valid* iff the following conditions are satisfied:

- The preference defined by resource variables is an approximate dominance relation, i.e., S dominates S' if S is preferred to S' for reachable states S and S'.
- Numeric expressions defined with the key dual_bounds are dual bound functions.

A user should always provide valid YAML-DyPDL files.

A.1.2 Syntax of Expressions

⟨*eexpr*⟩ ::= ⟨*nonnegative integer*⟩ | ⟨*parameter name*⟩ | ⟨*element variable name*⟩
| '`(`' ⟨*element table name*⟩ ⟨*eexpr*⟩* '`)`' | '`(`' ⟨*ebop*⟩ ⟨*eexpr*⟩ ⟨*eexpr*⟩ '`)`' | '`(if`' ⟨*cond*⟩ ⟨*eexpr*⟩ ⟨*eexpr*⟩ '`)`'

⟨*ebop*⟩ ::= '`+`' | '`-`' | '`*`' | '`/`' | '`%`' | '`max`' | '`min`'

⟨*arg*⟩ ::= ⟨*eexpr*⟩ | ⟨*sexpr*⟩

⟨*sexpr*⟩ ::= ⟨*set variable name*⟩ | '`(`' ⟨*set table name*⟩ ⟨*eexpr*⟩* '`)`' | '`(`' ⟨*srop*⟩ ⟨*set table name*⟩ ⟨*arg*⟩* '`)`'
| '`(complement`' ⟨*sexpr*⟩ '`)`' | '`(`' ⟨*sbop*⟩ ⟨*sexpr*⟩ ⟨*sexpr*⟩ '`)`'
| '`(add`' ⟨*eexpr*⟩ ⟨*sexpr*⟩ '`)`' | '`(remove`' ⟨*eexpr*⟩ ⟨*sexpr*⟩ '`)`' | '`(if`' ⟨*cond*⟩ ⟨*sexpr*⟩ ⟨*sexpr*⟩ '`)`'

⟨*srop*⟩ ::= '`union`' | '`intersection`' | '`disjunctive_union`'

⟨*sbop*⟩ ::= '`union`' | '`intersection`' | '`difference`'

⟨*iexpr*⟩ ::= ⟨*integer*⟩ | ⟨*integer variable name*⟩
| '`(`' ⟨*integer table name*⟩ ⟨*eexpr*⟩* '`)`' | '`(`' ⟨*nrop*⟩ ⟨*integer table name*⟩ ⟨*arg*⟩* '`)`'
| '`(abs`' ⟨*iexpr*⟩ '`)`' | '`(`' ⟨*ibop*⟩ ⟨*iexpr*⟩ ⟨*iexpr*⟩ '`)`' | '`(`' ⟨*round*⟩ ⟨*cexpr*⟩ '`)`' | '`|`' ⟨*sexpr*⟩ '`|`'
| '`(if`' ⟨*cond*⟩ ⟨*iexpr*⟩ ⟨*iexpr*⟩ '`)`'

⟨*nrop*⟩ ::= '`sum`' | '`max`' | '`min`'

⟨*ibop*⟩ ::= '`+`' | '`-`' | '`*`' | '`/`' | '`%`' | '`max`' | '`min`'

⟨*round*⟩ ::= '`ceil`' | '`floor`' | '`round`' | '`trunc`'

⟨*cexpr*⟩ ::= ⟨*integer*⟩ | ⟨*floating-point number*⟩ | ⟨*integer variable name*⟩ | ⟨*continuous variable name*⟩
| '`(`' ⟨*integer table name*⟩ ⟨*eexpr*⟩* '`)`' | '`(`' ⟨*nrop*⟩ ⟨*integer table name*⟩ ⟨*arg*⟩* '`)`'
| '`(`' ⟨*continuous table name*⟩ ⟨*eexpr*⟩* '`)`' | '`(`' ⟨*nrop*⟩ ⟨*continuous table name*⟩ ⟨*arg*⟩* '`)`'
| '`(abs`' ⟨*cexpr*⟩ '`)`' | '`(sqrt`' ⟨*cexpr*⟩ '`)`' | '`(`' ⟨*cbop*⟩ ⟨*cexpr*⟩ ⟨*cexpr*⟩ '`)`' | (' ⟨*round*⟩ ⟨*cexpr*⟩ '`)`'
| '`|`' ⟨*seexpr*⟩ '`|`' | '`(continuous`' ⟨*iexpr*⟩ '`)`' | '`(if`' ⟨*cond*⟩ ⟨*cexpr*⟩ ⟨*cexpr*⟩ '`)`'

⟨*cbop*⟩ ::= '`+`' | '`-`' | '`*`' | '`/`' | '`%`' | '`max`' | '`min`' | '`pow`' | '`log`'

⟨*cond*⟩ ::= '`(`' ⟨*boolean table name*⟩ ⟨*eexpr*⟩* '`)`' | '`(not`' ⟨*cond*⟩ '`)`'
| '`(and`' ⟨*cond*⟩ ⟨*cond*⟩ '`)`' | '`(or`' ⟨*cond*⟩ ⟨*cond*⟩ '`)`'
| '`(`' ⟨*comp*⟩ ⟨*eexpr*⟩ ⟨*eexpr*⟩ '`)`' | '`(`' ⟨*comp*⟩ ⟨*iexpr*⟩ ⟨*iexpr*⟩ '`)`' | '`(`' ⟨*comp*⟩ ⟨*cexpr*⟩ ⟨*cexpr*⟩ '')'
| '`(is_empty`' ⟨*sexpr*⟩ '`)`' | '`(`' ⟨*scomp*⟩ ⟨*sexpr*⟩ ⟨*sexpr*⟩ '`)`' | '`(is_in`' ⟨*eexpr*⟩ ⟨*sexpr*⟩ '`)`'

⟨*comp*⟩ ::= '`=`' | '`!=`' | '`>`' | '`>=`' | '`<`' | '`<=`'

⟨*scomp*⟩ ::= '`=`' | '`!=`' | '`is_subset`'

Figure A.1: BNF of expressions.

We define the syntax of expressions and conditions using the Backus-Naur form (BNF). An expression following the syntax may not have a well-defined meaning, e.g., zero division. In such a case, the computational result is undefined, and a system should raise an exception in practice.

An element expression (⟨*eexpr*⟩) can refer to a nonnegative integer (⟨*nonegative integer*⟩), the name of a parameter (⟨*parameter name*⟩) referring to the index of an object, and the name of an element variable (⟨*element variable name*⟩). With '`(`' ⟨*element table name*⟩ ⟨*eexpr*⟩∗ '`)`', we can access a constant in a table. Here, ⟨*element table name*⟩ is the name of an element table, and ⟨*eexpr*⟩∗ is a sequence of element expressions separated by white spaces, meaning arguments. Thus, '`(`' ⟨*element table name*⟩ ⟨*eexpr*⟩∗ '`)`' refers to a constant in the table with the key specified by the element expressions. If the key is undefined, e.g., an element expression returns a value greater

than the number of associated objects to the argument, the result is undefined. The addition (`+`), subtraction (`-`), multiplication (`*`), division (`/`), and modulo (`%`) of two element expressions yields a new element expression. If a negative value results from subtraction, the behavior is undefined. For division, we truncate fractions. Zero-division is undefined. We can also take the maximum (`max`) and the minimum (`min`) of two element expressions.

In a set expression (⟨*sexpr*⟩), we can take the union (`union`), intersection (`intersection`), and difference (`difference`) of two set expressions. We can also add (`add`) an element expression to a set expression and remove (`remove`) an element expression from a set expression. Set variables and set tables are associated with object types. When we take the union, intersection, and difference of two set expressions, they should be associated with the same object type, and the resulting set expression is associated with that object type. If they are associated with different object types, the behavior is undefined. Similarly, if we add or remove an element whose index is greater than or equal to the number of the associated objects, the behavior is undefined. Since a set expression is associated with an object type, the maximum index that can be included in the set is known ($n-1$ if there are n objects), so we can take the complement (`complement`) of a set expression. We can also take the union (`union`), intersection (`intersection`), and disjunctive union (`disjunctive_union`) of multiple constants in a set table over multiple indices with '`(`' ⟨*srop*⟩ ⟨*set table name*⟩ ⟨*arg*⟩∗ '`)`'. Here, each argument ⟨*arg*⟩ can be an element or a set expression. We consider the set of indices defined by the Cartesian product of the arguments, considering an element expression as a set with a single element. Then, we take the union, intersection, and disjunctive union of sets in the table over the set of indices. For example, suppose a 3-dimensional set table `E`, each of whose element is denoted by E_{ijk}. Let the value of a set variable `A` be $\{0, 1\}$ and the value of a set variable `B` be $\{1, 3\}$. If we use `(union E 0 A B)`, then, we consider a set of indices $I = \{(0, 0, 1), (0, 0, 3), (0, 1, 1), (0, 1, 3)\}$, and the expression returns $\bigcup_{(i,j,k)\in I} E_{ijk}$.

In an integer expression (⟨*iexpr*⟩), we can refer to an integer (⟨*integer*⟩), an integer variable (⟨*integer variable name*⟩), and an integer table (⟨*integer table name*⟩). We can take the summation (`sum`), maximum (`max`), and minimum (`min`) of constants in an integer table over multiple indices defined by element and set expressions. Arithmetic operations are defined in the same way as in an element expression, while a negative result is allowed in subtraction. In addition, we can take the absolute value (`abs`) of an integer expression. We can also round a continuous expression to an integer expression by using the ceiling function (`ceil`) and the floor function (`floor`). If we use `round`, it returns the nearest integer (the lower one for a half-way between two integers). If we use `trunc`, fractions are truncated. We can take the cardinality of a set expression by using '`|`' ⟨*sexpr*⟩ '`|`'.

A continuous expression (⟨*cexpr*⟩) is defined similarly to an integer expression. The modulo, '`(% x y)`', is defined to be $x - \mathrm{trunc}(x/y) \cdot y$, where trunc truncates fractions. In a continuous expression, we can take the square root (`sqrt`), the exponentiation (`pow`, where the second argument is the exponent), and the logarithm (`log`, where the second argument is the base). The square root of a negative value is undefined. For the logarithm, if either of the arguments is negative, the result is undefined. We can convert an integer expression into a continuous expression by using `continuous`. In addition, integers, integer variables, and integer tables can be considered continuous expressions.

In a condition (⟨*condition*⟩), we can refer to a constant in a boolean table (⟨*boolean table name*⟩), the negation of a condition (`not`), and the conjunction (`and`) or disjunction (`or`) of two conditions.

For two element, set, integer, or continuous expressions, we can check if they return the same value (`=`) or different values (`!=`). In addition, for two element, integer, or continuous expressions, we can check if one is greater than the other (`>`), one is greater than or equal to the other (`>=`), one is less than the other (`<`), or one is less than or equal to the other (`<=`). When we compare two expressions that can be parsed as both integer and continuous expressions, e.g., integers, a parser can assume that they are integer expressions. This assumption does not change the result of the comparison. For two set expressions, we can check if the first one is a subset of the second one (`is_subset`). For an element expression and a set expression, we can check if the element is included in the set (`is_in`). We can check if a set expression returns an empty set (`is_empty`). With a condition, we can use an 'if-then-else' expression (`if`), which returns the second argument if the condition (the first argument) holds and the third argument otherwise.

An integer cost expression ($\langle icexpr \rangle$) and a continuous cost expression ($\langle ccexpr \rangle$) are defined in the same way as integer and continuous expressions, respectively. In addition, '`cost`' can be used as an integer or a continuous cost expression, which refers to the cost of the successor state (x of a cost expression $\texttt{cost}_\tau(x, S)$). In element expressions, set expressions, and conditions used in cost expressions, we can use integer and continuous cost expressions instead of integer and continuous expressions.

A.2 YAML-DyPDL Domain Files

We present YAML-DyPDL domain files for the DyPDL models of the eleven combinatorial optimization problems introduced in Section 3.3. For the traveling salesperson problem with time windows (TSPTW) and the capacitated vehicle routing problem (CVRP), we present domain files where the cost of the base case is zero in Figures A.2 and A.5, which are equivalent to the models used in Chapter 6. In Figure A.3, we present a domain file for TSPTW to minimize the makespan, which is used in Appendix C.2. For the orienteering problem with time windows (OPTW), the definition of the dual bound function depends on an instance, so we present a domain file in Figure A.7 and an example problem file in Figure A.8. For the multi-dimensional knapsack problem (MDKP), since the number of state variables depends on an instance, we show domain and problem files for a two-dimensional instance, i.e., $m = 2$ in Figure A.9.

```
cost_type: integer
objects:
  - customer
state_variables:
  - name: U
    type: set
    object: customer
  - name: i
    type: element
    object: customer
  - name: t
    type: integer
    preference: less
tables:
  - name: a
    type: integer
    args:
      - customer
  - name: b
    type: integer
    args:
      - customer
  - name: c
    type: integer
    args:
      - customer
      - customer
  - name: cstar
    type: integer
    args:
      - customer
      - customer
  - name: cin
    type: integer
    args:
      - customer
  - name: cout
    type: integer
    args:
      - customer
```

```
transitions:
  - name: visit
    parameters:
      - name: j
        object: U
    effect:
      U: (remove j U)
      i: j
      t: (max (+ t (c i j)) (a j))
    cost: (+ (c i j) cost)
    preconditions:
      - (<= (+ t (c i j)) (b j))
  - name: return
    preconditions:
      - (is_empty U)
      - (!= i 0)
    effect:
      i: 0
      t: (+ t (c i 0))
    cost: (+ (c i 0) cost)
constraints:
  - condition: (<= (+ t (cstar i j)) (b j))
    forall:
      - name: j
        object: U
base_cases:
  - conditions:
      - (is_empty U)
      - (= i 0)
    cost: 0
dual_bounds:
  - (+ (sum cin U) (if (!= i 0) (cin 0) 0))
  - (+ (sum cout U) (if (!= i 0) (cout i) 0))
reduce: min
```

Figure A.2: YAML-DyPDL domain file for the traveling salesperson problem with time windows (TSPTW) where the cost of the base case is zero.

```
cost_type: integer
objects:
  - customer
state_variables:
  - name: U
    type: set
    object: customer
  - name: i
    type: element
    object: customer
  - name: t
    type: integer
    preference: less
tables:
  - name: a
    type: integer
    args:
      - customer
  - name: b
    type: integer
    args:
      - customer
  - name: c
    type: integer
    args:
      - customer
      - customer
  - name: cstar
    type: integer
    args:
      - customer
      - customer
  - name: cin
    type: integer
    args:
      - customer
  - name: cout
    type: integer
    args:
      - customer
```

```
transitions:
  - name: visit
    parameters:
      - name: j
        object: U
    effect:
      U: (remove j U)
      i: j
      t: (max (+ t (c i j)) (a j))
    cost: (+ (max (c i j) (- (a j) t)) cost)
    preconditions:
      - (<= (+ t (c i j)) (b j))
constraints:
  - condition: (<= (+ t (cstar i j)) (b j))
    forall:
      - name: j
        object: U
base_cases:
  - conditions:
      - (is_empty U)
    cost: (c i 0)
dual_bounds:
  - (+ (sum cin U) (cin 0))
  - (+ (sum cout U) (cout i))
reduce: min
```

Figure A.3: YAML-DyPDL domain file for the traveling salesperson problem with time windows to minimize the makespan (TSPTW-M).

```
cost_type: integer
objects:
  - customer
state_variables:
  - name: U
    type: set
    object: customer
  - name: i
    type: element
    object: customer
  - name: l
    type: integer
    preference: less
  - name: k
    type: integer
    preference: less
tables:
  - name: m
    type: integer
  - name: q
    type: integer
  - name: d
    type: integer
    args:
      - customer
  - name: c
    type: integer
    args:
      - customer
      - customer
  - name: cin
    type: integer
    args:
      - customer
  - name: cout
    type: integer
    args:
      - customer
```

```
transitions:
  - name: visit
    parameters:
      - name: j
        object: U
    effect:
      U: (remove j U)
      i: j
      l: (+ l (d j))
    cost: (+ (c i j) cost)
    preconditions:
      - (<= (+ l (d j)) q)
  - name: visit-via-depot
    parameters:
      - name: j
        object: U
    effect:
      U: (remove j U)
      i: j
      l: (d j)
      k: (+ k 1)
    cost: (+ (+ (c i 0) (c 0 j)) cost)
    preconditions:
      - (< k m)
constraints:
  - condition: (>= (+ (* (- m k) q) q) (+ l (sum d U)))
base_cases:
  - conditions:
      - (is_empty U)
    cost: (c i 0)
dual_bounds:
  - (+ (sum cin U) (cin 0))
  - (+ (sum cout U) (cout i))
reduce: min
```

Figure A.4: YAML-DyPDL domain file for the capacitated vehicle routing problem (CVRP).

```
cost_type: integer
objects:
  - customer
state_variables:
  - name: U
    type: set
    object: customer
  - name: i
    type: element
    object: customer
  - name: l
    type: integer
    preference: less
  - name: k
    type: integer
    preference: less
tables:
  - name: m
    type: integer
  - name: q
    type: integer
  - name: d
    type: integer
    args:
      - customer
  - name: c
    type: integer
    args:
      - customer
      - customer
  - name: cin
    type: integer
    args:
      - customer
  - name: cout
    type: integer
    args:
      - customer
```

```
transitions:
  - name: visit
    parameters:
      - name: j
        object: U
    effect:
      U: (remove j U)
      i: j
      l: (+ l (d j))
    cost: (+ (c i j) cost)
    preconditions:
      - (<= (+ l (d j)) q)
  - name: visit-via-depot
    parameters:
      - name: j
        object: U
    effect:
      U: (remove j U)
      i: j
      l: (d j)
      k: (+ k 1)
    cost: (+ (+ (c i 0) (c 0 j)) cost)
    preconditions:
      - (< k m)
  - name: return
    preconditions:
      - (is_empty U)
      - (!= i 0)
    effect:
      i: 0
    cost: (+ (c i 0) cost)
constraints:
  - condition: (>= (+ (* (- m k) q) q) (+ l (sum d U)))
base_cases:
  - conditions:
      - (is_empty U)
      - (= i 0)
    cost: 0
dual_bounds:
  - (+ (sum cin U) (if (!= i 0) (cin 0) 0))
  - (+ (sum cout U) (if (!= i 0) (cout i) 0))
reduce: min
```

Figure A.5: YAML-DyPDL domain file for the capacitated vehicle routing problem (CVRP) where the cost of the base case is zero.

```
cost_type: integer
objects:
  - customer
state_variables:
  - name: U
    type: set
    object: customer
  - name: i
    type: element
    object: customer
  - name: l
    type: integer
    preference: less
tables:
  - name: goal
    type: element
    object: customer
  - name: q
    type: integer
  - name: P
    type: set
    object: customer
    args:
      - customer
  - name: delta
    type: integer
    args:
      - customer
  - name: A
    type: bool
    args:
      - customer
      - customer
    default: false
  - name: c
    type: integer
    args:
      - customer
      - customer
  - name: cin
    type: integer
    args:
      - customer
  - name: cout
    type: integer
    args:
      - customer
```

```
transitions:
  - name: visit
    parameters:
      - name: j
        object: U
    effect:
      U: (remove j U)
      i: j
      l: (+ l (delta j))
    cost: (+ (c i j) cost)
    preconditions:
      - (A i j)
      - (<= (+ l (delta j)) q)
      - (is_empty (intersection U (P j)))
base_cases:
  - conditions:
      - (is_empty U)
      - (A i goal)
    cost: (c i goal)
dual_bounds:
  - (+ (sum cin U) (cin goal))
  - (+ (sum cout U) (cout i))
reduce: min
```

Figure A.6: YAML-DyPDL domain file for multi-commodity pickup and delivery traveling salesperson problem (m-PDTSP).

```
cost_type: integer
objects:
  - customer
state_variables:
  - name: U
    type: set
    object: customer
  - name: i
    type: element
    object: customer
  - name: t
    type: integer
    preference: less
tables:
  - name: p
    type: integer
    args:
      - customer
  - name: a
    type: integer
    args:
      - customer
  - name: b
    type: integer
    args:
      - customer
  - name: c
    type: integer
    args:
      - customer
      - customer
  - name: cstar
    type: integer
    args:
      - customer
      - customer
  - name: cstar_cstar0
    type: integer
    args:
      - customer
      - customer
  - name: c_cstar0
    type: integer
    args:
      - customer
      - customer
  - name: cin
    type: continuous
    args:
      - customer
  - name: cout
    type: continuous
    args:
      - customer
  - name: ein
    type: continuous
    args:
      - customer
  - name: eout
    type: continuous
    args:
      - customer
```

```
transitions:
  - name: remove-by-time
    parameters:
      - name: j
        object: U
    effect:
      U: (remove j U)
    cost: cost
    preconditions:
      - >
        (or (> (+ t (cstar i j)) (b j))
            (> (+ t (cstar_cstar0 i j)) (b 0)))
    forced: true
  - name: remove-by-paths
    parameters:
      - name: j
        object: U
    effect:
      U: (remove j U)
    preconditions:
      - condition: >
          (or (> (+ t (c i k)) (b k))
              (> (+ t (c_cstar0 i k)) (b 0)))
        forall:
          - name: k
            object: U
    cost: cost
    forced: true
  - name: visit
    parameters:
      - name: j
        object: U
    effect:
      U: (remove j U)
      i: j
      t: (max (+ t (c i j)) (a j))
    cost: (+ (p j) cost)
    preconditions:
      - (<= (+ t (c i j)) (b j))
      - (<= (+ t (c_cstar0 i j)) (b 0))
base_cases:
  - - (is_empty U)
    - (<= (+ t (c i 0)) (b 0))
reduce: max
```

Figure A.7: YAML-DyPDL domain file for the orienteering problem with time windows (OPTW).

```
object_numbers:
  customer: 4
target:
  U: [1, 2, 3]
  i: 0
  t: 0
table_values:
  p: { 1: 1, 2: 2, 3: 3 }
  a: { 0: 0, 1: 5, 2: 0, 3: 8 }
  b: { 0: 20, 1: 16, 2: 10, 3: 14 }
  c:
    {
      [0, 1]: 3, [0, 2]: 4, [0, 3]: 5,
      [1, 0]: 3, [1, 2]: 5, [1, 3]: 4,
      [2, 0]: 4, [2, 1]: 5, [2, 3]: 3,
      [3, 0]: 5, [3, 1]: 4, [3, 2]: 3,
    }
  cstar:
    {
      [0, 1]: 3, [0, 2]: 4, [0, 3]: 5,
      [1, 0]: 3, [1, 2]: 5, [1, 3]: 4,
      [2, 0]: 4, [2, 1]: 5, [2, 3]: 3,
      [3, 0]: 5, [3, 1]: 4, [3, 2]: 3,
    }
  c_cstar0:
    {
      [0, 1]: 6, [0, 2]: 8, [0, 3]: 10,
      [1, 0]: 3, [1, 2]: 9, [1, 3]: 9,
      [2, 0]: 4, [2, 1]: 8, [2, 3]: 8,
      [3, 0]: 5, [3, 1]: 7, [3, 2]: 7,
    }
  cstar_cstar0:
    {
      [0, 1]: 6, [0, 2]: 8, [0, 3]: 10,
      [1, 0]: 3, [1, 2]: 9, [1, 3]: 9,
      [2, 0]: 4, [2, 1]: 8, [2, 3]: 8,
      [3, 0]: 5, [3, 1]: 7, [3, 2]: 7,
    }
  cin: { 0: 3, 1: 3, 2: 3, 3: 3 }
  cout: { 0: 3, 1: 3, 2: 3, 3: 3 }
  ein: { 1: 0.334, 2: 0.667, 3: 1.0 }
  eout: { 1: 0.334, 2: 0.667, 3: 1.0 }
dual_bounds:
  - >
    (+ (if (and (is_in 1 U)
                (and (<= (+ t (cstar i 1)) 16)
                     (<= (+ t (cstar_cstar0 i 1)) 20))) 1 0)
    (+ (if (and (is_in 2 U)
                (and (<= (+ t (cstar i 2)) 10)
                     (<= (+ t (cstar_cstar0 i 2)) 20))) 2 0)
       (if (and (is_in 3 U)
                (and (<= (+ t (cstar i 3)) 14)
                     (<= (+ t (cstar_cstar0 i 3)) 20))) 3 0)))
  - >
    (floor (* (- (- 20 t) (cin i))
              (max (if (and (is_in 1 U)
                            (and (<= (+ t (cstar i 1)) 16)
                                 (<= (+ t (cstar_cstar0 i 1)) 20))) 0.334 0)
              (max (if (and (is_in 2 U)
                            (and (<= (+ t (cstar i 2)) 10)
                                 (<= (+ t (cstar_cstar0 i 2)) 20))) 0.667 0)
                   (if (and (is_in 3 U)
                            (and (<= (+ t (cstar i 3)) 14)
                                 (<= (+ t (cstar_cstar0 i 3)) 20))) 1.0 0)))))
  - >
    (floor (* (- (- 20 t) 3)
              (max (if (and (is_in 1 U)
                            (and (<= (+ t (cstar i 1)) 16)
                                 (<= (+ t (cstar_cstar0 i 1)) 20))) 0.334 0)
              (max (if (and (is_in 2 U)
                            (and (<= (+ t (cstar i 2)) 10)
                                 (<= (+ t (cstar_cstar0 i 2)) 20))) 0.667 0)
                   (if (and (is_in 3 U)
                            (and (<= (+ t (cstar i 3)) 14)
                                 (<= (+ t (cstar_cstar0 i 3)) 20))) 1.0 0)))))
```

Figure A.8: YAML-DyPDL problem file for the orienteering problem with time windows (OPTW).

```
cost_type: integer
objects:
  - item
state_variables:
  - name: i
    type: element
    object: item
  - name: r0
    type: integer
  - name: r1
    type: integer
tables:
  - name: p
    type: integer
    args:
      - item
  - name: sum_p
    type: integer
    args:
      - item
  - name: w0
    type: integer
    args:
      - item
  - name: w1
    type: integer
    args:
      - item
  - name: e0
    type: continuous
    args:
      - item
  - name: e1
    type: continuous
    args:
      - item
transitions:
  - name: pack
    effect:
      i: (+ i 1)
      r0: (- r0 (w0 i))
      r1: (- r1 (w1 i))
    cost: (+ (p i) cost)
    preconditions:
      - (>= r0 (w0 i))
      - (>= r1 (w1 i))
  - name: ignore
    effect:
      i: (+ i 1)
    cost: cost
dual_bounds:
  - (sum_p i)
  - (floor (* (e0 i) (max r0 1)))
  - (floor (* (e1 i) (max r1 1)))
reduce: max
```

```
object_numbers:
  item: 4
target:
  i: 0
  r0: 8
  r1: 6
table_values:
  p: { 0: 1, 1: 2, 2: 3, 3: 0 }
  sum_p: { 0: 6, 1: 5, 2: 3, 3: 0 }
  w0: { 0: 2, 1: 3, 2: 4, 3: 0 }
  w1: { 0: 3, 1: 4, 2: 2, 3: 0 }
  e0: { 0: 0.5, 1: 0.667, 2: 0.75, 3: 0 }
  e1: { 0: 0.34, 1: 0.5, 2: 1.5, 3: 0  }
base_cases:
  - - (= i 3)
```

Figure A.9: YAML-DyPDL domain and problem files for the multi-dimensional knapsack problem (MDKP).

```
cost_type: integer
objects:
  - item
state_variables:
  - name: U
    type: set
    object: item
  - name: r
    type: integer
    preference: greater
  - name: k
    type: element
    object: item
    preference: less
tables:
  - name: q
    type: integer
  - name: w
    type: integer
    args:
      - item
  - name: a
    type: integer
    args:
      - item
    default: 0
  - name: b
    type: continuous
    args:
      - item
    default: 0.0
  - name: c
    type: continuous
    args:
      - item
    default: 0.0
```

```
transitions:
  - name: open-and-pack
    parameters:
      - name: i
        object: U
    effect:
      U: (remove i U)
      r: (- q (w i))
      k: (+ k 1)
    cost: (+ 1 cost)
    preconditions:
      - (>= i k)
      - forall:
          - name: j
            object: U
        condition: (< r (w j))
    forced: true
  - name: pack
    parameters:
      - name: i
        object: U
    effect:
      U: (remove i U)
      r: (- r (w i))
    cost: cost
    preconditions:
      - (>= r (w i))
      - (>= (+ i 1) k)
base_cases:
  - - (is_empty U)
dual_bounds:
  - (ceil (/ (- (sum w U) r) q))
  - (- (+ (sum a U) (ceil (sum b U))) (if (>= r (/ q 2.0)) 1 0))
  - (- (ceil (sum c U)) (if (>= r (/ q 3.0)) 1 0))
reduce: min
```

Figure A.10: YAML-DyPDL domain file for bin packing.

```
cost_type: integer
objects:
  - task
state_variables:
  - name: U
    type: set
    object: task
  - name: r
    type: integer
    preference: greater
tables:
  - name: q
    type: integer
  - name: w
    type: integer
    args:
      - task
  - name: P
    type: set
    object: task
    args:
      - task
    default: []
  - name: a
    type: integer
    args:
      - task
    default: 0
  - name: b
    type: continuous
    args:
      - task
    default: 0.0
  - name: c
    type: continuous
    args:
      - task
    default: 0.0
```

```
transitions:
  - name: open-new-station
    forced: true
    effect:
      r: q
    cost: (+ 1 cost)
    preconditions:
      - forall:
          - name: i
            object: U
        condition: (or (> (w i) r) (> |(intersection U (P i))| 0))
  - name: schedule
    parameters:
      - name: i
        object: U
    effect:
      U: (remove i U)
      r: (- r (w i))
    cost: cost
    preconditions:
      - (is_empty (intersection U (P i)))
      - (<= (w i) r)
base_cases:
  - - (is_empty U)
dual_bounds:
  - (ceil (/ (- (sum w U) r) q))
  - (- (+ (sum a U) (ceil (sum b U))) (if (>= r (/ q 2.0)) 1 0))
  - (- (ceil (sum c U)) (if (>= r (/ q 3.0)) 1 0))
reduce: min
```

Figure A.11: YAML-DyPDL domain file for the simple assembly line balancing problem (SALBP-1).

```
cost_type: integer
objects:
  - job
state_variables:
  - name: F
    type: set
    object: job
tables:
  - name: 'N'
    type: set
    object: job
  - name: p
    type: integer
    args:
      - job
  - name: d
    type: integer
    args:
      - job
  - name: w
    type: integer
    args:
      - job
  - name: P
    type: set
    object: job
    args:
      - job
```

```
transitions:
  - name: schedule
    parameters:
      - name: i
        object: job
    effect:
      F: (add i F)
    cost: (+ (* (w i) (max 0 (- (+ (sum p F) (p i)) (d i)))) cost)
    preconditions:
      - (not (is_in i F))
      - (is_empty (difference (P i) F))
base_cases:
  - - (= F N)
dual_bounds:
  - 0
reduce: min
```

Figure A.12: YAML-DyPDL domain file for single machine total weighted tardiness ($1||\sum w_i T_i$).

```
cost_type: integer
objects:
  - scene
  - actor
state_variables:
  - name: Q
    type: set
    object: scene
tables:
  - name: A
    type: set
    object: actor
    args:
      - scene
  - name: d
    type: integer
    args:
      - scene
  - name: c
    type: integer
    args:
      - actor
  - name: db
    type: integer
    args:
      - scene
  - name: P
    type: set
    object: scene
    args:
      - scene
```

```
transitions:
  - name: shoot-with-actors-on-location
    parameters:
      - name: s
        object: Q
    effect:
      Q: (remove s Q)
    cost: (+ (db s) cost)
    preconditions:
      - (= (A s) (intersection (union A Q) (union A ~Q)))
    forced: true
  - name: shoot
    parameters:
      - name: s
        object: Q
    effect:
      Q: (remove s Q)
    cost: >
      (+ (* (d s)
            (sum c (union (A s)
                          (intersection (union A Q) (union A ~Q)))))
          cost)
    preconditions:
      - forall:
          - name: t
            object: Q
        condition: >
          (not (and (is_in t (P s))
                    (is_subset (A t) (union (union A ~Q) (A s)))))
base_cases:
  - - (is_empty Q)
dual_bounds:
  - (sum db Q)
reduce: min
```

Figure A.13: YAML-DyPDL domain file for talent scheduling.

```
cost_type: integer
objects:
  - customer
state_variables:
  - name: R
    type: set
    object: customer
  - name: O
    type: set
    object: customer
tables:
  - name: 'N'
    type: set
    object: customer
    args:
      - customer
```

```
transitions:
  - name: close
    parameters:
      - name: c
        object: R
    effect:
      R: (remove c R)
      O: (union O (N c))
    cost: >
      (max |(union (intersection O R) (difference (N c) O))| cost)
base_cases:
  - - (is_empty R)
dual_bounds:
  - 0
reduce: min
```

Figure A.14: YAML-DyPDL domain file for the minimization of open stacks problem (MOSP).

```
cost_type: integer
objects:
  - node
state_variables:
  - name: C
    type: set
    object: node
tables:
  - name: 'N'
    type: set
    object: node
  - name: a
    type: integer
    args:
      - node
  - name: b
    type: integer
    args:
      - node
      - node
    default: 0
```

```
transitions:
  - name: sweep
    parameters:
      - name: c
        object: node
    effect:
      C: (add c C)
    cost: (max (+ (a c) (+ (sum b c N) (sum b C (remove c ~C)))) cost)
    preconditions:
      - (not (is_in c C))
base_cases:
  - - (= C N)
dual_bounds:
  - 0
reduce: min
```

Figure A.15: YAML-DyPDL domain file for graph-clear.

Appendix B

Mixed-Integer Programming and Constraint Programming Models

We present the mixed-integer programming (MIP) and constraint programming (CP) models for the combinatorial optimization problems used in the experimental evaluation.

B.1 Traveling Salesperson Problem with Time Windows

In the traveling salesperson problem with time windows (TSPTW) [377] introduced in Section 3.2.1, a set of customers $N = \{0, ..., n-1\}$, where 0 is the depot, the travel time c_{ij} from customer i to j, and time window $[a_j, b_j]$ for each customer $j \in N \setminus \{0\}$ are given. The objective is to minimize the total travel time of a tour that visits each customer exactly once within time window starting from and returning to the depot.

B.1.1 MIP Model

We use the MIP model proposed by Hungerländer and Truden [222]. They reduced edges between customers based on time windows. A customer j can be visited after another customer i only if $a_i \leq b_j$. Furthermore, if there exists a customer k such that $b_i < a_k$ and $b_k < a_j$, k must be visited after i and before j.[1] Therefore, edge (i, j) with $i \neq j$ belongs to the set of edges considered, $\tilde{E}$, when one of the following conditions is satisfied:

- $i = 0$ or $j = 0$.
- $a_i \leq b_j$ and there does not exists $k \in N \setminus \{0\}$ such that $b_i < a_k$ and $b_k < a_j$.[2]

In the MIP model, binary decision variable x_{ij} with $(i, j) \in \tilde{E}$ represents whether j is visited directly after i, and integer decision variable t_i represents the time when i is visited. Since we assume that the tour starts from and ends at the depot, we introduce a decision variable t_n, the time when

[1] The original paper used $a_i < b_j$, $b_i \leq a_k$, and $b_k \leq a_j$ assuming $c_{ik}, c_{ij}, c_{jk} > 0$.

[2] Since the original paper used $a_i < b_j$, $b_i \leq a_k$, and $b_k \leq a_j$, they explicitly included (i, j) with $a_i = a_j$ and $b_i = b_j$ in addition.

the tour ends at the depot. In addition, the original paper considered the service time s_i at customer i, but we assume $s_i = 0$;

$$\min \sum_{(i,j)\in\tilde{E}} c_{ij}x_{ij} \tag{B.1}$$

$$\text{s.t.} \sum_{(j,i)\in\tilde{E}} x_{ji} = \sum_{(i,j)\in\tilde{E}} x_{ij} = 1 \qquad \forall i \in N \tag{B.2}$$

$$a_i \le t_i \le b_i \qquad i \in N \setminus \{0\} \tag{B.3}$$

$$t_i - t_j + (b_i - a_j + c_{ij})x_{ij} \le b_i - a_j \qquad \forall (i,j) \in \tilde{E} \wedge i \neq 0 \wedge j \neq 0 \tag{B.4}$$

$$t_i - c_{0i}x_{0i} \ge 0 \qquad \forall i \in N \setminus \{0\} \tag{B.5}$$

$$t_i + c_{i0} \le t_n \qquad \forall i \in N \setminus \{0\} \tag{B.6}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in \tilde{E} \tag{B.7}$$

$$t_i \ge 0 \qquad \forall i \in N \cup \{n\}. \tag{B.8}$$

Constraint (B.2) ensures that each customer is visited once. Constraint (B.3) is the time window constraint. Constraint (B.4) ensures that when j is visited after i, i.e., $x_{ij} = 1$, $t_i + c_{ij} \le t_j$. When j is not visited after i, i.e., $x_{ij} = 0$, it becomes $t_i - t_j \le b_i - a_j$, which is always satisfied since $t_i \le b_i$ and $t_j \ge a_j$. Constraints (B.5) and (B.6) ensure that the tour starts from and returns to the depot. When the travel time is always positive, Constraints (B.4)–(B.6) are sufficient to eliminate subtours, cycles that do not visit all customers. However, if there exists $(i,j) \in \tilde{E}$ with $c_{ij} = 0$, a solution for the above MIP model may include subtours. If there are such (i,j), the following flow-based subtour elimination constraints [157] are added to the model before solving.

$$y_{ij} \le (n-1)x_{ij} \qquad \forall (i,j) \in \tilde{E} \tag{B.9}$$

$$y_{0i} = n - 1 \qquad \forall i \in N \setminus \{0\} \tag{B.10}$$

$$\sum_{(i,j)\in\tilde{E}} y_{ij} - \sum_{(j,k)\in\tilde{E}} y_{jk} = 1 \qquad \forall j \in N \setminus \{0\} \tag{B.11}$$

$$y_{ij} \ge 0 \qquad \forall (i,j) \in \tilde{E}. \tag{B.12}$$

B.1.2 CP Model

We adapt a CP model for a single machine scheduling problem with time windows and sequence-dependent setup times [49]. We define an interval variable x_i that represents visiting customer i in $[a_i, b_i]$ with the service time of 0. We use a sequence variable π to sequence the interval variables.

$$\min \sum_{i\in N} c_{i,\mathsf{TypeOfNext}(\pi,x_i)} \tag{B.13}$$

$$\text{s.t. } \mathsf{NoOverlap}(\pi, \{c_{ij} \mid (i,j) \in N \times N\}) \tag{B.14}$$

$$\mathsf{First}(\pi, x_0) \tag{B.15}$$

$$x_i : \mathsf{intervalVar}(0, [a_i, b_i]) \qquad \forall i \in N \setminus \{0\} \tag{B.16}$$

$$x_0 : \mathsf{intervalVar}(0, [0,0]) \tag{B.17}$$

$$\pi : \mathsf{sequenceVar}(\{x_i \mid i \in N\}). \tag{B.18}$$

In Objective (B.13), $\mathsf{TypeOfNext}(\pi, x_i)$ represents the index of the next variable in sequence π. For example, if x_j comes directly after x_i in π, $\mathsf{TypeOfNext}(\pi, x_i) = j$. For the last variable in the sequence, we define $\mathsf{TypeOfNext}(\pi, x_i) = 0$. Constraint (B.14) ensures that if x_j comes directly after x_i in π, the distance between them is at least c_{ij}, using $c_{ii} = 0$. Constraint (B.15) ensures that the tour starts from the depot.

B.2 Capacitated Vehicle Routing Problem

In the capacitated vehicle routing problem (CVRP) [95] introduced in Section 3.3.1, a set of customers $N = \{0, ..., n-1\}$, where 0 is the depot, and m vehicles are given. Each customer has the demand d_i, and each vehicle has the capacity q. If a vehicle visits customer i, the load increases by d_i, and the total load of the vehicle must not exceed q. Visiting customer j from i incurs the travel time c_{ij}. The objective is to visit all customers using the vehicles starting from and returning to the depot.

B.2.1 MIP Model

We use the MIP model proposed by Gadegaard and Lysgaard [151]. The model is based on a flow-based formulation [157] but exploits symmetry assuming that $c_{ij} = c_{ji}$. The idea is to represent a route for a vehicle as two paths from the depot to one customer, called a peak customer, and consider two flows from the depot to the peak customer. The peak customer is defined to be the customer who has the highest index in the route. The model uses binary decision variable x_{ij} representing visiting j directly after i. Considering a tour that visits only j, x_{0j} can be assigned 2 in addition to 0 and 1. A binary decision variable p_i represents if i is a peak customer. To make sure that i is the maximum index in the route when $p_i = 1$, a decision variable u_i is used, which satisfies $u_i = i$ if $p_i = 1$ and $i \leq u_i \leq n-2$ otherwise. Since $n-1$ is always a peak customer, u_{n-1} is not considered. A decision variable f_{ij} represents the load of the vehicle when it visits j directly after i,

and t_i represents the total load of the route whose peak customer is i.

$$\min \sum_{i \in N, j \in N \setminus \{i\}} c_{ij} x_{ij} \tag{B.19}$$

$$\text{s.t.} \sum_{j \in N \setminus \{i\}} x_{ji} - p_i = \sum_{j \in N \setminus \{i\}} x_{ij} + p_i = 1 \qquad \forall i \in N \setminus \{0\} \tag{B.20}$$

$$t_i + \sum_{j \in N \setminus \{i\}} f_{ij} = \sum_{k \in N \setminus \{i\}} f_{ki} + q_i \qquad \forall i \in N \setminus \{0\} \tag{B.21}$$

$$d_i x_{ij} \leq f_{ij} \leq (q - d_j) x_{ij} \qquad \forall i \in N \setminus \{0\}, j \in N \setminus \{0, i\} \tag{B.22}$$

$$d_i p_i \leq t_i \leq q p_i \qquad \forall i \in N \setminus \{0\} \tag{B.23}$$

$$\sum_{i \in N \setminus \{0\}} x_{0i} = 2m \tag{B.24}$$

$$\sum_{i \in N \setminus \{0\}} p_i = m \tag{B.25}$$

$$i \leq u_i \leq i p_i + (n-2)(1 - p_i) \qquad \forall i \in N \setminus \{0, n-1\} \tag{B.26}$$

$$u_i - u_j + (n - j - 2) x_{ij} + (n - \max\{i, j\} - 2) x_{ji} \leq n - j - 2 \qquad \forall i \in N \setminus \{0, n-1\} \tag{B.27}$$

$$\sum_{i \in N : i \geq j} p_i \geq \left\lceil \sum_{i \in N : i \geq j} d_i / q \right\rceil \qquad \forall j \in N \setminus \{0\} \tag{B.28}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in N \setminus \{0\}, j \in N \setminus \{0, i\} \tag{B.29}$$

$$x_{0j} \in \{0, 1, 2\} \qquad \forall j \in N \setminus \{0\} \tag{B.30}$$

$$p_i \in \{0, 1\} \qquad \forall i \in N \setminus \{0\}. \tag{B.31}$$

Constraint (B.20) ensures that all customers, excluding the depot and peak customers, are visited only once by paths from the depot to a peak customer. Since $\sum_{j \in N \setminus \{i\}} x_{ji} = 2$ if $p_i = 1$, a peak customer is visited by two paths forming a tour. Similarly, since $\sum_{j \in N \setminus \{i\}} x_{ij} = 0$ if $p_i = 1$, no customer is visited from a peak customer. Constraints (B.21)–(B.23) are capacity constraints, considering capacity increase at each customer. Constraints (B.24) and (B.25) ensure that there are $2m$ paths and m peak customers, corresponding to m routes. Constraints (B.26) and (B.27) ensure that $p_i = 1$ iff i is the maximum index in the route visiting i. Constraint (B.28) is a valid inequality, which is redundant but improves the performance.

B.2.2 CP Model

We use the CP model proposed by Rabbouch, Saâdaoui, and Mraihi [350]. The model has interval variable x_i representing visiting customer i, optional interval variable y_{ki} representing visiting customer i using the k-th vehicle, and sequence variable π_k representing the tour of the k-th vehicle. We use x_n and y_{kn} to represent returning to the depot.

$$\min \sum_{k=1}^{m} \sum_{i \in N} c_{i,\mathsf{TypeOfNext}(\pi_k, y_{ki})} \tag{B.32}$$

$$\text{s.t. } \mathsf{NoOverlap}(\pi_k, \{c_{ij} \mid (i,j) \in N \times N\}) \qquad k = 1, ..., m \tag{B.33}$$

$$\mathsf{First}(\pi_k, y_{k0}) \qquad k = 1, ..., m \tag{B.34}$$

$$\sum_{i \in N} d_i \mathsf{Pres}(y_{ki}) \leq q \qquad k = 1, ..., m \tag{B.35}$$

$$\mathsf{Alternative}(x_i, \{y_{ki} \mid k = 1, ..., m\}) \qquad i \in N \setminus \{0\} \tag{B.36}$$

$$x_i : \mathsf{intervalVar}(1, [0, \infty)) \qquad i \in N \tag{B.37}$$

$$y_{k0} : \mathsf{intervalVar}(1, [0, \infty)) \qquad k = 1, ..., m \tag{B.38}$$

$$y_{ki} : \mathsf{optIntervalVar}(1, [0, \infty)) \qquad k = 1, ..., m, i \in N \setminus \{0\} \tag{B.39}$$

$$\pi_k : \mathsf{sequenceVar}(\{y_{ki} \mid i \in N\}) \qquad k = 1, ..., m \tag{B.40}$$

We assume that $\mathsf{TypeOfNext}(\pi_k, y_{ki})$ returns i if y_{ki} is not present and $c_{ii} = 0$. For the last interval variable in π_k, we assume $\mathsf{TypeOfNext}(\pi_k, y_{ki}) = 0$. Constraint (B.35) is the capacity constraint, where $\mathsf{Pres}(y_{ki}) = 1$ if y_{ki} presents and $\mathsf{Pres}(y_{ki}) = 0$ otherwise. Constraint (B.36) ensures that only one optional interval variable presents from $\{y_{ki} \mid k = 1, ..., m\}$, i.e., each customer is visited by one vehicle.

B.3 Multi-Commodity Pickup and Delivery TSP

In the multi-commodity pickup and delivery traveling salesperson problem (m-PDTSP) [208] introduced in Section 3.3.2, a set of customers $N = \{0, ..., n-1\}$, a set of edges $A \subseteq N \times N$, and a set of commodities $M = \{0, ..., m-1\}$ are given. Each commodity k has the weight w_k and is picked up at customer $p_k \in N$ and delivered to $d_k \in N$. The total weight of picked-up commodities must not exceed the capacity q. Visiting customer j directly after i is possible only if $(i, j) \in A$ and incurs the travel time c_{ij}. The objective is to minimize the total travel time to deliver all commodities while visiting each customer exactly once, starting from 0, and finishing at $n - 1$.

Previous work proved that some edges can be removed from A without loss of optimality [9, 15, 125, 184, 287]. The set of customers that must be visited before i is given by $P_i = \{p_k \mid k \in M : d_k = i\} \cup \{0\}$, and let $P = \{(i, j) \in N \times N \mid i \in P_j\}$. We define $P_{n-1} = N \setminus \{n-1\}$. The set of all customers that must be visited before i is given by

$$\tilde{P}_i = \{j \in N \mid j \in P_i \vee \exists k \in \tilde{P}_j, j \in \tilde{P}_k\}. \tag{B.41}$$

Similarly, the set of all customers that must be visited after i is given by

$$\tilde{S}_i = \{j \in N \mid i \in P_j \vee \exists k \in \tilde{S}_i, j \in \tilde{S}_k\}. \tag{B.42}$$

Let $\tilde{P} = \{(i,j) \in N \times N \mid i \in \tilde{P}_j\}$ and $P^- = \{(i,j) \in \tilde{P} \mid \tilde{S}_i \cap \tilde{P}_j = \emptyset\}$. In other words, P^- is the set of precedence relations that cannot be inferred from other precedence relations. It is known that edge (i,j) can be removed from A without loss of optimality if $(j,i) \in \tilde{P}$ or $(i,j) \in \tilde{P} \setminus P^-$ [9, 15, 125].

Gouveia and Ruthmair [184] proposed removing edges based on the capacity. They construct a weighted graph (N, P^-) with the following procedure:

- Initialize the graph with (N, P), where the weight of edge $(p_k, d_k) \in P$ is w_k.
- While there exists an edge $(p_k, d_k) \in P \setminus \tilde{P}$ in the graph, remove (p_k, d_k) and increase the weight of each edge in paths from p_k to d_k by w_k.

Let the weight of edge (p,d) in the resulting graph be w^-_{pd}. We can remove edge (i,j) from A if one of the following conditions is satisfied:

- $\sum_{(p,d)\in P^- \wedge p \in N\setminus\{i,j\} \wedge d \in \{i,j\}} w^-_{pd} > q$.
- $\sum_{(p,d)\in P^- \wedge ((p=i \wedge d \in N\setminus\{i\}) \vee (p \in N\setminus\{j\} \wedge d=j))} w^-_{pd} > q$.
- $\sum_{(p,d)\in P^- \wedge p \in \{i,j\} \wedge d \in N\setminus\{i,j\}} w^-_{pd} > q$.

In the MIP, CP, and DyPDL models, edges satisfying the above conditions are removed from A. In what follows, we assume that A does not contain such edges.

B.3.1 MIP Model

We use the MCF2C+IP formulation by Letchford and Salazar-González [287]. In this MIP model, binary decision variable x_{ij} represents traveling from i to j, and f^{pd}_{ij} represents traveling from i to j after visiting p and before visiting d. If any of the following conditions are satisfied, we can use $f^{pd}_{ij} = 0$ without loss of optimality.

- $i \in \tilde{P}_p \cup \tilde{S}_d \cup \{d\}$.
- $j \in \tilde{P}_p \cup \tilde{S}_d \cup \{p\}$.
- $\sum_{(k,i)\in P^-} w^-_{ki} + \sum_{(k,j)\in P^-} w^-_{kj} + w^-_{pd} > q$.
- $\sum_{(k,l)\in P^- : (k=i \wedge l \in N\setminus\{i\}) \vee (k \in N\setminus\{j\} \wedge l=j)} w^-_{kl} + w^-_{pd} > q$.
- $\sum_{(i,k)\in P^-} w^-_{ik} + \sum_{(j,k)\in P^-} w^-_{jk} + w^-_{pd} > q$.

Let F be the set of (p,d,i,j) satisfying one of the above conditions. We use $\delta_i = \sum_{k\in M: p_k = i} w_i - \sum_{k\in M: d_k = i} w_i$, i.e., the net load increase at customer i. Let π_i be the set of edges in a shortest path

from 0 to j in the graph (N, P^-), which is precomputed.

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{B.43}$$

$$\text{s.t.} \sum_{(i,j)\in A} x_{ij} = 1 \qquad \forall i \in N \setminus \{n-1\} \tag{B.44}$$

$$\sum_{(i,j)\in A} x_{ij} = 1 \qquad \forall j \in N \setminus \{0\} \tag{B.45}$$

$$\sum_{(p,j)\in A} f_{pj}^{pd} - \sum_{(j,p)\in A} f_{jp}^{pd} = 1 \qquad \forall (p,d) \in P^- \tag{B.46}$$

$$\sum_{(d,j)\in A} f_{dj}^{pd} - \sum_{(j,d)\in A} f_{jd}^{pd} = -1 \qquad \forall (p,d) \in P^- \tag{B.47}$$

$$\sum_{(i,j)\in A} f_{ij}^{pd} - \sum_{(j,i)\in A} f_{ji}^{pd} = 0 \qquad \forall (p,d) \in P^-, \forall i \in P^- \setminus \{p,d\} \tag{B.48}$$

$$0 \le f_{ij}^{pd} \le x_{ij} \qquad \forall (p,d) \in P^-, (i,j) \in A \tag{B.49}$$

$$\sum_{(p,d)\in P^-} w_{pd}^- f_{ij}^{pd} \le (q - \max\{0, -\delta_i, \delta_j\})x_{ij} \qquad \forall a \in A \tag{B.50}$$

$$\sum_{(p,d)\in\pi_j} \sum_{(i,k)\in A} f_{ik}^{pd} + \sum_{(p,d)\in\pi_i} \sum_{(j,k)\in A} f_{jk}^{pd} = 1 \quad \forall i,j \in N \setminus \{0, n-1\}, i \ne j, (i,j) \notin \tilde{P}, (j,i) \notin \tilde{P} \tag{B.51}$$

$$f_{ij}^{pd} = 0 \qquad \forall (p,d,i,j) \in F \tag{B.52}$$

$$x_{ij} \in \{0,1\} \qquad (i,j) \in A. \tag{B.53}$$

Constraints (B.46)–(B.49) ensure pickup and delivery while eliminating subtours. Constraint (B.50) is an enhanced version of the capacity constraint. Constraint (B.51) is a valid inequality.

B.3.2 CP Model

We use the CP model proposed by Castro, Cire, and Beck [65]. The model uses interval variable x_i representing visiting customer i. Let u be an upper bound on the objective cost.

$$\min \ \mathsf{StartOf}(x_{n-1}) - n + 1 \tag{B.54}$$

$$\text{s.t. } \mathsf{NoOverlap}(\pi, \{c_{ij} \mid (i,j) \in N \times N\}) \tag{B.55}$$

$$\mathsf{Before}(\pi, x_i, x_j) \qquad \forall j \in N, i \in P_j \tag{B.56}$$

$$\sum_{i \in N} \mathsf{StepAtStart}(x_i, \delta_i) \leq q \tag{B.57}$$

$$\mathsf{First}(\pi, x_0) \tag{B.58}$$

$$\mathsf{Last}(\pi, x_{n-1}) \tag{B.59}$$

$$x_i : \mathsf{intervalVar}(1, [0, u+n]) \qquad i \in N \tag{B.60}$$

$$\pi : \mathsf{sequenceVar}(\{x_i \in N\}). \tag{B.61}$$

In Objective (B.54), $\mathsf{StartOf}(x_{n-1})$ represents the time when x_{n-1} starts, i.e., when $n-1$ is visited. In Constraint (B.55), we use $c_{ii} = 0$ and $c_{ij} = u + n$ if $(i,j) \notin A$. Constraint (B.56) ensures that x_i presents before x_j in π if $i \in P_j$. Constraint (B.57) is the capacity constraint, representing that the load increases by δ_i at customer i and must not exceed the capacity q. Constraint (B.59) ensures that $n-1$ is visited at the end. In our implementation, we use $u = \sum_{i=0}^{n} \max_{j \in N:(i,j) \in A} c_{ij}$.

B.4 Orienteering Problem with Time Windows

In the orienteering problem with time windows (OPTW) [238] introduced in Section 3.3.3, we are given a set of customers $N = \{0, ..., n-1\}$, the travel time c_{ij} from customer i to j, and the profit p_i of customer i. A customer can be visited within the time window $[a_i, b_i]$. We start from the depot (0) and need to return to the depot by b_0. The objective is to maximize the total profit of customers visited.

B.4.1 MIP Model

We use the MIP model based on Vansteenwegen, Souffriau, and Oudheusden [433]. This model uses binary decision variable x_{ij} that represents visiting j from i and decision variable t_i that represents the time i is visited. We introduce x_{in} that represents returning to the depot from i and t_n that

represents the time to return to the depot, using $c_{i0} = c_{in}$.

$$\max \sum_{i \in N \setminus \{0\}} \sum_{j \in (N \setminus \{0,i\}) \cup \{n\}} p_i x_{ij} \tag{B.62}$$

$$\text{s.t.} \sum_{i \in (N \setminus \{0\}) \cup \{n\}} x_{0i} = \sum_{i \in N} x_{in} = 1 \tag{B.63}$$

$$\sum_{i \in N \setminus \{j\}} x_{ij} \leq 1 \qquad \forall j \in (N \setminus \{0\}) \cup \{n\} \tag{B.64}$$

$$\sum_{j \in (N \setminus \{0,i\}) \cup \{n\}} x_{ij} \leq 1 \qquad \forall i \in N \tag{B.65}$$

$$t_i + c_{ij} - t_j \leq M(1 - x_{ij}) \qquad \forall i \in N, j \in (N \setminus \{0,i\}) \cup \{n\} \tag{B.66}$$

$$0 \leq t_n \leq b_0 \tag{B.67}$$

$$a_i \leq t_i \leq b_i \qquad \forall i \in N \tag{B.68}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in N, j \in (N \setminus \{0,i\}) \cup \{n\} \tag{B.69}$$

where M is an upper bound of the left-hand side in Constraint (B.66). In our implementation, we use $M = b_i + c_{ij} - a_j$. Differently from Vansteenwegen, Souffriau, and Oudheusden, we do not have a separate time budget from b_0.

B.4.2 CP Model

In our CP model, we define an optional interval variable x_i that represents visiting customer i in $[a_i, b_i]$. We also introduce an interval variable x_n that represents returning to the depot (0) and define $c_{ii} = 0$, $c_{in} = c_{i0}$, and $c_{ni} = c_{0i}$ for each $i \in N$.

$$\max \sum_{i \in N \setminus \{0\}} p_i \mathsf{Pres}(x_i) \tag{B.70}$$

$$\text{s.t. } \mathsf{NoOverlap}(\pi, \{c_{ij} \mid (i,j) \in (N \cup \{n\}) \times (N \cup \{n\})\}) \tag{B.71}$$

$$\mathsf{First}(\pi, x_0) \tag{B.72}$$

$$\mathsf{Last}(\pi, x_n) \tag{B.73}$$

$$x_i : \mathsf{optIntervalVar}(0, [a_i, b_i]) \qquad \forall i \in N \setminus \{0\} \tag{B.74}$$

$$x_0 : \mathsf{intervalVar}(0, [0,0]) \tag{B.75}$$

$$x_n : \mathsf{intervalVar}(0, [0, b_0]) \tag{B.76}$$

$$\pi : \mathsf{sequenceVar}(\{x_0, ..., x_n\}). \tag{B.77}$$

B.5 Multi-Dimensional Knapsack Problem

In the multi-dimensional knapsack problem (MDKP) [306, 249] introduced in Section 3.3.4, we are given a set of items N, the profit p_i of item i, the capacity q_j of the knapsack in dimension $j = 0, ..., m-1$, and the weight w_{ij} of item i in dimension j. The objective is to maximize the total profit of items included in the knapsack.

B.5.1 MIP Model

We use the MIP model described in Cacchiani et al. [61]. The model has binary decision variable x_i representing whether item i is included in the knapsack or not.

$$\max \sum_{i \in N} p_i x_i \tag{B.78}$$

$$\text{s.t.} \sum_{i \in N} w_{ij} x_i \leq q_j \qquad j = 0, ..., m-1 \tag{B.79}$$

$$x_i \in \{0, 1\} \qquad \forall i \in N. \tag{B.80}$$

B.5.2 CP Model

In our CP model, we use the Pack global constraint [391] and consider packing all items into two bins; one represents the knapsack, and the other represents items not selected. We introduce a binary variable x_i representing the bin where item i is packed ($x_i = 0$ represents that the item is in the knapsack). We define an integer variable y_{j0} representing the total weight of the items in the knapsack in dimension j and y_{j1} representing the total weight of the items not selected.

$$\max \sum_{i \in N} p_i (1 - x_i) \tag{B.81}$$

$$\text{s.t. } \mathsf{Pack}(\{y_{j0}, y_{j1}\}, \{x_i \mid i \in N\}, \{w_{ij} \mid i \in N\}) \qquad j = 0, ..., m-1 \tag{B.82}$$

$$y_{j0} \leq q_j \qquad j = 0, ..., m-1 \tag{B.83}$$

$$y_{j0}, y_{j1} \in \mathbb{Z}_0^+ \qquad j = 0, ..., m-1 \tag{B.84}$$

$$x_i \in \{0, 1\} \qquad \forall i \in N. \tag{B.85}$$

B.6 Bin Packing

In bin packing [306] introduced in Section 3.3.5, we are given a set of items $N = \{0, ..., n-1\}$, the weight w_i of each item $i \in N$, and the capacity of a bin q. The objective is to minimize the number of bins to pack all items. In the MIP and CP models, we compute the upper bound $\bar{m}$ on the number of bins using the first-fit decreasing (FFD) heuristic and use $M = \{0, ..., \hat{m} - 1\}$. FFD selects an item in a non-decreasing order of weights and packs it in the first bin where it fits. If there is no such bin, FFD opens a new bin.

B.6.1 MIP Model

We use the MIP model described in Delorme, Iori, and Martello [99], which extends a model in Martello and Toth [306] with symmetry breaking. In this model, binary decision variable x_{ij} represents that item i is included in bin j, and y_j represents whether bin j is opened. We define x_{ij} only

for $i \leq j$, i.e., item i is packed in bin j or earlier, to break the symmetry.

$$\min \sum_{j \in M} y_j \tag{B.86}$$

$$\text{s.t.} \sum_{i \in N} w_i x_{ij} \leq q y_j \qquad \forall j \in M \tag{B.87}$$

$$\sum_{j \in M : i \leq j} x_{ij} = 1 \qquad \forall i \in N \tag{B.88}$$

$$y_j \in \{0, 1\} \qquad \forall j \in M \tag{B.89}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in N, j \in M, i \leq j. \tag{B.90}$$

B.6.2 CP Model

In our CP model, we use x_i to represent the index of the bin where item i is packed and y_j to represent the total weight of items packed in bin j. We use Pack and ensure that item i is packed in bin i or earlier.

$$\min \max_{i \in N} x_i + 1 \tag{B.91}$$

$$\text{s.t. } \mathsf{Pack}(\{y_j \mid j \in M\}, \{x_i \mid i \in N\}, \{w_i \mid i \in N\}) \tag{B.92}$$

$$0 \leq y_j \leq q \qquad \forall j \in M \tag{B.93}$$

$$0 \leq x_i \leq i \qquad \forall i \in N \tag{B.94}$$

$$y_j \in \mathbb{Z} \qquad \forall j \in M \tag{B.95}$$

$$x_i \in \mathbb{Z} \qquad \forall i \in N. \tag{B.96}$$

B.7 Simple Assembly Line Balancing Problem

In the simple assembly line balancing problem (SALBP-1) [374, 22] introduced in Section 3.3.6, in addition to tasks N and the capacity q of a station as in bin packing, we are given P_i, the set of the predecessors of task i. A task has weight w_i, and the total weight of tasks scheduled in one station must not exceed q. The objective is to minimize the number of stations to schedule all tasks satisfying the precedence constraints. In the MIP and CP Model, we use $\bar{m} = \min\{n, 2\lceil \sum_{i \in N} w_i / q \rceil\}$ as an upper bound on the number of stations, following Ritt and Costa [363], and use $M = \{0, ..., \bar{m} - 1\}$. We also use the set of all direct and indirect predecessors of task i, $\tilde{P}_i = \{j \in N \mid j \in P_i \vee \exists k \in \tilde{P}_j, j \in \tilde{P}_k\}$. Similarly, the set of all direct and indirect successors of task i is $\tilde{S}_i = \{j \in N \mid i \in P_j \vee \exists k \in \tilde{S}_i, j \in \tilde{S}_k\}$.

B.7.1 MIP Model

We use the NF4 formulation by Ritt and Costa [363]. In this model, binary decision variable x_{ij} represents that task i is scheduled in station j, and y_j represents whether station j is opened. Let e_i be the index of the earliest station and l_i be the index of the latest station where task i can be scheduled. Then, x_{ij} is defined only for i and j such that $e_i \leq j \leq l_i$. We also use l_{ij}, the latest

possible station where task i can be scheduled when we use at most j stations.

$$\min \sum_{j \in M} y_j \tag{B.97}$$

$$\sum_{i \in N : e_i \le j \le l_i} w_i x_{ij} \le q y_j \qquad \forall j \in M \tag{B.98}$$

$$\sum_{j=e_i}^{l_i} x_{ij} = 1 \qquad \forall i \in N \tag{B.99}$$

$$\sum_{k=e_p : k \le j}^{l_p} x_{pk} \ge \sum_{k=e_i : k \le j}^{l_i} x_{ik} \qquad \forall i \in N, p \in P_i, j \in M \tag{B.100}$$

$$\sum_{k=e_i : k \ge l_{ij}}^{l_i} x_{ik} \le y_j \qquad \forall i \in N, j \in M \tag{B.101}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in N, j \in M, e_i \le j \le l_i. \tag{B.102}$$

Constraint (B.100) is the precedence constraint. Constraint (B.101) is a valid inequality. Following Ritt and Costa, we use $e_i = \left\lceil \frac{w_i + \sum_{j \in \tilde{P}_i} w_j}{q} \right\rceil$, $l_i = \overline{m} + 1 - \left\lceil \frac{w_i + \sum_{j \in \tilde{S}_i} w_j}{q} \right\rceil$, and $l_{ij} = j + 1 - \left\lceil \frac{w_i + \sum_{k \in \tilde{S}_i} w_k}{q} \right\rceil$.

B.7.2 CP Model

We implement the CP model proposed by Bukchin and Raviv [54] with the addition of Pack. Let m be the number of stations, x_i be the index of the station of task i, and y_j be the sum of the weights of tasks scheduled in station j. We use $e_i = \left\lceil \frac{w_i + \sum_{k \in \tilde{P}_i} w_k}{q} \right\rceil$, a lower bound on the number of stations required to schedule task i, $l_i = \left\lfloor \frac{w_i - 1 + \sum_{k \in \tilde{S}_i} w_k}{q} \right\rfloor$, a lower bound on the number of stations between the station of task i and the last station, and $d_{ij} = \left\lfloor \frac{w_i + w_j - 1 + \sum_{k \in \tilde{S}_i \cap \tilde{P}_j} w_k}{q} \right\rfloor$, a lower bound on the number of stations between the stations of tasks i and j.

$$\min m \tag{B.103}$$

$$\text{s.t. } \mathsf{Pack}(\{y_j \mid j \in M\}, \{x_i \mid i \in N\}, \{w_i \mid i \in N\}) \tag{B.104}$$

$$0 \le y_j \le q \qquad \forall j \in M \tag{B.105}$$

$$e_i - 1 \le x_i \le m - 1 - l_i \qquad \forall i \in N \tag{B.106}$$

$$x_i + d_{ij} \le x_j \qquad \forall j \in N, \forall i \in \tilde{P}_j, \nexists k \in \tilde{S}_i \cap \tilde{P}_j : d_{ij} \le d_{ik} + d_{kj} \tag{B.107}$$

$$m \in \mathbb{Z} \tag{B.108}$$

$$y_j \in \mathbb{Z} \qquad \forall j \in M \tag{B.109}$$

$$x_i \in \mathbb{Z} \qquad \forall i \in N. \tag{B.110}$$

Constraint (B.106) states the lower and upper bounds on the index of the station of i. Constraint (B.107) is an enhanced version of the precedence constraint using d_{ij}. In the original paper, $\overline{m}$ is computed by a heuristic.

B.8 Single Machine Total Weighted Tardiness

In single machine total weighted tardiness ($1||\sum w_i T_i$) [123] introduced in Section 3.3.7, we are given a set of jobs N, the processing time p_i of job i, the deadline d_i for i, the weight w_i for i, and the set P_i of jobs processed before i, extracted by preprocessing in Kanet [237]. We schedule jobs in a single machine, and the objective is to minimize the total weighted tardiness, which is computed as $\sum_{i \in N} w_i \max\{C_i - d_i, 0\}$, where C_i is the completion time of job i.

B.8.1 MIP Model

For MIP, we use the F4 formulation, which is based on using assignment and positional date variables [248]. In this model, binary decision variable u_{ik} represents whether job $i \in N$ is assigned to position $k \in \{0, ..., |N| - 1\}$, i.e., i is the k-th job in the schedule if $u_{ik} = 1$. Decision variable γ_k represents the completion time of the k-th job, C_i represents the completion time of job i, and T_i represents the tardiness of job i.

$$\min \sum_{i \in N} w_i T_i \tag{B.111}$$

$$\text{s.t.} \sum_{k=0}^{|N|-1} u_{ik} = 1 \qquad \forall i \in N \tag{B.112}$$

$$\sum_{i \in N} u_{ik} = 1 \qquad k = 0, ..., |N| - 1 \tag{B.113}$$

$$\gamma_0 \geq \sum_{i \in N} p_i u_{i0} \tag{B.114}$$

$$\gamma_k \geq \gamma_{k-1} + \sum_{i \in N} p_i u_{ik} \qquad k = 1, ..., |N| - 1 \tag{B.115}$$

$$C_i \geq \gamma_k - M(1 - u_{ik}) \qquad \forall i \in N, k = 0, ..., |N| - 1 \tag{B.116}$$

$$T_i \geq C_i - d_i \qquad \forall i \in N \tag{B.117}$$

$$\gamma_k \geq 0 \qquad k = 0, ..., |N| - 1 \tag{B.118}$$

$$C_i, T_i \geq 0 \qquad \forall i \in N \tag{B.119}$$

$$u_{ik} \in \{0, 1\} \qquad \forall i \in N, k = 0, ..., |N| - 1 \tag{B.120}$$

where M in Constraint (B.116) is an upper bound on γ_k. We use $M = \sum_{i \in N} p_i$. Constraints (B.112) and (B.113) ensure that each job is assigned a position, and each position is assigned to a job. Constraints (B.114) and (B.115) ensure that γ_k is the completion time of position k. Constraints (B.116) and (B.119) ensure that C_i is the completion time of job i since $C_i \geq \gamma_k$ if $u_{ik} = 1$ and $C_i \geq 0$ otherwise. Constraints (B.117) and (B.119) ensure that T_i is the tardiness of job i. While valid inequalities are proposed in the original paper, we do not use them since the performance was degraded in our preliminary experiment. We do not use precedence constraints extracted by preprocessing for

the same reason.

B.8.2 CP Model

We use an interval variable x_i with the duration p_i and the starting time within $[0, \sum_{j \in N} p_j]$, representing the time interval when job i is processed.

$$\min \sum_{i \in N} w_i \max\{\mathsf{EndOf}(x_i) - d_i, 0\} \tag{B.121}$$

$$\text{s.t. } \mathsf{NoOverlap}(\pi) \tag{B.122}$$

$$\mathsf{Before}(\pi, x_i, x_j) \qquad \forall j \in N, i \in P_j \tag{B.123}$$

$$x_i : \mathsf{intervalVar}\left(p_i, \left[0, \sum_{j \in N} p_j\right]\right) \qquad \forall i \in N \tag{B.124}$$

$$\pi : \mathsf{seqeunceVar}(\{x_i \mid i \in N\}). \tag{B.125}$$

In Objective (B.121), $\mathsf{EndOF}(x_i)$ is the time when x_i ends in π.

B.9 Talent Scheduling

In talent scheduling [74] introduced in Section 3.3.8, we are given a set of scenes $N = \{0, ..., n-1\}$, a set of actors $A = \{0, ..., m-1\}$, the set of actors $A_s \subseteq A$ playing in scene s, and the set of scenes $N_a \subseteq N$ where actor a plays. A scene s has duration d_s. An actor a is on location from the first to the last days that require a. We need to pay the cost c_a per day when actor a is on location. The objective is to minimize the total cost of shooting all scenes.

Garcia de la Banda, Stuckey, and Chu [153] proposed the following preprocessing methods to reduce the size of an instance:

- Remove actor a if a plays in only one scene s. The total cost for the original problem is reconstructed by adding $d_s c_a$ to the total cost of the reduced problem.
- Merge two scenes s and s' with $a_s = a_{s'}$ into a single scene s'' with $a_{s''} = a_s$ and $d_{s''} = d_s + d_{s'}$.

The MIP, CP, and DyPDL models use the above preprocessing methods.

B.9.1 MIP Model

We use the MIP model proposed by Qin et al. [349]. In this model, binary decision variable x_{sr} represents if scene r is shot immediately after scene s. We introduce dummy scenes -1 and n representing the first and last scenes. Integer decision variable t_s represents the starting day of scene s, e_a represents the day actor a comes to the location, and l_a represents the day actor a leaves

the location. An auxiliary variable z_{sr} represents $t_r x_{sr}$.

$$\min \sum_{a \in A} c_a (l_a - e_a + 1) \tag{B.126}$$

$$\text{s.t.} \sum_{s \in N} x_{-1,s} = \sum_{s \in N} x_{sn} = 1 \tag{B.127}$$

$$\sum_{r \in (N \setminus \{s\}) \cup \{n\}} x_{sr} = \sum_{(r \in N \setminus \{s\}) \cup \{-1\}} x_{rs} = 1 \qquad \forall s \in N \tag{B.128}$$

$$t_{-1} = 0 \tag{B.129}$$

$$t_n = \sum_{s \in N} d_s + 1 \tag{B.130}$$

$$\sum_{r \in N \setminus \{s\}} z_{sr} = t_s + d_s \qquad \forall s \in N \cup \{-1\} \tag{B.131}$$

$$e_a \leq t_s \qquad \forall s \in N, a \in A_s \tag{B.132}$$

$$t_s + d_s - 1 \leq l_a \qquad \forall s \in N, a \in A_s \tag{B.133}$$

$$0 \leq z_{sr} \leq t_r \qquad \forall s \in N \cup \{-1\}, r \in N \setminus \{s\} \tag{B.134}$$

$$t_r + M(x_{sr} - 1) \leq z_{sr} \leq M x_{sr} \qquad \forall s \in N \cup \{-1\}, r \in N \setminus \{s\} \tag{B.135}$$

$$x_{sr} \in \{0, 1\} \qquad \forall s \in N \cup \{-1\}, r \in (N \setminus \{s\}) \cup \{n\} \tag{B.136}$$

$$e_a, l_a \in \mathbb{Z}_0^+ \qquad \forall a \in A \tag{B.137}$$

$$t_s \in \mathbb{Z}_0^+ \qquad \forall s \in N \cup \{-1, n\} \tag{B.138}$$

where M in Constraint (B.135) is an upper bound on $t_r x_{sr}$. We use $M = \sum_{s \in N} d_s$. Constraints (B.127) and (B.128) ensure that all scenes are shot, and each scene is shot once. Constraints (B.129)–(B.131) ensure that t_s is the day when s is shot. Constraints (B.132) and (B.133) ensure that actor a is on location from day e_a to day l_a, which are used in Objective (B.126). Constraints (B.134) and (B.135) ensure that $z_{sr} = t_s x_{sr}$.

B.9.2 CP Model

We extend the model used by Chu and Stuckey [78], originally implemented with MiniZinc [326], with the AllDifferent global constraint [280]. Let x_k be a variable representing the k-th scene in the schedule, b_{sk} be a variable representing if scene s is shot before the k-th scene, o_{ak} be a variable representing if any scene in N_a is shot by the k-th scene, and f_{ak} be a variable representing if all

scenes in N_a finish before the k-th scene. The CP model is

$$\min \sum_{k=0}^{|N|-1} d_{x_k} \sum_{a \in A} c_a o_{ak}(1 - f_{ak}) \tag{B.139}$$

$$\text{s.t. } \mathsf{AllDifferent}(\{x_k \mid k = 0, ..., |N| - 1\}) \tag{B.140}$$

$$b_{s0} = 0 \qquad \forall s \in N \tag{B.141}$$

$$b_{sk} = b_{s,k-1} + \mathbb{1}(x_{k-1} = s) \qquad \forall s \in N, k = 1, ..., |N| - 1 \tag{B.142}$$

$$b_{sk} = 1 \rightarrow x_k \neq s \qquad \forall s \in N, k = 1, ..., |N| - 1 \tag{B.143}$$

$$o_{a0} = \mathbb{1}\left(\bigvee_{s \in N_a} x_0 = s\right) \qquad \forall a \in A \tag{B.144}$$

$$o_{ak} = \mathbb{1}\left(o_{a,k-1} = 1 \vee \bigvee_{s \in N_a} x_k = s\right) \qquad \forall a \in A, k = 1, ..., |N| - 1 \tag{B.145}$$

$$f_{ak} = \prod_{s \in N_a} b_{sk} \qquad \forall a \in A, k = 0, ..., |N| - 1 \tag{B.146}$$

$$x_k \in N \qquad k = 0, ..., |N| - 1 \tag{B.147}$$

$$b_{sk}, f_{sk} \in \{0, 1\} \qquad \forall s \in N, k = 0, ..., |N| - 1 \tag{B.148}$$

where $\mathbb{1}$ is a function that returns 1 if a given condition holds and 0 otherwise. In Objective (B.139), $o_{ak}(1 - f_{ak})$ represents if actor a is on location when the k-th scene is shot. Constraint (B.140) (AllDifferent) is redundant, but it slightly improved the performance in our preliminary experiment. Other constraints logically declare the relationships between the decision variables.

B.10 Minimization of Open Stacks Problem

In the minimization of open stacks problem (MOSP) [446] introduced in Section 3.3.9, a set of customers C and a set of products P are given, and each customer c orders a subset of products $P_c \subseteq P$. We decide the order in which products are produced. The stack for customer c is opened when product $p \in P_c$ is produced and is closed when all products in P_c are produced. The objective is to minimize the maximum number of open stacks at a time.

B.10.1 MIP Model

We use the MOSP-ILP-I formulation proposed by Martin, Yanasse, and Pinto [307]. Similar to the DyPDL model, this model considers the order in which stacks are closed. Let $N_c = \{c' \in C \mid P_c \cap P_{c'} \neq \emptyset\}$ be the set of customers that order the same product as c. Binary decision variable x_{ck} represents that stack for customer c is closed at time step k, and y_{ck} represents that stack for c has been opened (and was possibly closed) up to k. Integer decision variable V represents the objective. Let l be a lower bound on the objective. After closing $|N| - l$ stacks, since only l stacks are remaining, the order in which these stacks are closed does not change the objective value. Therefore, the model considers closing all remaining stacks at time step $|N| - l$, and x_{ck} and y_{ck} are defined

for $k = 0, ..., |N| - l$.

$$\min V \tag{B.149}$$

$$\text{s.t.} \sum_{c \in C} x_{ck} = 1 \qquad k = 0, ..., |N| - l - 1 \tag{B.150}$$

$$\sum_{c \in C} x_{c,|N|-l} = l \tag{B.151}$$

$$\sum_{k=0}^{|N|-l} x_{ck} = 1 \qquad \forall c \in C \tag{B.152}$$

$$\sum_{d \in N_c} \sum_{i=0}^{k} x_{di} \leq \min\{k, |N_c|\} y_{ck} \qquad \forall c \in C, k = 0, ..., |N| - l - 1 \tag{B.153}$$

$$y_{c,|N|-l} = 1 \qquad \forall c \in C \tag{B.154}$$

$$V \geq \sum_{c \in C} y_{ck} - k \qquad k = 0, ..., |N| - l \tag{B.155}$$

$$x_{ck} \in \{0, 1\} \qquad \forall c \in C, k = 0, ..., |N| - l \tag{B.156}$$

$$y_{ck} \in \{0, 1\} \qquad \forall c \in C, k = 0, ..., |N| - l \tag{B.157}$$

$$V \in \mathbb{Z}_0^+. \tag{B.158}$$

Constraints (B.150) and (B.151) ensure that one stack is closed at time step $t \leq |N| - l - 1$, and l stacks are closed at $t = |N| - l$. Constraint (B.152) ensures that each stack is closed only once. Constraints (B.153) and (B.154) ensure that $y_{ct} = 1$ if stack for customer c has been opened up to t. Constraint (B.155) ensure that V is the maximum number of open stacks at a time since $\sum_{c \in C} y_{ck}$ stacks have been opened and k stacks have been closed at time step k. Following Martin, Yanasse, and Pinto, we use $l = \min_{c \in C} |N_C|$.

B.10.2 CP Model

Martin, Yanasse, and Pinto [307] also proposed a CP model. The model considers the order in which products are produced and uses interval variable x_p representing that product p is produced. It also uses interval variable y_c representing the duration when the stack for customer c is opened, which has a variable length in $[|P_c|, |P|]$. The objective is represented by integer decision variable V.

$$\min V \tag{B.159}$$

$$\text{s.t. } \mathsf{NoOverlap}(\pi) \tag{B.160}$$

$$\mathsf{Span}(y_c, \{x_p \mid p \in P_c\}) \qquad \forall c \in C \tag{B.161}$$

$$V \geq \sum_{c \in C} \mathsf{Pulse}(y_c, 1) \tag{B.162}$$

$$x_p : \mathsf{intervalVar}(1, [0, |P|]) \qquad \forall p \in P \tag{B.163}$$

$$y_c : \mathsf{intervalVar}([|P_c|, |P|], [0, |P|]) \qquad \forall c \in C \tag{B.164}$$

$$\pi : \mathsf{sequenceVar}(\{x_p \mid p \in P\}) \tag{B.165}$$

$$V \in \mathbb{Z}_0^+. \tag{B.166}$$

Constraints (B.160), (B.163), and (B.165) ensure that $\{x_p \mid p \in P\}$ are ordered in one sequence, and each interval variable has the duration of 1, starts in $[0, |P|-1]$, and finishes in $[1, |P|]$. In contrast, the interval variables $\{y_c \mid c \in C\}$ defined in Constraint (B.164) can overlap and have variable durations. Constraint (B.161) ensures that y_c covers interval variables $\{x_p \mid p \in P_c\}$, i.e., the stack for customer c is open during products in P_c are produced. In Constraint (B.162), $\mathsf{Pulse}(y_c, 1)$ states that the right-hand side increases by 1 during the time duration covered by y_c. Thus, the maximum value of the right-hand side over time corresponds to the maximum number of open stacks at a time.

B.11 Graph-Clear

In graph-clear [256] introduced in Section 3.3.10, an undirected graph (N, E) is given, where the weight of node $i \in N$ is a_i, and the weight of edge $\{i, j\} \in E$ is b_{ij}. Initially, all nodes are contaminated. In each step, one node i is swept by a_i robots, and each edge $\{i, j\}$ connected to i must be blocked by b_{ij} robots. A node becomes clean when it is swept, but it becomes contaminated again if any path connecting the node and a contaminated node is not blocked. The objective is to make all nodes clean while minimizing the maximum number of robots used at a time. As described in Section 3.3.10, there exists an optimal solution where already swept nodes never become contaminated again [317]. In such an optimal solution, given a set of swept nodes C, to clean node c, we need to use $a_c + \sum_{i \in N : \{c,i\} \in E} b_{ci} + \sum_{i \in C} \sum_{j \in (N \setminus C) \setminus \{c\} : \{i,j\} \in E} b_{ij}$ robots to block all edges connecting swept nodes and contaminated nodes. The MIP, CP, and DyPDL models are based on this observation. In the MIP and CP models proposed by Morin et al. [317], the objective is represented by a decision variable Z, and lower and upper bounds $\underline{Z}$ and $\overline{Z}$ are used. Following Morin et al., we use $\underline{Z} = 1$ and $\overline{Z} = \max_{c \in N} a_c + \sum_{\{i,j\} \in E} b_{ij}$.

B.11.1 MIP Model

In the MIP model proposed by Morin et al. [317], binary decision variable x_{ik} represents if node i is clean at time step k, and y_{ijk} represents if edge $\{i,j\}$ is blocked at time step k.

$$\min Z \tag{B.167}$$

$$\text{s.t. } Z \geq \sum_{i \in N} a_i x_{i0} + \sum_{\{i,j\} \in E} b_{ij} y_{ij0} \tag{B.168}$$

$$Z \geq \sum_{i \in N} a_i (x_{ik} - x_{i,k-1}) + \sum_{\{i,j\} \in E} b_{ij} y_{ijk} \qquad k = 1, ..., |N| - 1 \tag{B.169}$$

$$\sum_{i \in N} x_{ik} = k \qquad k = 0, ..., |N| - 1 \tag{B.170}$$

$$x_{ik} \leq x_{i,k+1} \qquad \forall i \in N, k = 0, ..., |N| - 2 \tag{B.171}$$

$$x_{ik} - x_{jk} \leq y_{ijk} \qquad \forall \{i,j\} \in E, k = 0, ..., |N| - 1 \tag{B.172}$$

$$x_{jk} - x_{ik} \leq y_{ijk} \qquad \forall \{i,j\} \in E, k = 0, ..., |N| - 1 \tag{B.173}$$

$$x_{ik} - x_{i,k-1} \leq y_{ijk} \qquad \forall \{i,j\} \in E, k = 1, ..., |N| - 1 \tag{B.174}$$

$$x_{ik} \in \{0,1\} \qquad \forall i \in N, k = 0, ..., |N| - 1 \tag{B.175}$$

$$y_{ijt} \in \{0,1\} \qquad \forall \{i,j\} \in E, k = 0, ..., |N| - 1 \tag{B.176}$$

$$\underline{Z} \leq Z \leq \overline{Z}. \tag{B.177}$$

Constraints (B.169) and (B.170) ensure that the objective is to minimize the maximum number of robots used at a time. Constraint (B.170) ensures that one node is swept at each time step. Constraint (B.171) states that a swept node never becomes contaminated again. Constraints (B.172) and (B.173) ensure that edges connecting clean and contaminated nodes are blocked, and Constraint (B.174) ensures that edges connected to the node being swept are blocked.

B.11.2 Node-Based CP Model (CPN)

Since edges connecting clean and contaminated nodes are blocked, an edge $\{i,j\}$ starts to be blocked when either i or j is swept, and it is released after the remaining one is swept. Since an already swept node never becomes contaminated again, the edge will never be blocked again after released. Based on this observation, in the node-based CP model (CPN) proposed by Morin et al. [317], integer decision variable l_{ij} represents the time step when $\{i,j\}$ starts to be blocked, and u_{ij} represents the time step when $\{i,j\}$ is released. Integer decision variable t_i represents the time step at which node i is swept. In addition, integer decision variable s_k represents the cost to sweep a node, r_k represents the cost to block edges that are not connected to the node being swept at time step k. To define domains of these variables, the maximum cost to sweep a node, $\overline{s}$, and the maximum cost to block edges, $\overline{r}$, are used. Binary decision variable y_{ijk} represents if edge $\{i,j\}$ is blocked at time step k

while i and j are not being swept.

$$\min Z \tag{B.178}$$

$$\text{s.t. } Z = \max_{k=0,...,|N|-1} s_k + r_k \tag{B.179}$$

$$t_i = k \rightarrow s_k = a_i + \sum_{j \in N:\{i,j\} \in E} b_{ij} \qquad \forall i \in N, k = 0, ..., |N| - 1 \tag{B.180}$$

$$\mathsf{AllDifferent}(\{t_i \mid i \in N\}) \tag{B.181}$$

$$t_i < t_j \rightarrow l_{ij} = t_i \qquad \forall \{i,j\} \in E \tag{B.182}$$

$$t_i < t_j \rightarrow u_{ij} = t_j \qquad \forall \{i,j\} \in E \tag{B.183}$$

$$(l_{ij} \leq k \wedge u_{ij} \geq k) \wedge (t_i \neq k \wedge t_j \neq k) \rightarrow y_{ijk} = 1 \qquad \forall \{i,j\} \in E, k = 0, ..., |N| - 1 \tag{B.184}$$

$$r_k = \sum_{\{i,j\} \in E} b_{ij} y_{ijt} \qquad k = 0, ..., |N| - 1 \tag{B.185}$$

$$t_i \in \{0, ..., |N| - 1\} \qquad \forall i \in N \tag{B.186}$$

$$l_{ij}, u_{ij} \in \{0, ..., |N| - 1\} \qquad \forall \{i,j\} \in E \tag{B.187}$$

$$s_k \in \{1, ..., \overline{s}\} \qquad k = 0, ..., |N| - 1 \tag{B.188}$$

$$r_k \in \{1, ..., \overline{r}\} \qquad k = 0, ..., |N| - 1 \tag{B.189}$$

$$y_{ijt} \in \{0, 1\} \qquad k = 0, ..., |N| - 1 \tag{B.190}$$

$$\underline{Z} \leq Z \leq \overline{Z} \tag{B.191}$$

$$Z \in \mathbb{Z}^+. \tag{B.192}$$

Constraint (B.179) ensures that Z is the objective. Constraint (B.180) ensures that s_k is the cost to sweep node i at time step k. Constraints (B.182) and (B.183) ensure that l_{ij} is the time step when edge $\{i,j\}$ starts to be blocked, and u_{ij} is the time step when $\{i,j\}$ is released. Constraint (B.184) ensures that $y_{ijk} = 1$ if $\{i,j\}$ is blocked while i and j are not being swept. Constraint (B.185) ensures that r_k is the cost to block edges that are not connected to the node being swept at k. Following Morin et al., we use $\overline{s} = \max_{c \in N} a_c + \sum_{i \in N:\{c,i\} \in E} b_{ci}$ and $\overline{r} = \sum_{\{i,j\} \in E} b_{ij}$.

B.11.3 Sequence-Based CP Model (CPS)

In the sequence-based CP model (CPS) proposed by Morin et al. [317], integer decision variable x_k represents the node swept at time step k.

$$\min Z \tag{B.193}$$

$$\text{s.t. } \mathsf{AllDifferent}(\{x_k \mid k = 0, ..., |N| - 1\}) \tag{B.194}$$

$$Z \geq a_{x_k} + \sum_{i \in N:\{x_k,i\} \in E} b_{x_k i} + \sum_{i \in C} \sum_{j \in (N \setminus C) \setminus \{x_k\}:\{i,j\} \in E} b_{ij} \qquad k = 0, ..., |N| - 1 \tag{B.195}$$

$$x_k \in N \qquad k = 0, ..., |N| - 1 \tag{B.196}$$

$$\underline{Z} \leq Z \leq \overline{Z} \tag{B.197}$$

$$Z \in \mathbb{Z}^+. \tag{B.198}$$

We observe that CPS achieves better solution quality than CPN, while CPN solves more instances to optimality.

Appendix C

Additional Results for Chapter 4

C.1 Detailed Comparison of the MIP, CP, and DIDP Solvers

Figures C.1–C.11 show the coverage over time and the distributions of the optimality gap over instances. On the left-hand side of each figure, the x-axis is time in seconds, and the y-axis is the ratio of coverage achieved within x seconds over the total number of instances. On the right-hand side, the x-axis is the optimality gap, and the y-axis is the ratio of instances where the optimality gap is less than or equal to x. Similarly, Figures C.12–C.22 show the distributions of the primal integral over instances. Higher and left is better in all figures.

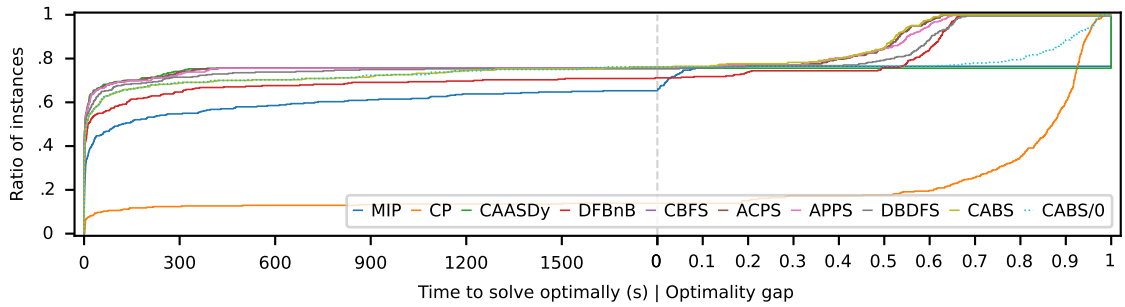


Figure C.1: The ratio of the coverage against time and the ratio of instances against the optimality gap in the traveling salesperson problem with time windows (TSPTW).

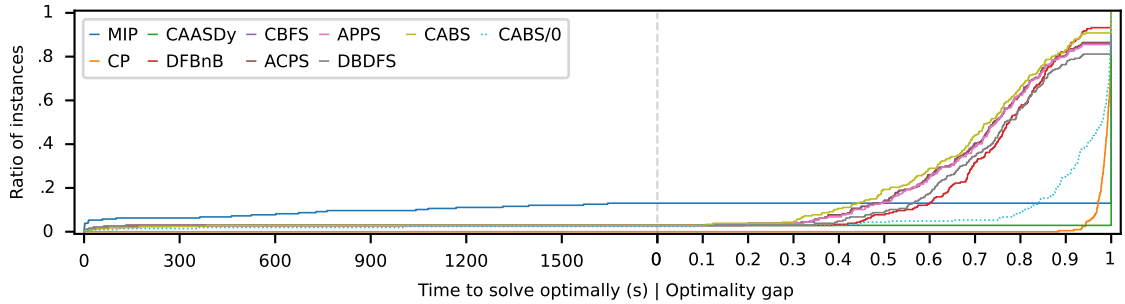


Figure C.2: The ratio of the coverage against time and the ratio of instances against the optimality gap in the capacitated vehicle routing problem (CVRP).

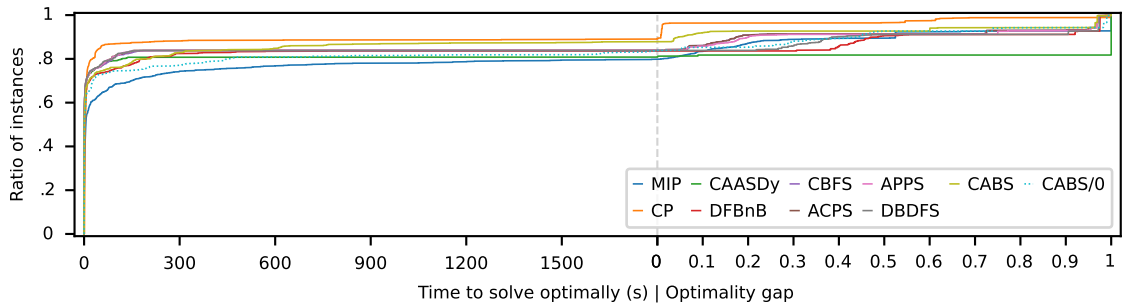


Figure C.3: The ratio of the coverage against time and the ratio of instances against the optimality gap in the multi-commodity pickup and delivery traveling salesperson problem (m-PDTSP).

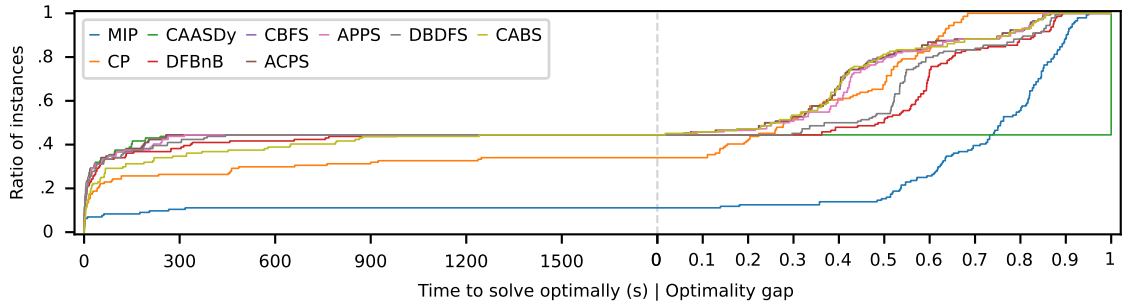


Figure C.4: The ratio of the coverage against time and the ratio of instances against the optimality gap in the orienteering problem with time windows (OPTW).

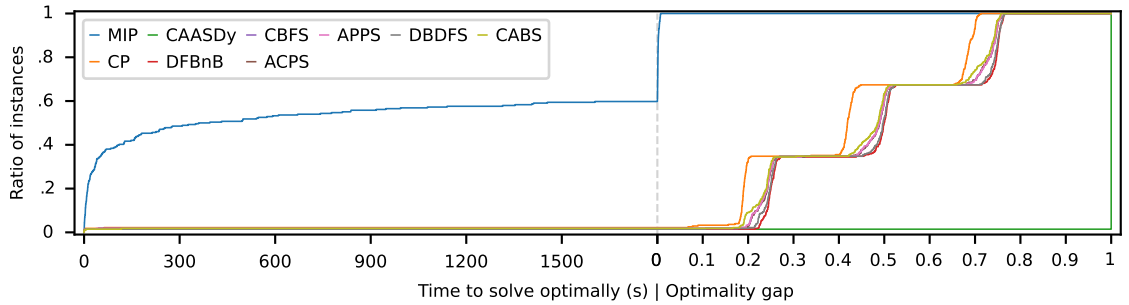


Figure C.5: The ratio of the coverage against time and the ratio of instances against the optimality gap in the multi-dimensional knapsack problem (MDKP).

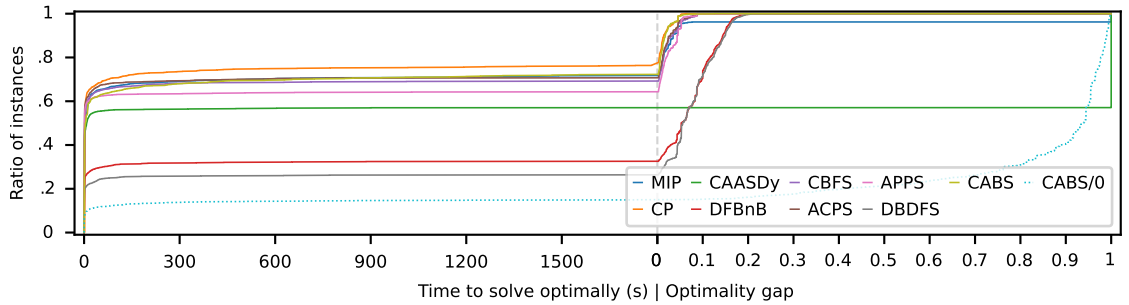


Figure C.6: The ratio of the coverage against time and the ratio of instances against the optimality gap in bin packing.

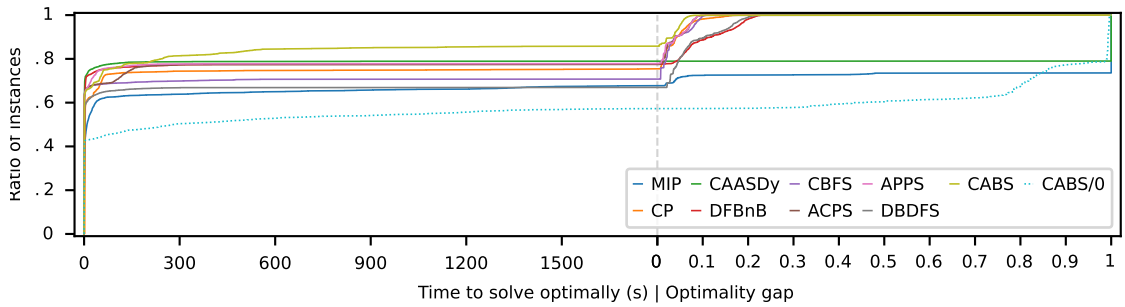


Figure C.7: The ratio of the coverage against time and the ratio of instances against the optimality gap in the simple assembly line balancing problem (SALBP-1).

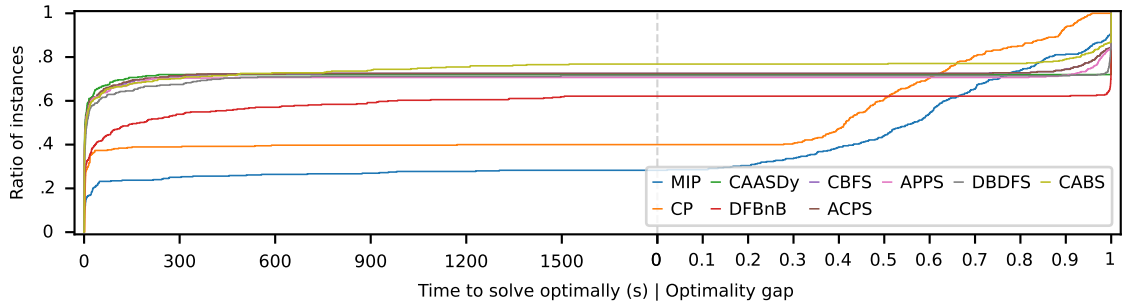


Figure C.8: The ratio of the coverage against time and the ratio of instances against the optimality gap in single machine total weighted tardiness ($1||\sum w_i T_i$).

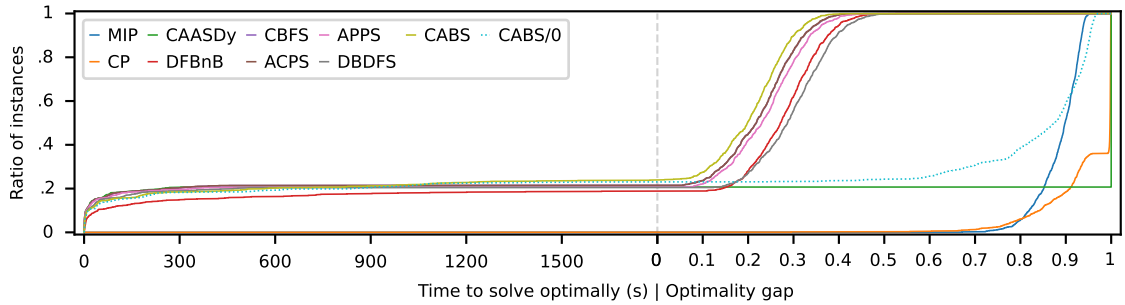


Figure C.9: The ratio of the coverage against time and the ratio of instances against the optimality gap in talent scheduling.

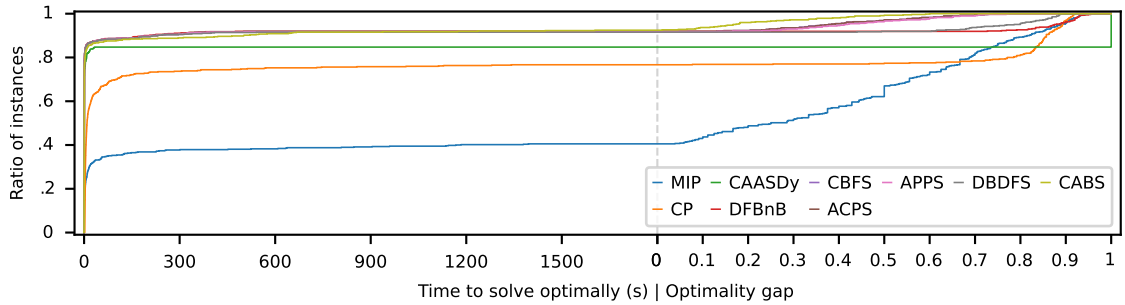


Figure C.10: The ratio of the coverage against time and the ratio of instances against the optimality gap in the minimization of open stacks problem (MOSP).

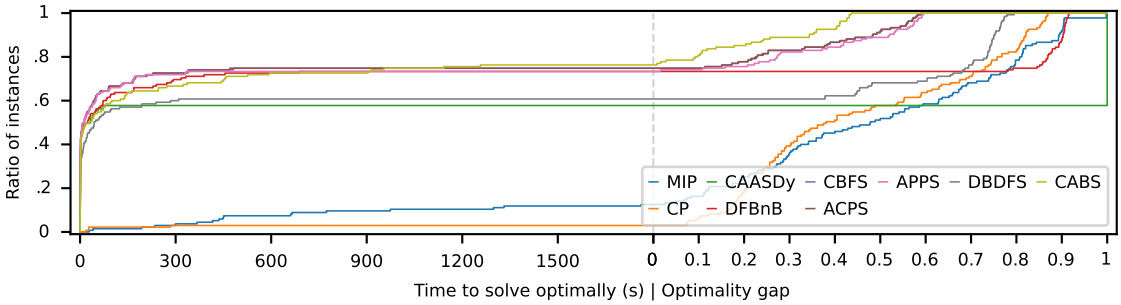


Figure C.11: The ratio of the coverage against time and the ratio of instances against the optimality gap in graph-clear.

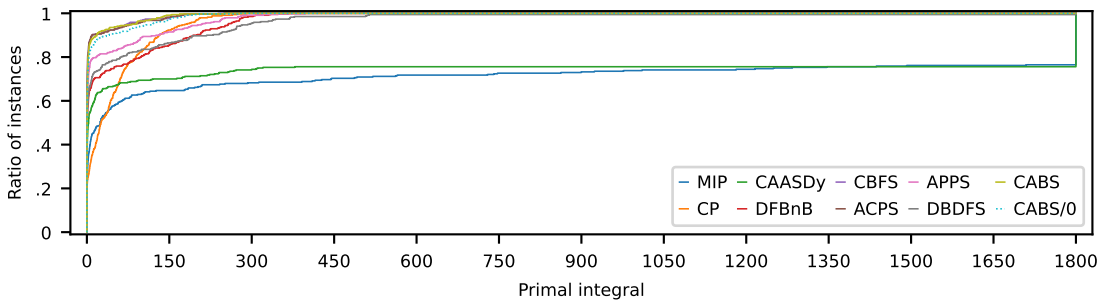


Figure C.12: The ratio of instances against the primal integral in the traveling salesperson problem with time windows (TSPTW).

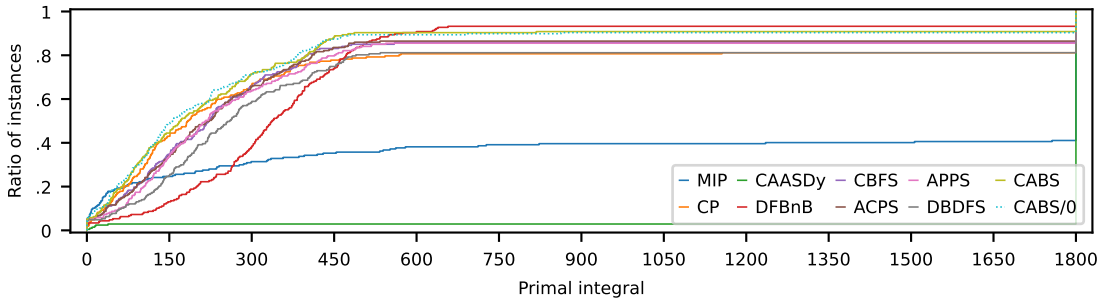


Figure C.13: The ratio of instances against the primal integral in the capacitated vehicle routing problem (CVRP).

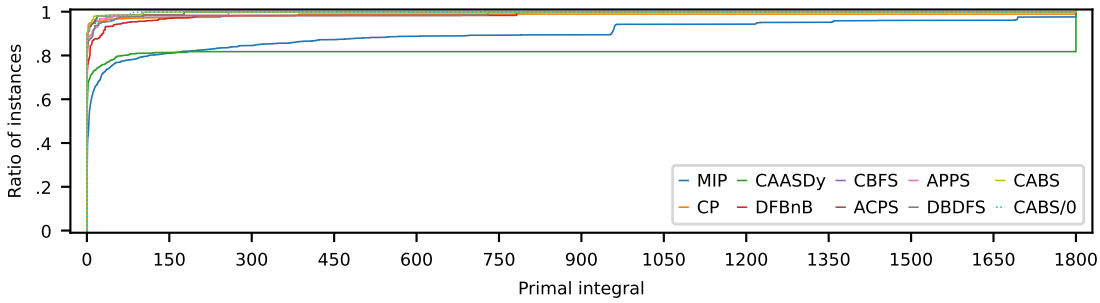


Figure C.14: The ratio of instances against the primal integral in the multi-commodity pickup and delivery traveling salesperson problem (m-PDTSP).

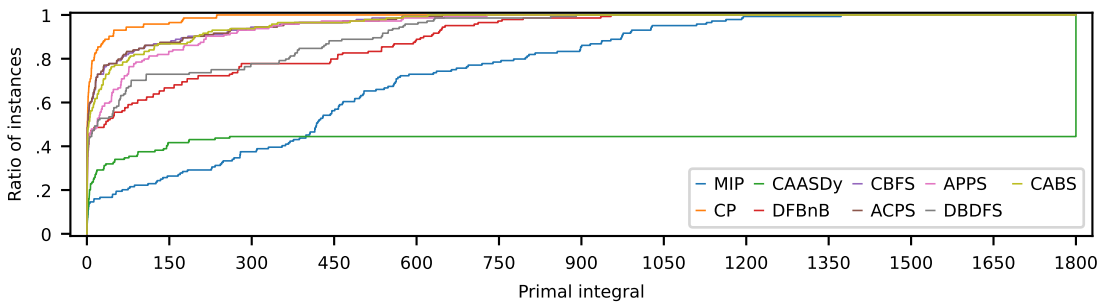


Figure C.15: The ratio of instances against the primal integral in the orienteering problem with time windows (OPTW).

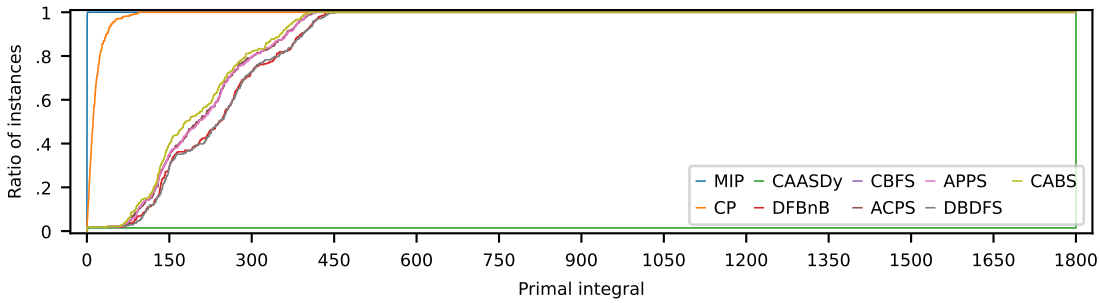


Figure C.16: The ratio of instances against the primal integral in the multi-dimensional knapsack problem (MDKP).

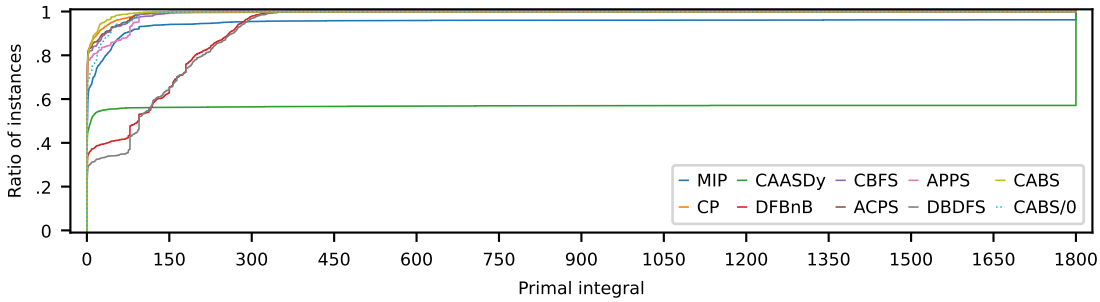


Figure C.17: The ratio of instances against the primal integral in bin packing.

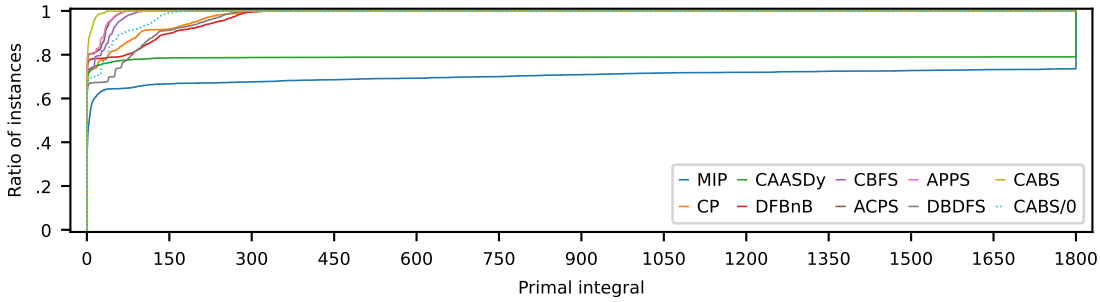


Figure C.18: The ratio of instances against the primal integral in the simple assembly line balancing problem (SALBP-1).

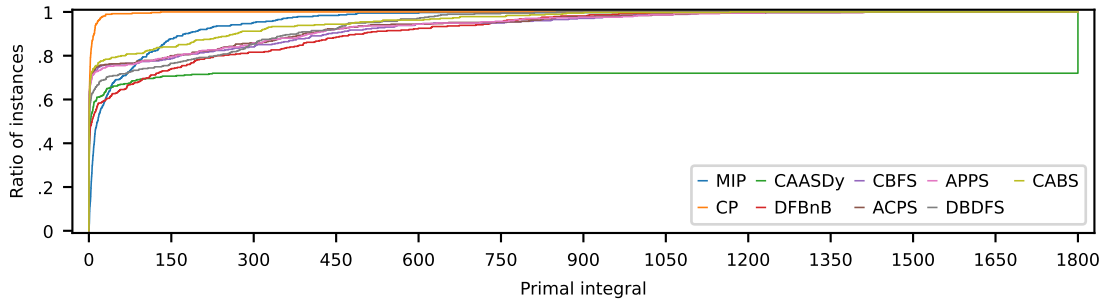


Figure C.19: The ratio of instances against the primal integral in single machine total weighted tardiness ($1||\sum w_i T_i$).

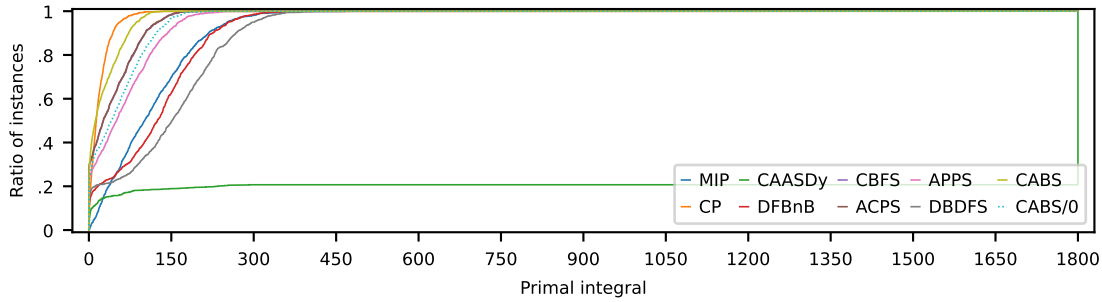


Figure C.20: The ratio of instances against the primal integral in talent scheduling.

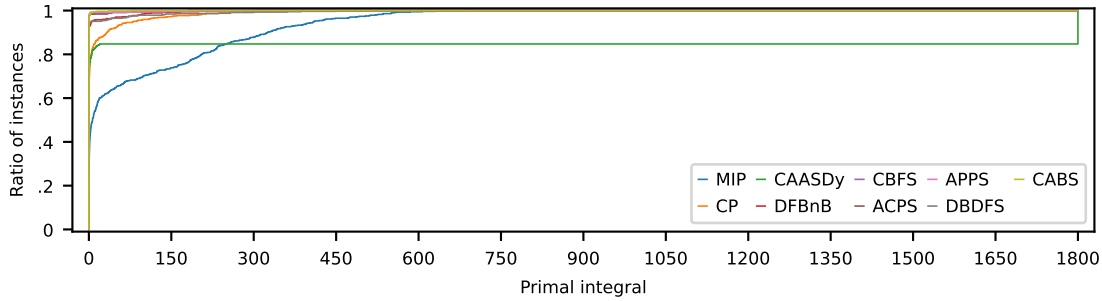


Figure C.21: The ratio of instances against the primal integral in the minimization of open stack problem (MOSP).

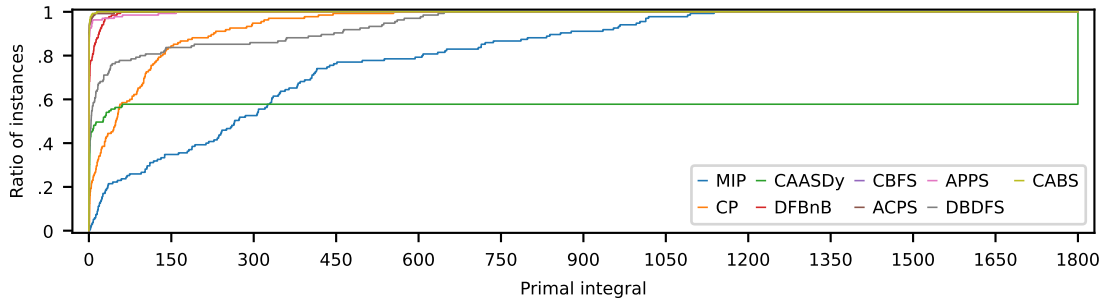


Figure C.22: The ratio of instances against the primal integral in graph-clear.

C.2 Comparison of DIDP and Other State-Based Approaches

We compare DIDP with other state-based approaches: domain-independent artificial intelligence (AI) planning [165] (Section 2.3.3), Picat [449], a logic programming language hybridized with AI planning (Section 2.4.2), and ddo [171], a decision diagram solver (Section 2.4.4). We explain how we formulate state-based models for these approaches before presenting the experimental results.

C.2.1 Domain-Independent AI Planning

We model the traveling salesperson problem with time windows (TSPTW) (Section 3.2.1), the capacitated vehicle routing problem (CVRP) (Section 3.3.1), the multi-commodity pickup and delivery traveling salesperson problem (m-PDTSP) (Section 3.3.2), bin packing (Section 3.3.5), and the simple assembly line balancing problem to minimize the number of stations (SALBP-1) (Section 3.3.6) as linear numeric planning tasks [212] (Section 3.1.1), where preconditions and effects of actions are represented by linear formulas of numeric state variables. In these models, the objective is to minimize the sum of nonnegative action costs, which is a standard in optimal numeric planning [378, 340, 341, 380, 286, 276, 274, 400, 273]. We use Planning Domain Definition Language (PDDL) 2.1 [141] to formulate the models and NLM-CutPlan Orbit [275], the winner of the optimal numeric track of the International Planning Competition (IPC) 2023,[1] to solve the models. NLM-CutPlan Orbit uses A* [190] with an admissible heuristic function [274, 273] and symmetry breaking [400]. The PDDL models are adaptations of the DyPDL models presented in Section 3.3, where a numeric or element variable in DyPDL becomes a numeric variable in PDDL (except for the element variable representing the current location in the routing problems as explained in the next paragraph), a set variable is represented by a predicate and a set of objects, the target state becomes the initial state, base cases become the goal conditions, and a transition becomes an action. However, we are unable to model dominance between states and dual bound functions in PDDL. In addition, we cannot differentiate forced transitions and other transitions. We explain other differences between the PDDL models and the DyPDL models in the following paragraphs.

In the DyPDL models of the routing problems (TSPTW, CVRP, and m-PDTSP), each transition visits one customer and increases the cost by the travel time from the current location to the customer. Since a goal state in PDDL is not associated with a cost unlike a base case in DyPDL, we also define a transition to return to the depot, which increases the cost by the travel time to the depot. While the travel time depends on the current location, for NLM-CutPlan Orbit, the cost of an action must be a nonnegative constant independent of a state, which is a standard in admissible heuristic functions for numeric planning [378, 341, 380, 276, 274, 273]. Thus, in the PDDL models, similar to the model presented in Figure 2.2 of Section 2.3.3, we define one action with two parameters, the current location (?from) and the destination (?to), so that the cost of each grounded action becomes a state-independent constant (c ?from ?to) corresponding to the travel time. We define a predicate (visited ?customer) representing if a customer is visited and (location ?customer) representing the current location. In each action, we use (location ?from) and (not (visited ?to)) as preconditions and (not (location ?from)), (location ?to), and (visited ?to) as effects.

In the DyPDL models of TSPTW, each transition updates the current time t to $\max\{t + c_{ij}, a_j\}$, where c_{ij} is the travel time and a_j is the beginning of the time window at the destination. While

[1] https://ipc2023-numeric.github.io/

this effect can be written as (assign (t) (max (+ (t) (c ?from ?to)) (a ?to))) in PDDL, to represent effects as linear formulas, we introduce two actions: one with a precondition (>= (+ (t) (c ?from ?to)) (a ?to)) and an effect (increase (t) (c ?from ?to)), corresponding to $t \leftarrow t + c_{ij}$ if $t + c_{ij} \geq a_j$, and another with a precondition (< (+ (t) (c ?from ?to)) (a ?to)) and an effect (assign (t) (a ?to)), corresponding to $t \leftarrow a_j$ if $t + c_{ij} < a_j$.

In the DyPDL models of TSPTW and CVRP, we have redundant state constraints. While a state constraint could be modeled by introducing it as a precondition of each action, we do not use the state constraints in the PDDL models of TSPTW and CVRP because efficiently modeling them is non-trivial: straightforward approaches result in an exponential number of actions. For TSPTW, the state constraint checks if all unvisited customers can be visited by the deadline, represented as $\forall j \in U, t + c^*_{ij} \leq b_j$, where U is the set of the unvisited customers, c^*_{ij} is the shortest travel time from the current location to customer j, and b_j is the deadline. One possible way to model this constraint is to define a disjunctive precondition (or (visited ?j) (<= (+ (t) (cstar ?from ?j)) (b ?j))) for each customer ?j, where (t) is a numeric variable corresponding to t, (cstar ?from ?j) is a numeric constant corresponding to c^*_{ij}, and (b ?j) is a numeric constant corresponding to b_j. However, the heuristic function used by NLM-CutPlan Orbit does not support disjunctive preconditions, and NLM-CutPlan Orbit compiles an action with disjunctive preconditions into a set of actions with different combinations of the preconditions.[2] In our case, each action has one of the two preconditions, (visited ?j) or (<= (+ (t) (cstar ?from ?j)) (b ?j)), resulting in 2^n actions in total, where n is the number of customers. In CVRP, the state constraint takes the sum of demands over all unvisited customers. To model this computation independently of a state, we need to define an action for each possible set of unvisited customers, resulting in 2^n actions in total.

In the DyPDL models of bin packing and SALBP-1, each transition packs an item in the current bin (schedules a task in the current station for SALBP-1) or opens a new bin. When opening a new bin, the transition checks if no item can be packed in the current bin as a precondition, which is unnecessary but useful to exclude suboptimal solutions. However, for similar reasons to the state constraint in TSPTW, we do not model this precondition in the PDDL models. We could model this condition by defining (or (packed ?j) (> (w ?j) (r))) for each item ?j, where (packed ?j) represents if ?j is already packed, (w ?j) represents the weight of ?j, and (r) is the remaining capacity. However, as discussed above, NLM-CutPlan Orbit would generate an exponential number of actions with this condition.

In addition to the above problem classes, we also use the minimization of open stacks problem (MOSP) (Section 3.3.9): it was used as a benchmark domain in the classical planning tracks of the International Planning Competitions from 2006 to 2014. This PDDL formulation is different from our DyPDL model. To solve the model, we use Ragnarok [108], the winner of the optimal classical track of IPC 2023,[3] which is a portfolio of multiple optimal classical planners.

We do not use other problem classes since their DyPDL models do not minimize the sum of the state-independent and nonnegative action costs. In single machine total weighted tardiness $(1||\sum w_i T_i)$ (Section 3.3.7) and talent scheduling (Section 3.3.8), since the cost of each transition depends on a set variable, we need an exponential number of actions to make it state-independent. In the orienteering problem with time windows (OPTW) (Section 3.3.3) and the multi-dimensional

[2]This approach is inherited from Fast Downward [203], the standard classical planning framework on which NLM-CutPlan Orbit is based.

[3]`https://ipc2023-classical.github.io/`

knapsack problem (MDKP) (Section 3.3.4), the objective is to maximize the sum of the nonnegative profits. In graph-clear (Section 3.3.10), the objective is to minimize the maximum value of state-dependent weights associated with transitions.

C.2.2 Picat

Picat is a logic programming language, in which dynamic programming (DP) can be used with tabling without implementing a DP algorithm. Picat provides an AI planning module based on tabling, where a state, goal conditions, and actions can be programmatically described by expressions in Picat. While the cost of a plan is still restricted to the sum of the nonnegative action costs, each action cost can be state-dependent. In addition, an admissible heuristic function can be defined and used by a solving algorithm. Thus, we can define a dual bound function as an admissible heuristic function in the AI planning module. However, we cannot model dominance between states. Using the AI planning module, we formulate models for TSPTW, CVRP, m-PDTSP, bin packing, SALBP-1, $1||\sum w_i T_i$, and talent scheduling, which are the same as the DyPDL models except that they do not define dominance. To solve the formulated models, we use the `best_plan_bb` predicate, which performs a branch-and-bound algorithm using the heuristic function. For OPTW, MDKP, MOSP, and graph-clear, we do not use the AI planning module due to the objective structure of their DyPDL models. We define DP models for these problem classes, which are the same as the DyPDL models except that they do not define dominance and dual bound functions, using tabling without the AI planning module.

C.2.3 Ddo

Since ddo requires a user to define a merging operator and a ranking operator in addition to DP models, our DyPDL models cannot be used with ddo. We use TSPTW and talent scheduling for our evaluation, for which previous work developed models for ddo [169, 87, 88, 89]. For TSPTW, while we minimize the total travel time, which does not include the waiting time, the model for ddo minimizes the makespan, which is the time spent until returning to the depot. Therefore, we adapt our DyPDL model to minimize the makespan: when visiting customer j from the current location i with time t, we increase the cost by $\max\{c_{ij}, a_j - t\}$ instead of c_{ij}. We present a YAML-DyPDL domain file for this formulation in Figure A.3 in Appendix A.2. To avoid confusion, in what follows, we call TSPTW to minimize the makespan TSPTW-M.

C.2.4 Experimental Results

We evaluate NLM-CutPlan Orbit[4] and Ragnarok[5] using GCC 12.3, Picat 3.6, and ddo 2.0.0 using Rust 1.70.0. For Ragnarok, we use IBM ILOG CPLEX 22.1.1 as a linear programming solver. We use the ddo models for TSPTW-M and talent scheduling obtained from the published repository of ddo.[6] For domain-independent AI planning, a problem instance is translated to PDDL files by a Python script before using a planner. For Picat, a problem instance of CVRP, m-PDTSP, OPTW, SALBP-1, $1||\sum w_i T_i$, and talent scheduling is preprocessed and formatted by a Python script so

[4] `https://github.com/ipc2023-numeric/team-1`
[5] `https://github.com/ipc2023-classical/planner17/tree/latest`
[6] `https://github.com/xgillard/ddo/tree/b2e68bfc085af7cc09ece38cc9c81acb0da6e965/ddo/examples`

Table C.1: Coverage of MIP, CP, domain-independent AI planners, Picat, and CABS. 'AI planning' represents the result of domain-independent AI planners, where Ragnarok is used for MOSP, and NLM-CutPlan Orbit is used for the other problem classes. The coverage of a solver is in bold if it is higher than MIP and CP, and the higher of MIP and CP is in bold if there is no better solver. The highest coverage is underlined.

	MIP	CP	AI Planning	Picat	CABS
TSPTW (340)	222	47	61	210	**259**
CVRP (207)	**27**	0	1	6	6
m-PDTSP (1178)	940	**1049**	1031	804	1035
OPTW (144)	16	49	-	26	**64**
MDKP (276)	**165**	6	-	3	5
Bin Packing (1615)	1159	**1234**	18	895	1167
SALBP-1 (2100)	1423	1584	871	**1590**	**1802**
$1\|\|\sum w_i T_i$ (2100)	106	150	-	**199**	**288**
Talent Scheduling (1000)	0	0	-	**84**	**239**
MOSP (570)	231	437	193	162	**527**
Graph-Clear (135)	17	4	-	**45**	**103**

Table C.2: Coverage and the average optimality gap of ddo and CABS in TSPTW-M and talent scheduling. For TSPTW-M, the optimality gap is not presented since ddo runs out of 8 GB memory in all unsolved instances of TSPTW-M and does not report intermediate solutions. For talent scheduling, the average optimality gap is computed from 976 instances where ddo does not reach the memory limit. The better value is in bold.

	Ddo		CABS	
	coverage	optimality gap	coverage	optimality gap
TSPTW-M (340)	213	-	**260**	-
Talent Scheduling (1000)	210	**0.1424**	**239**	0.1730

that Picat can easily parse it. We use the same experimental setting as Section 4.3.4: each solver uses a single thread, an 8 GB memory limit, and a 30-minute time limit.

Since the domain-independent AI planners and Picat return only an optimal solution, we evaluate only coverage, the number of optimally solved instances, for them. While ddo returns the best solution and dual bound found within the time limit, it does not return intermediate solutions and bounds during solving. Since we manage the memory limit using an external process, when ddo reaches the memory limit, it is killed without returning the best solution. In TSPTW-M, ddo reaches the memory limit in all unsolved instances. Therefore, we evaluate only coverage in TSPTW-M and present the average optimality gap computed from 976 out of 1000 talent scheduling instances where ddo does not reach the memory limit.

We present the coverage of the AI planners and Picat in Table C.1. We also include mixed-integer programming (MIP), constraint programming (CP), and complete anytime beam search (CABS) (Section 4.2.7), the best DIDP solver in Section 4.3. CABS has higher or equal coverage than the planners and Picat in all problem classes. As discussed in Section 4.3.8, this result is not surprising since the planners and the AI planning module and tabling in Picat are not designed for combinatorial optimization. It might be possible to improve the PDDL and Picat models so that they are more suited for these approaches. Moreover, for domain-independent AI planning, different planners might be better for combinatorial optimization. However, our point is to show that the performance achieved by the DIDP solvers is not a trivial consequence of the state-based modeling approach, and DIDP is doing something that existing approaches are not able to easily do.

We compare ddo and CABS in Table C.2. CABS is better than ddo in TSPTW-M and talent scheduling in coverage, but ddo has a better average optimality gap in talent scheduling.

Appendix D

Additional Results for Chapter 5

Figures D.1–D.14 show the distributions of the primal gap over instances for mixed-integer programming (MIP), constraint programming (CP), complete anytime search (CABS), large neighborhood search with decision diagrams (DD-LNS) [170], and large neighborhood beam search (LNBS) configurations. CABS/0 and LNBS/uniform/0 evaluated in bin packing and the simple assembly line balancing problem (SALBP-1) are CABS and LNBS using the zero dual bound function. CABS/blind and LNBS/uniform/blind used in OPTW do not use a dual bound function. The x-axis is the primal gap, and the y-axis is the ratio of instances where the primal gap is less than or equal to x. Higher and left is better. Similarly, Figures D.15–D.28 show the distributions of the primal integral over instances.

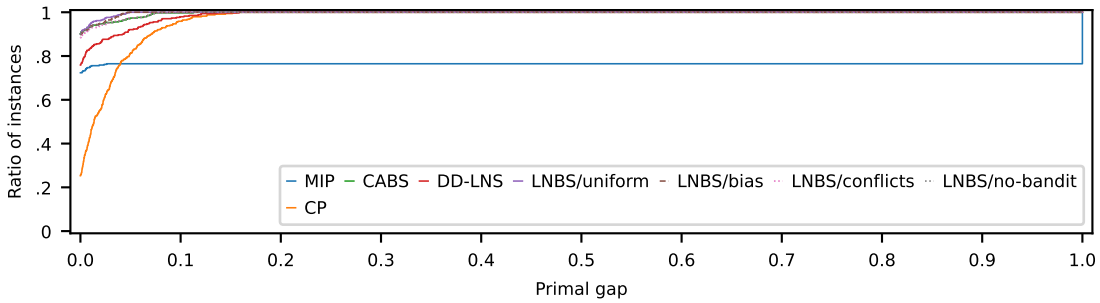


Figure D.1: The ratio of instances against the primal gap in the traveling salesperson problem with time windows (TSPTW).

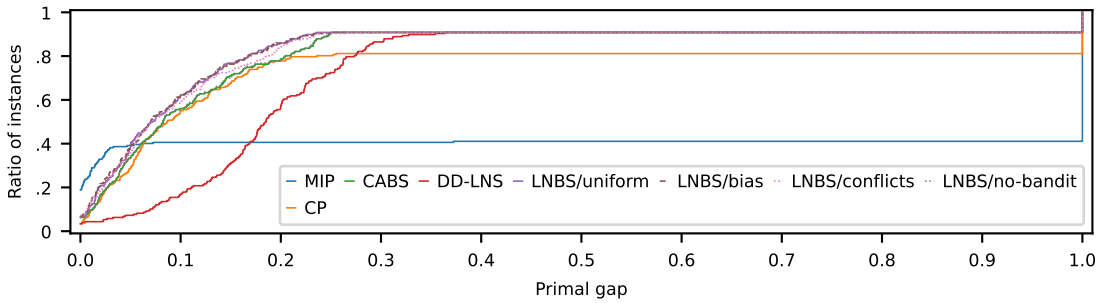


Figure D.2: The ratio of instances against the primal gap in the capacitated vehicle routing problem (CVRP).

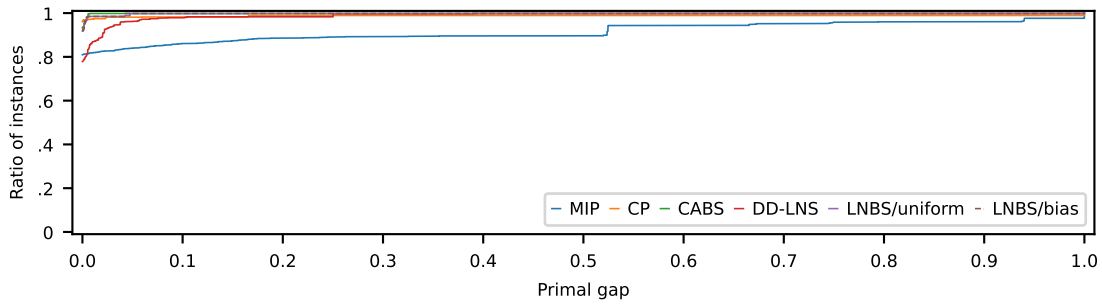


Figure D.3: The ratio of instances against the primal gap in the multi-commodity pickup and delivery traveling salesperson problem (m-PDTSP).

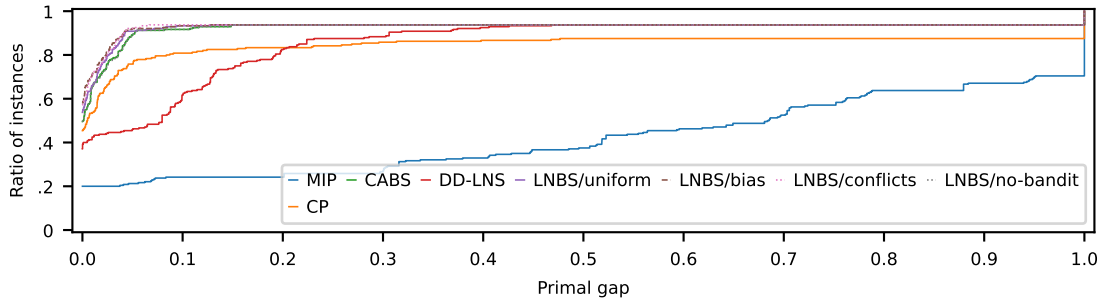


Figure D.4: The ratio of instances against the primal gap in the large instances of the multi-commodity pickup and delivery traveling salesperson problem (m-PDTSP).

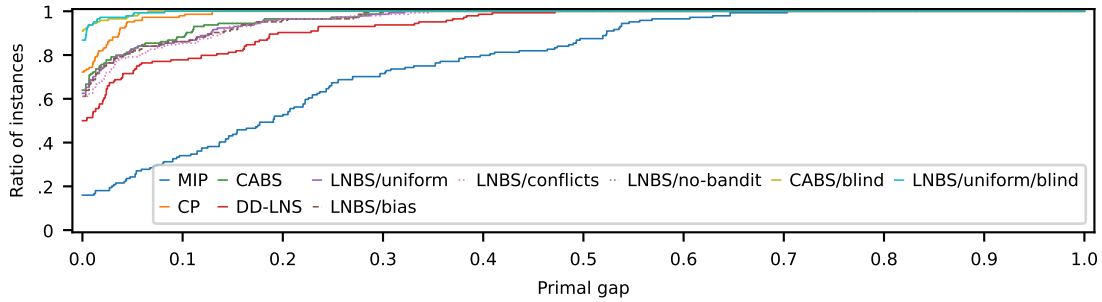


Figure D.5: The ratio of instances against the primal gap in the orienteering problem with time windows (OPTW).

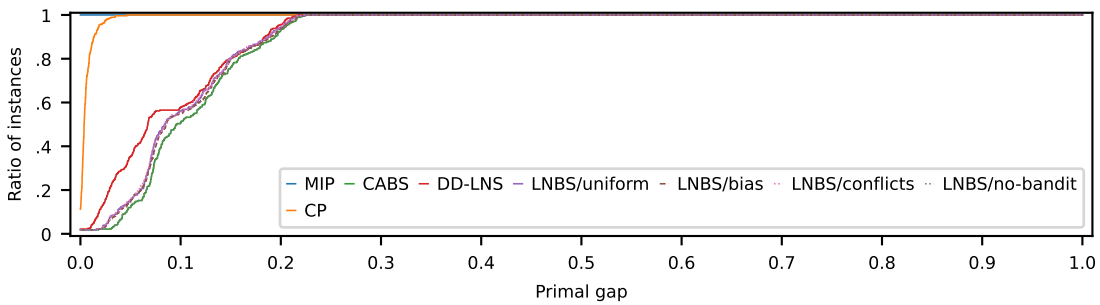


Figure D.6: The ratio of instances against the primal gap in the multi-dimensional knapsack problem (MDKP).

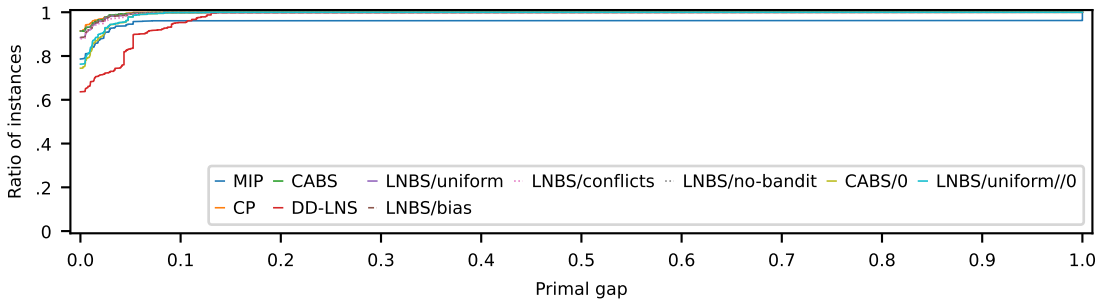


Figure D.7: The ratio of instances against the primal gap in bin packing.

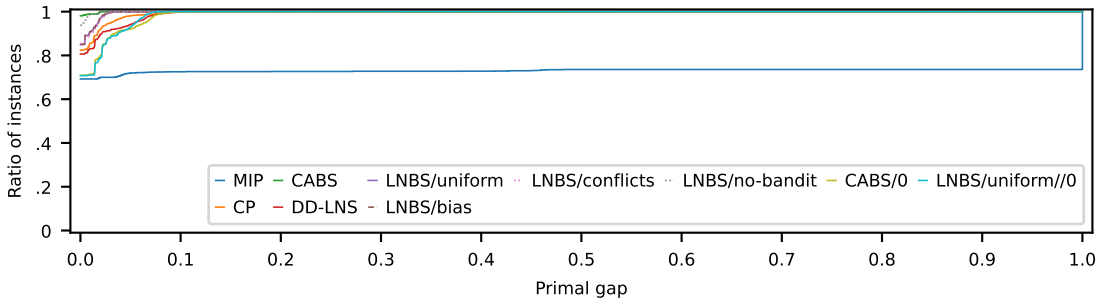


Figure D.8: The ratio of instances against the primal gap in the simple assembly line balancing problem (SALBP-1).

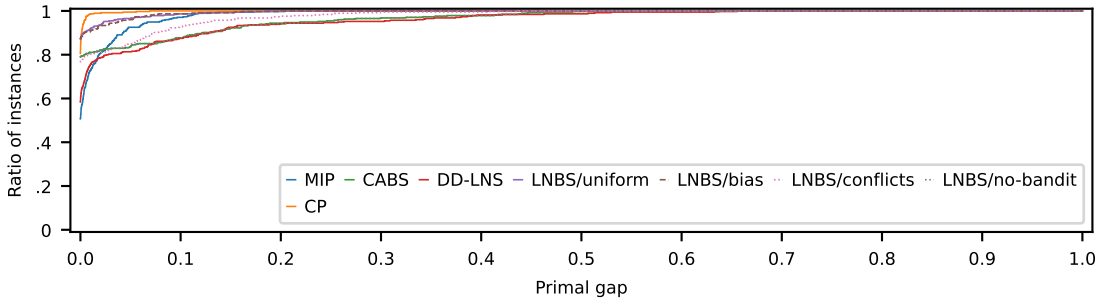


Figure D.9: The ratio of instances against the primal gap in single machine total weighted tardiness ($1||\sum w_i T_i$).

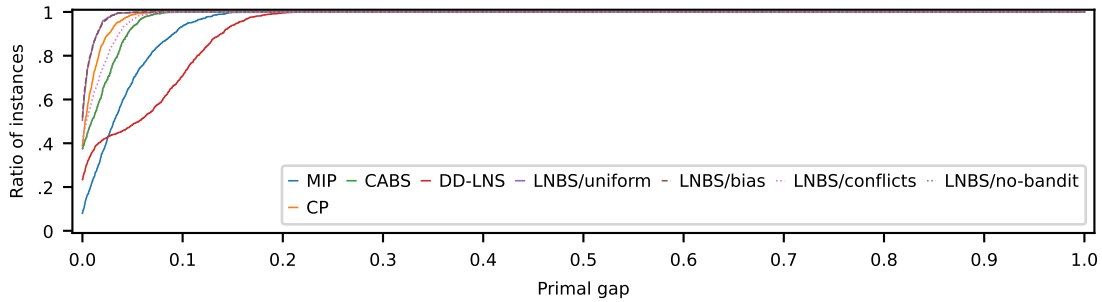


Figure D.10: The ratio of instances against the primal gap in talent scheduling.

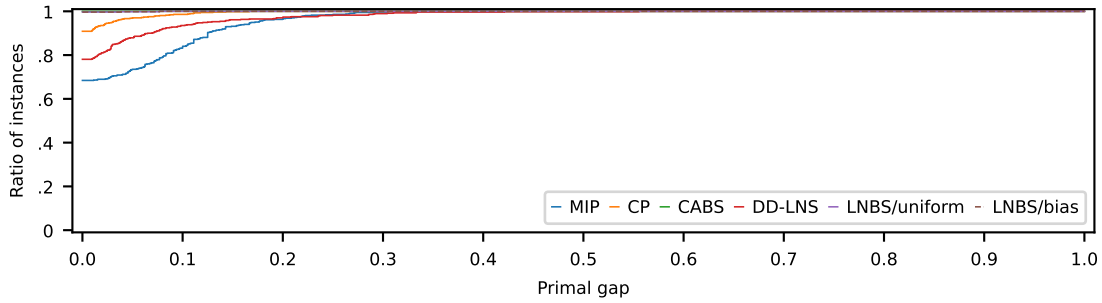


Figure D.11: The ratio of instances against the primal gap in the minimization of open stacks problem (MOSP).

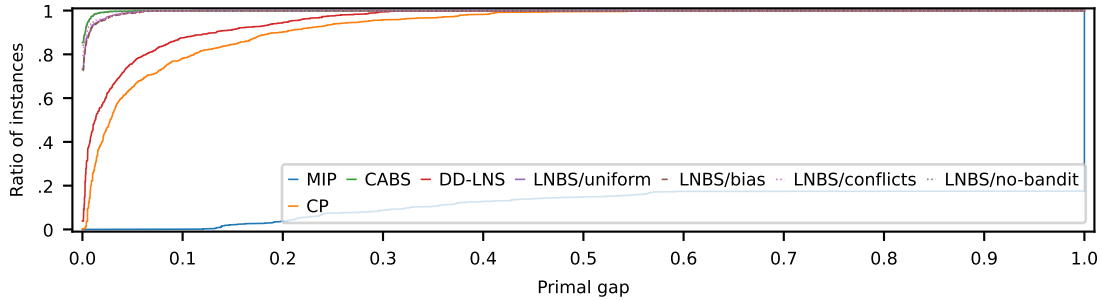


Figure D.12: The ratio of instances against the primal gap in the large instances of the minimization of open stacks problem (MOSP).

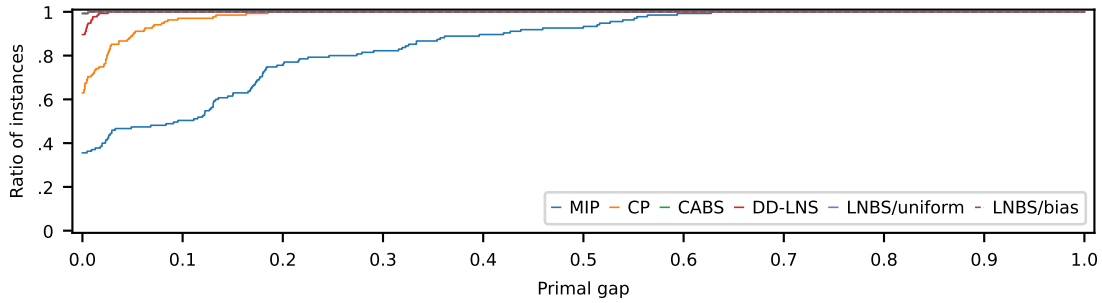


Figure D.13: The ratio of instances against the primal gap in graph-clear.

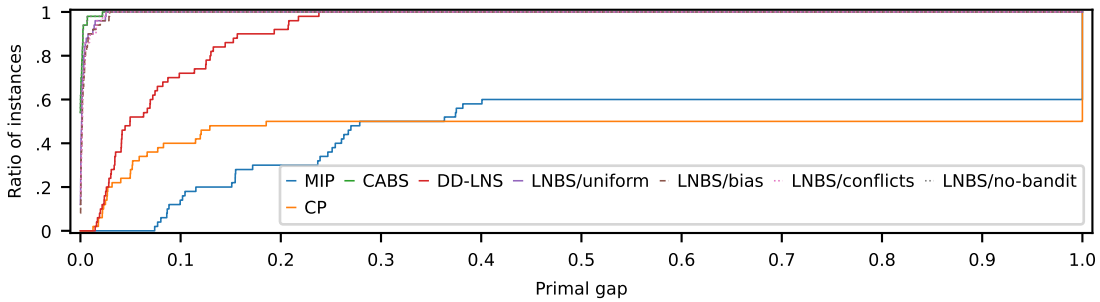


Figure D.14: The ratio of instances against the primal gap in the large instances of graph-clear.

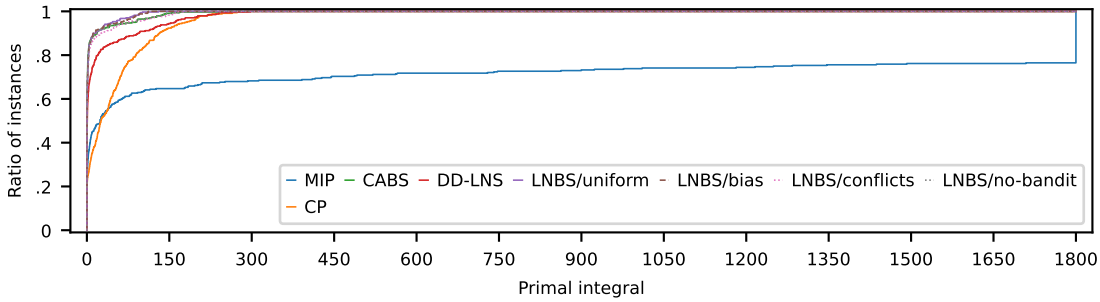


Figure D.15: The ratio of instances against the primal integral in the traveling salesperson problem with time windows (TSPTW).

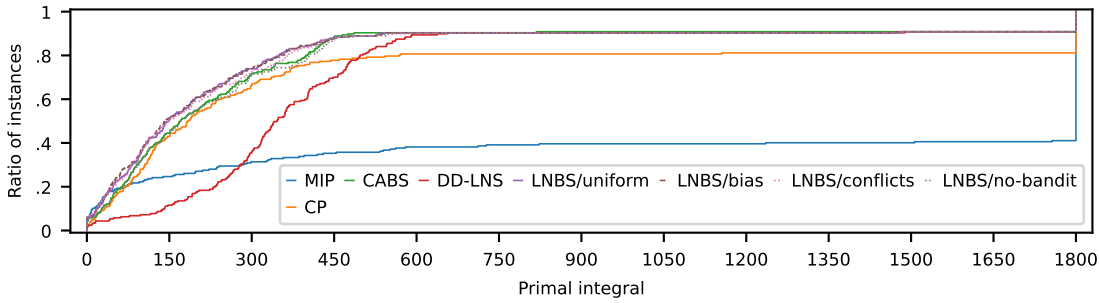


Figure D.16: The ratio of instances against the primal integral in the capacitated vehicle routing problem (CVRP).

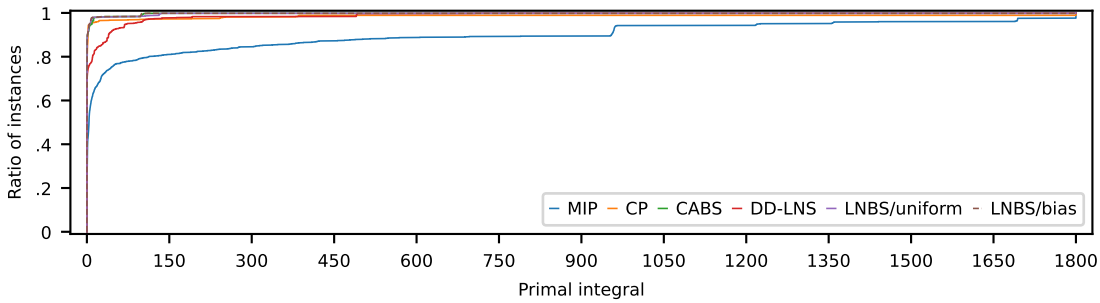


Figure D.17: The ratio of instances against the primal integral in the multi-commodity pickup and delivery traveling salesperson problem (m-PDTSP).

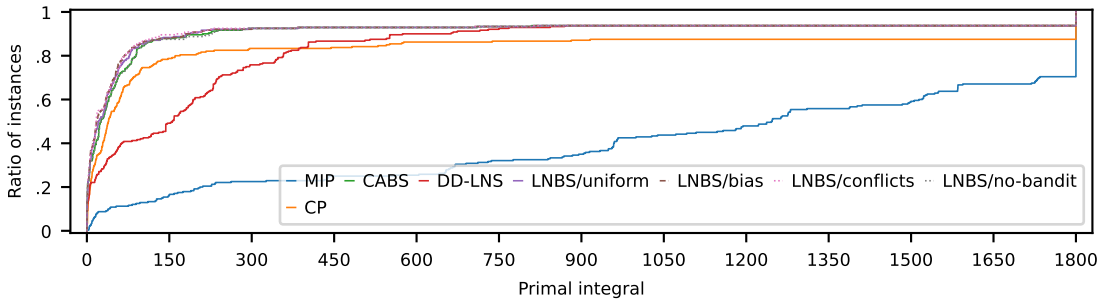


Figure D.18: The ratio of instances against the primal integral in the large instances of the multi-commodity pickup and delivery traveling salesperson problem (m-PDTSP).

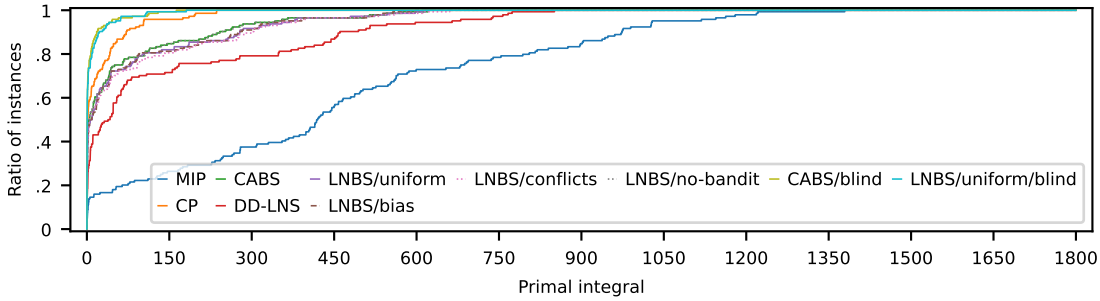


Figure D.19: The ratio of instances against the primal integral in the orienteering problem with time windows (OPTW).

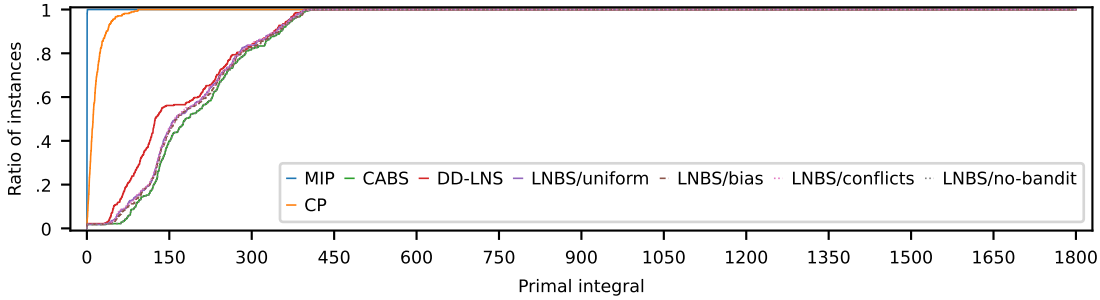


Figure D.20: The ratio of instances against the primal integral in the multi-dimensional knapsack problem (MDKP).

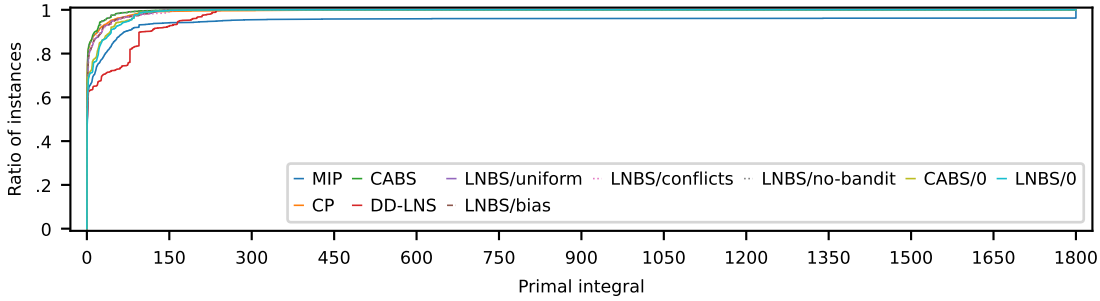


Figure D.21: The ratio of instances against the primal integral in bin packing.

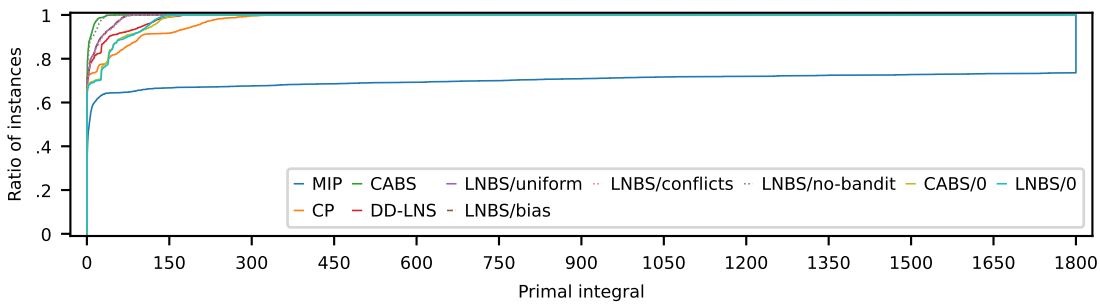


Figure D.22: The ratio of instances against the primal integral in the simple assembly line balancing problem (SALBP-1).

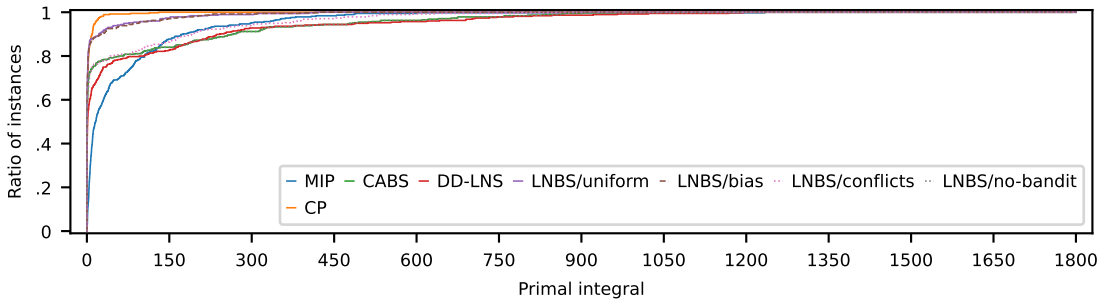


Figure D.23: The ratio of instances against the primal integral in single machine total weighted tardiness ($1||\sum w_i T_i$).

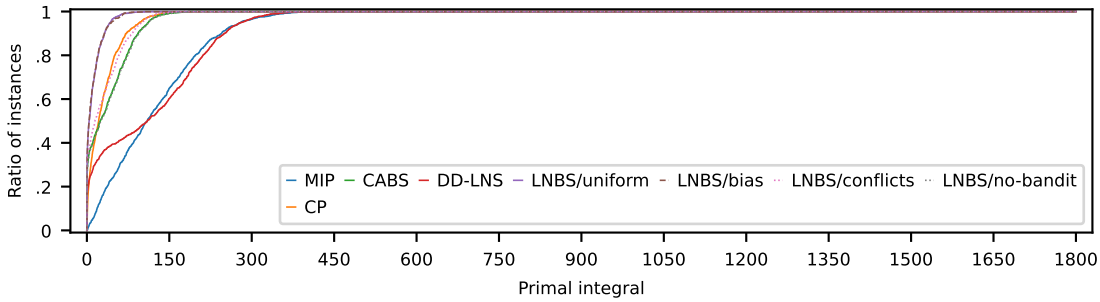


Figure D.24: The ratio of instances against the primal integral in talent scheduling.

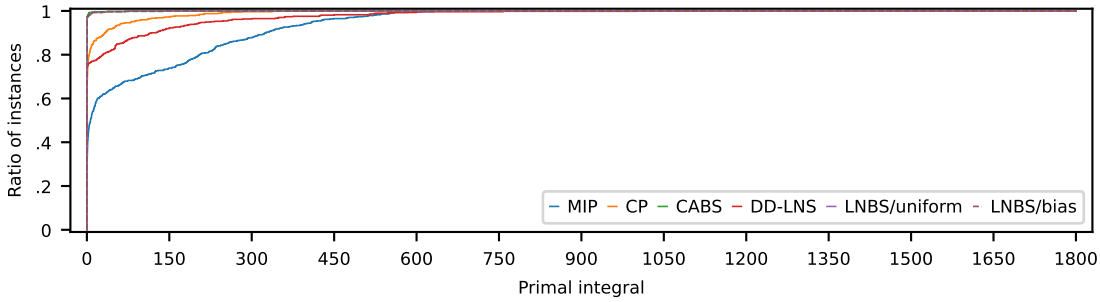


Figure D.25: The ratio of instances against the primal integral in the minimization of open stacks problem (MOSP).

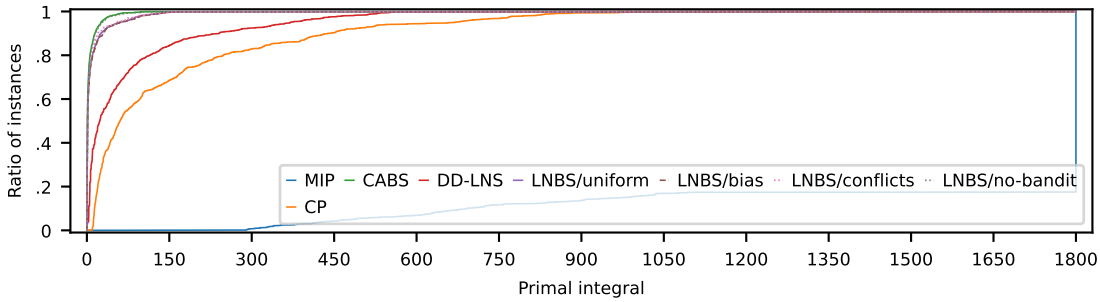


Figure D.26: The ratio of instances against the primal integral in the large instances of the minimization of open stacks problem (MOSP).

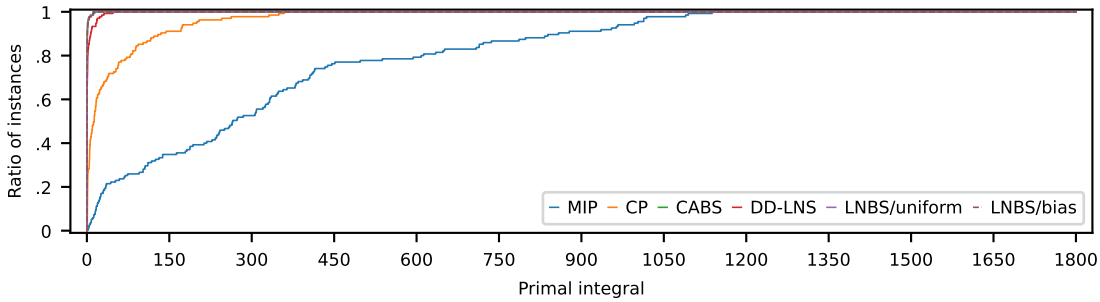


Figure D.27: The ratio of instances against the primal integral in graph-clear.

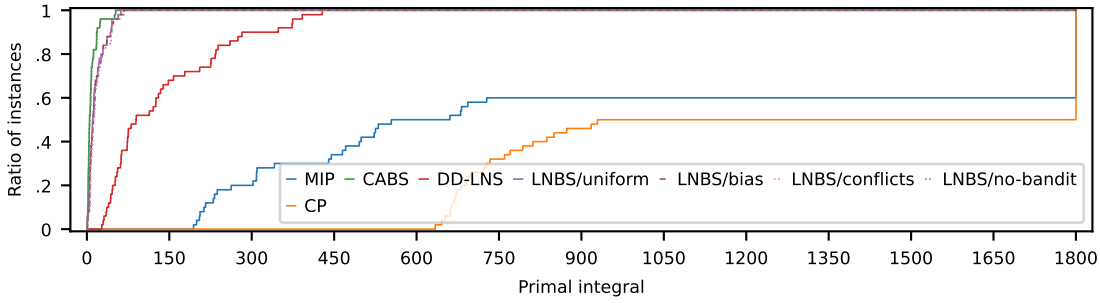


Figure D.28: The ratio of instances against the primal integral in the large instances of graph-clear.

Appendix E

Additional Results for Chapter 6

Figures E.1–E.6 show the coverage over time and the distributions of the optimality gap over instances for mixed-integer programming (MIP), constraint programming (CP), complete anytime search (CABS) with 1 thread and with 32 threads. For CABS, we show different parallelization methods, shared beam search (CASBS) and two variants of hash-distributed beam search (CAHDBS1 and CAHDBS2). Figures E.7–E.12 show the distribution of the primal integral. Figures E.13–E.20 show the results for CABS with 1 thread and CAHDBS2 with 8, 16, and 32 threads.

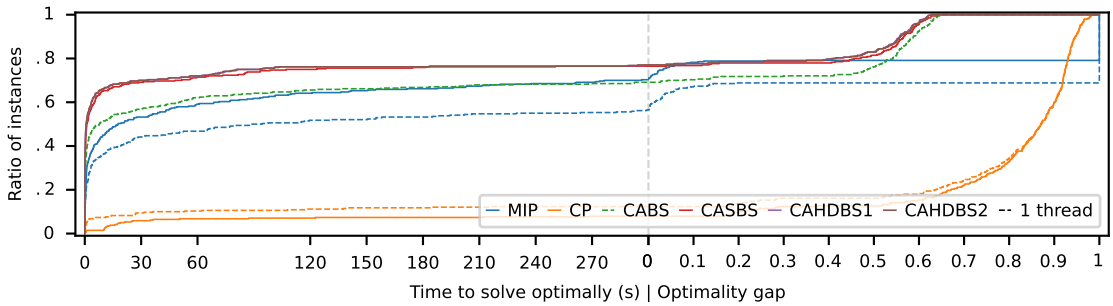


Figure E.1: The ratio of the coverage against time and the ratio of instances against the optimality gap in the traveling salesperson problem with time windows (TSPTW).

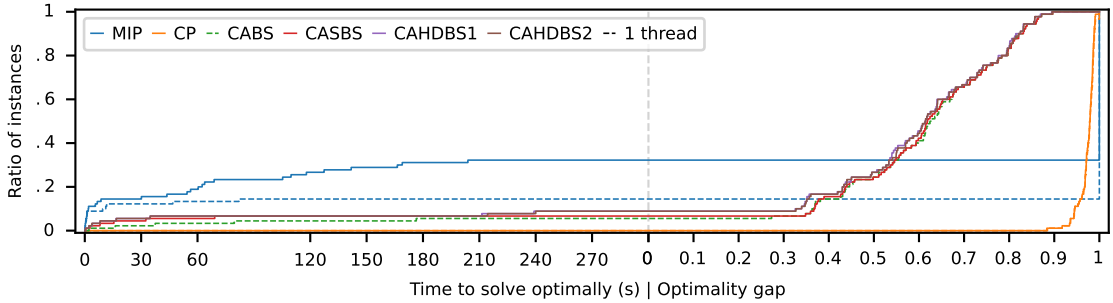


Figure E.2: The ratio of the coverage against time and the ratio of instances against the optimality gap in the capacitated vehicle routing problem (CVRP).

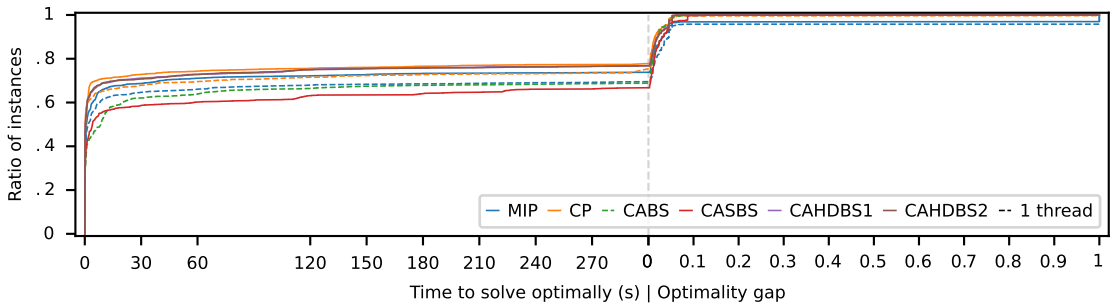


Figure E.3: The ratio of the coverage against time and the ratio of instances against the optimality gap in bin packing.

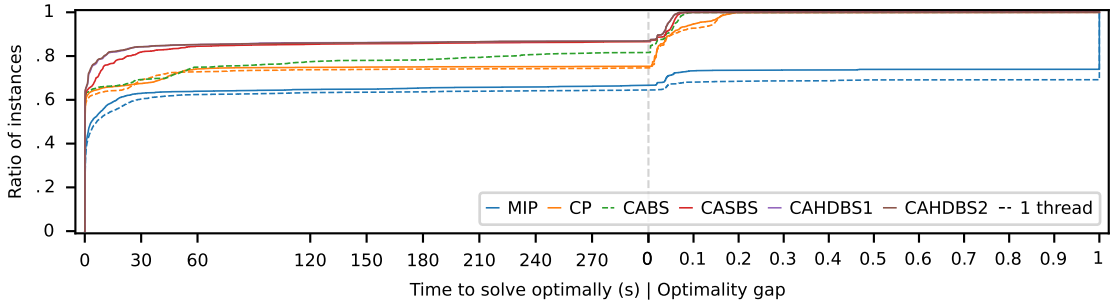


Figure E.4: The ratio of the coverage against time and the ratio of instances against the optimality gap in the simple assembly line balancing problem (SALBP-1).

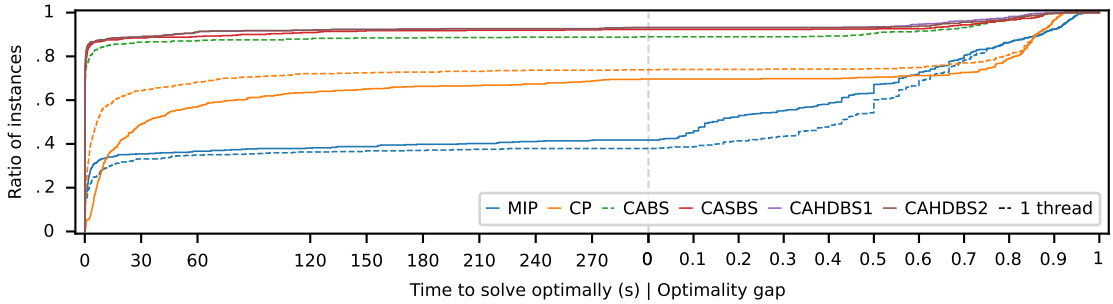


Figure E.5: The ratio of the coverage against time and the ratio of instances against the optimality gap in the minimization of open stacks problem (MOSP).

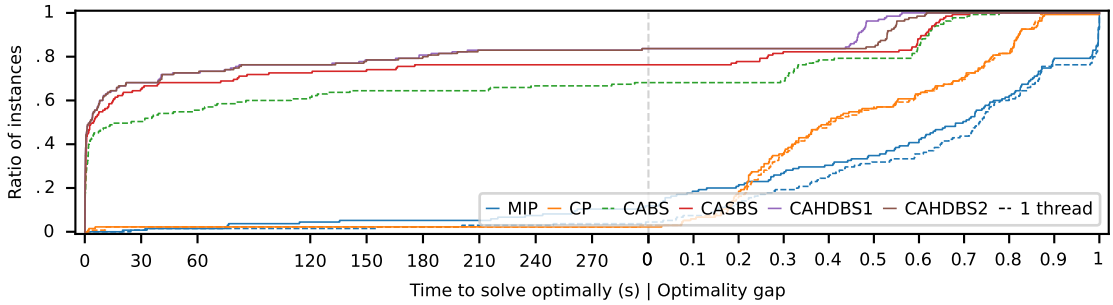


Figure E.6: The ratio of the coverage against time and the ratio of instances against the optimality gap in graph-clear.

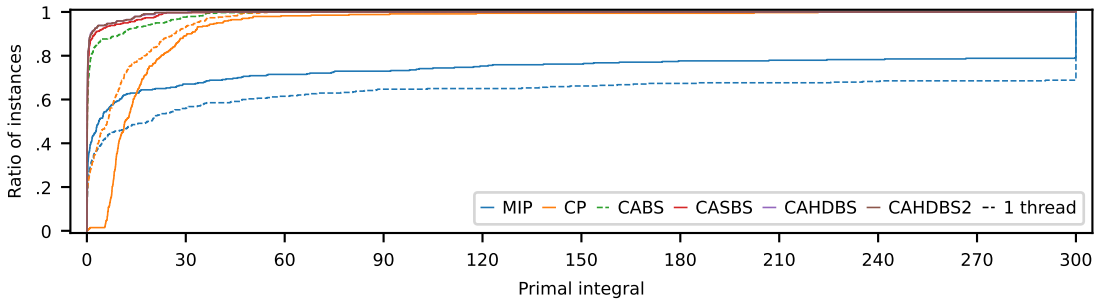


Figure E.7: The ratio of instances against the primal integral in the traveling salesperson problem with time windows (TSPTW).

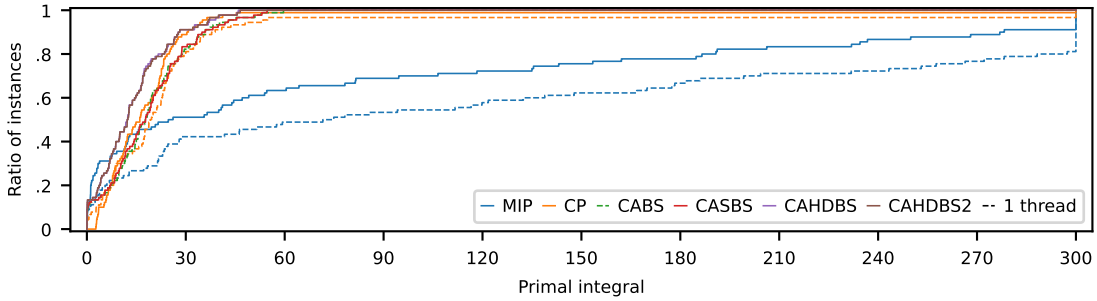


Figure E.8: The ratio of instances against the primal integral in the capacitated vehicle routing problem (CVRP).

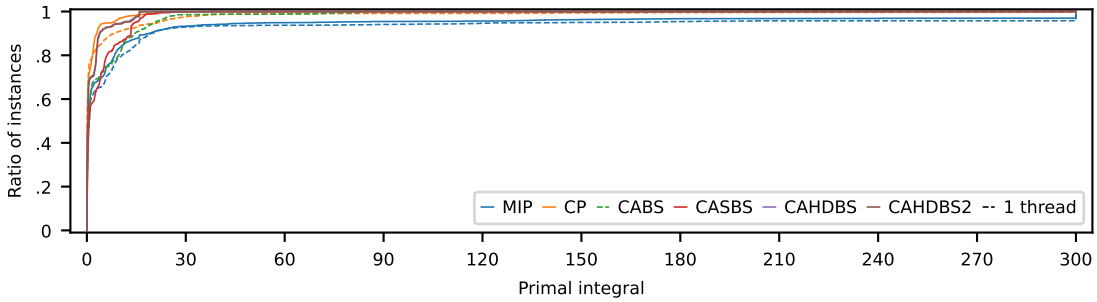


Figure E.9: The ratio of instances against the primal integral in bin packing.

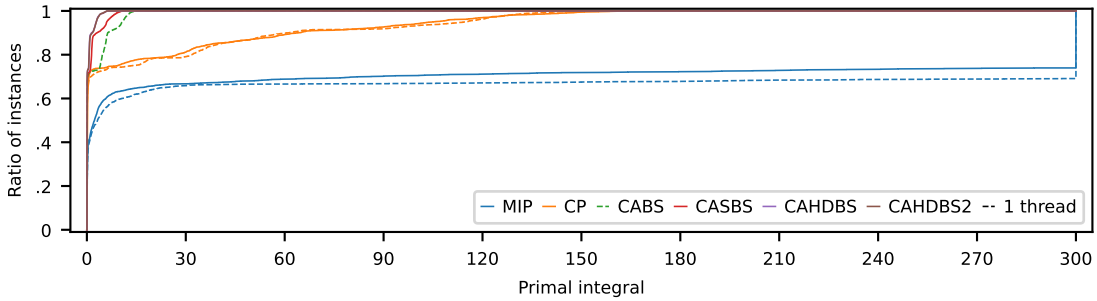


Figure E.10: The ratio of instances against the primal integral in the simple assembly line balancing problem (SALBP-1).

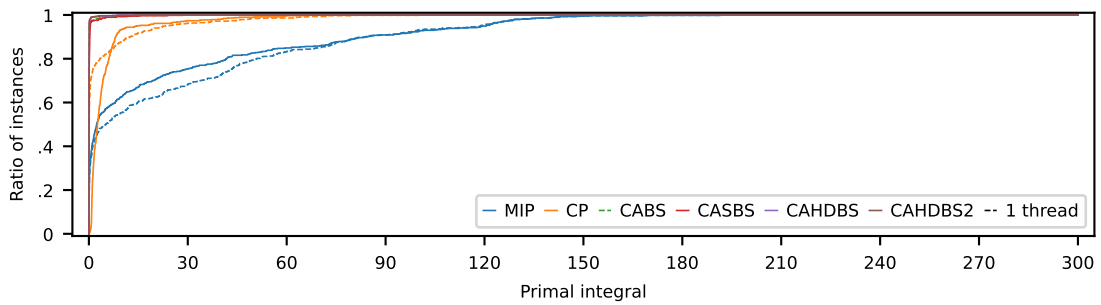


Figure E.11: The ratio of instances against the primal integral in the minimization of open stacks problem (MOSP).

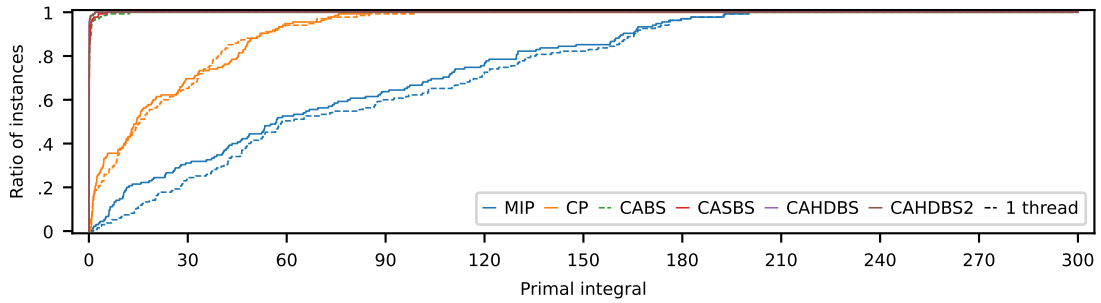


Figure E.12: The ratio of instances against the primal integral in graph-clear.

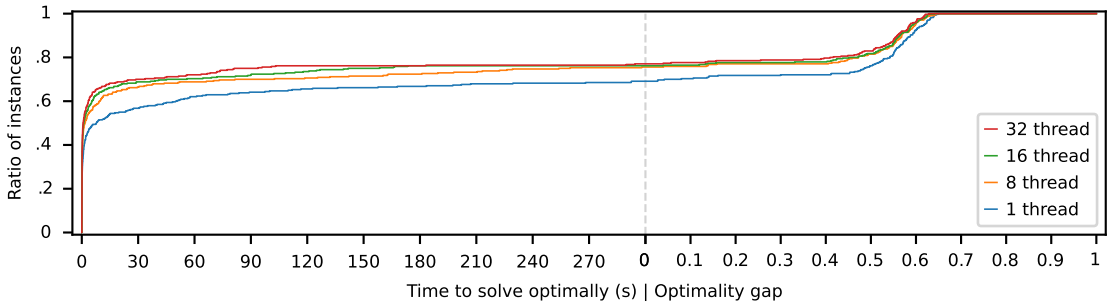


Figure E.13: The ratio of the coverage against time and the ratio of instances against the optimality gap in the traveling salesperson problem with time windows (TSPTW) for CAHDBS2.

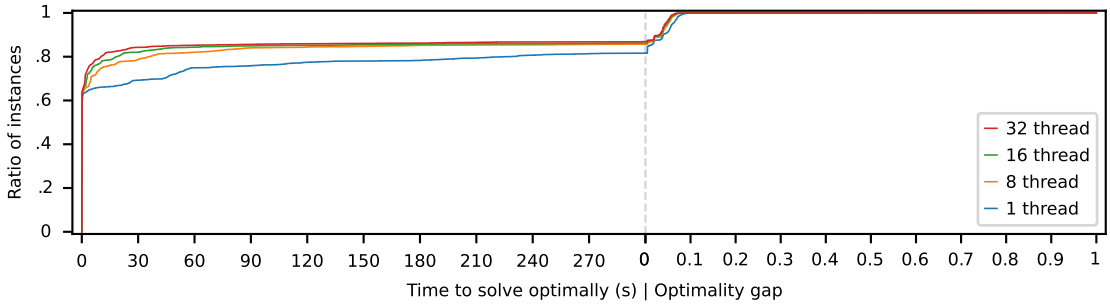


Figure E.14: The ratio of the coverage against time and the ratio of instances against the optimality gap in the simple assembly line balancing problem (SALBP-1) for CAHDBS2.

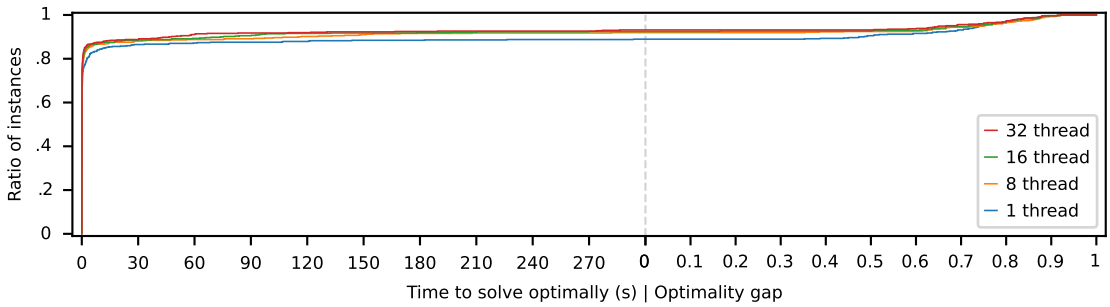


Figure E.15: The ratio of the coverage against time and the ratio of instances against the optimality gap in the minimization of open stacks problem (MOSP) for CAHDBS2.

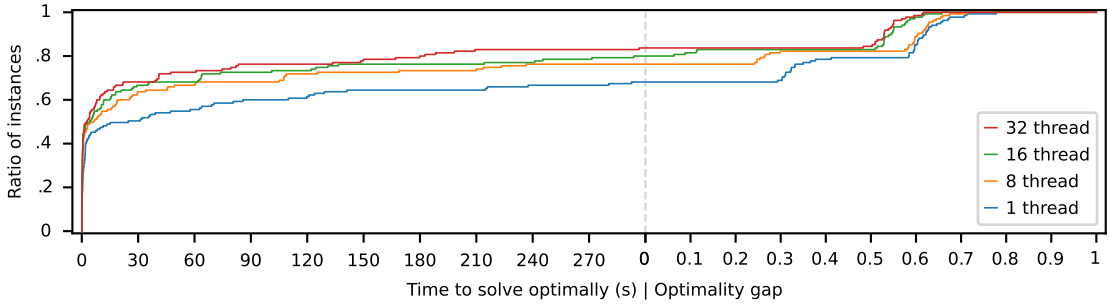


Figure E.16: The ratio of the coverage against time and the ratio of instances against the optimality gap in graph-clear for CAHDBS2.

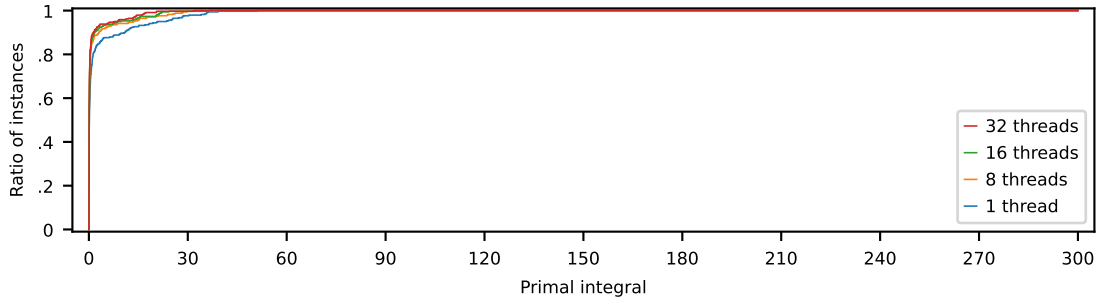


Figure E.17: The ratio of instances against the primal integral in the traveling salesperson problem with time windows (TSPTW) for CAHDBS2.

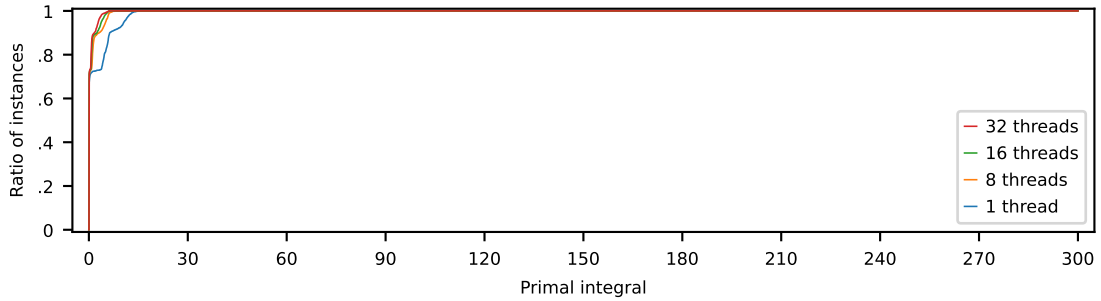


Figure E.18: The ratio of instances against the primal integral in the simple assembly line balancing problem (SALBP-1) for CAHDBS2.

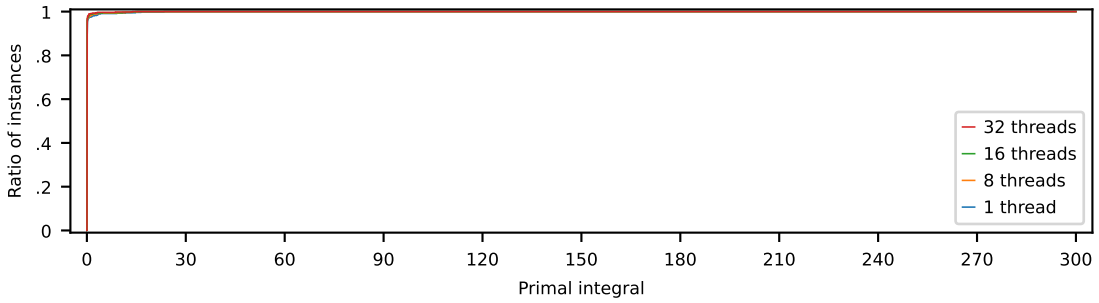


Figure E.19: The ratio of instances against the primal integral in the minimization of open stacks problem (MOSP) for CAHDBS2.

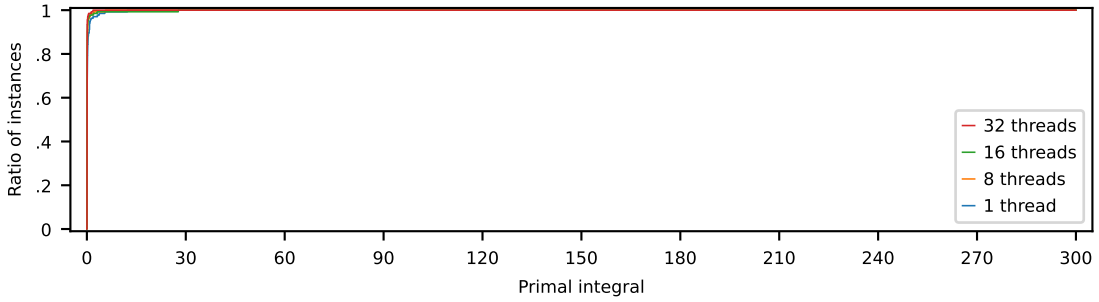


Figure E.20: The ratio of instances against the primal integral in graph-clear for CAHDBS2.

Bibliography

[1] T. S. Abdul-Razaq and C. N. Potts. "Dynamic Programming State-Space Relaxation for Single-Machine Scheduling". In: *The Journal of the Operational Research Society* 39.2 (1988), pp. 141–152. DOI: 10.1057/jors.1988.26.

[2] T. S. Abdul-Razaq, C. N. Potts, and L. N. Van Wassenhove. "A Survey of Algorithms for the Single Machine Total Weighted Tardiness Scheduling Problem". In: *Discrete Applied Mathematics* 26.2 (1990), pp. 235–253. DOI: 10.1016/0166-218X(90)90103-J.

[3] Tobias Achterberg. "Constraint Integer Programming". PhD thesis. Technische Universität Berlin, 2007.

[4] Forest Agostinelli, Stephen McAleer, Alexander Shmakov, and Pierre Baldi. "Solving the Rubik's Cube with Deep Reinforcement Learning and Search". In: *Nature Machine Intelligence* 1.8 (2019), pp. 356–363. DOI: 10.1038/s42256-019-0070-z.

[5] Sandip Aine, Siddharth Swaminathan, Venkatraman Narayanan, Victor Hwang, and Maxim Likhachev. "Multi-Heuristic A*". In: *International Journal on Robotics Research* 35.1–3 (2016), pp. 224–243. DOI: 10.1177/0278364915594029.

[6] Johannes Aldinger and Bernhard Nebel. "Interval Based Relaxation Heuristics for Numeric Planning with Action Costs". In: *KI 2017: Advances in Artificial Intelligence*. Cham: Springer International Publishing, 2017, pp. 15–28. DOI: 10.1007/978-3-319-67190-1_2.

[7] H. R. Andersen, T. Hadzic, J. N. Hooker, and P. Tiedemann. "A Constraint Store Based on Multivalued Decision Diagrams". In: *Principles and Practice of Constraint Programming – CP 2007*. 2007, pp. 118–132. DOI: 10.1007/978-3-540-74970-7_11.

[8] Krzysztof R. Apt and Mark Wallace. *Constraint Logic Programming using Eclipse*. Cambridge University Press. DOI: 10.1017/CBO9780511607400.

[9] N. Ascheuer, L. F. Escudero, M. Grötschel, and M. Stoer. "A Cutting Plane Approach to the Sequential Ordering Problem (with Applications to Job Scheduling in Manufacturing)". In: *SIAM Journal on Optimization* 3.1 (1993), pp. 25–42. DOI: 10.1137/0803002.

[10] Norbert Ascheuer. "Hamiltonian Path Problems in the On-Line Optimization of Flexible Manufacturing Systems". PhD thesis. Technische Universität Berlin, 1995.

[11] Norbert Ascheuer, Michael Jünger, and Gerhard Reinelt. "A Branch & Cut Algorithm for the Asymmetric Traveling Salesman Problem with Precedence Constraint". In: *Computational Optimization and Applications* 17 (2000), pp. 25–42. DOI: 10.1023/A:1008779125567.

[12] Behzad Babaki, Gilles Pesant, and Claude-Guy Quimper. "Solving Classical AI Planning Problems Using Planning-Independent CP Modeling and Search". In: *Proceedings of the 13th International Symposium on Combinatorial Search (SoCS)*. AAAI Press, 2020, pp. 2–10. DOI: 10.1609/socs.v11i1.18529.

[13] Christer Bäckström and Bernhard Nebel. "Complexity Results for SAS+ Planning". In: *Computational Intelligence* 11 (1995), pp. 625–656. DOI: 10.1111/j.1467-8640.1995.tb00052.x.

[14] Jorge A. Baier, Fahiem Bacchus, and Sheila A. McIlraith. "A Heuristic Search Approach to Planning with Temporally Extended Preferences". In: *Artificial Intelligence* 173.5 (2009). Advances in Automated Plan Generation, pp. 593–618. DOI: 10.1016/j.artint.2008.11.011.

[15] Egon Balas, Matteo Fischetti, and William R. Pulleyblank. "The Precedence-Constrained Asymmetric Traveling Salesman Polytope". In: *Mathematical Programming* 68 (1995), pp. 241–265. DOI: 10.1007/BF01585767.

[16] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. "New Route Relaxation and Pricing Strategies for the Vehicle Routing Problem". In: *Operations Research* 59.5 (2011), pp. 1269–1283. DOI: 10.1287/opre.1110.0975.

[17] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. "New State-Space Relaxations for Solving the Traveling Salesman Problem with Time Windows". In: *INFORMS Journal on Computing* 24.3 (2012), pp. 356–371. DOI: 10.1287/ijoc.1110.0456.

[18] J. Barnat, L. Brim, and J. Chaloupka. "Parallel Breadth-First Search LTL Model-Checking". In: *18th IEEE International Conference on Automated Software Engineering, 2003. Proceedings.* 2003, pp. 106–115. DOI: 10.1109/ASE.2003.1240299.

[19] Roman Barták, Lukáš Chrpa, Agostino Dovier, Jindřich Vodrážka, and Neng-Fa Zhou. "Modeling and Solving Planning Problems in Tabled Logic Programming: Experience from the Cave Diving Domain". In: *Science of Computer Programming* 147 (2017). Selected and Extended papers from the International Symposium on Principles and Practice of Declarative Programming 2015, pp. 54–77. DOI: 10.1016/j.scico.2017.04.007.

[20] Roman Barták and Daniel Toropila. "Reformulating Constraint Models for Classical Planning". In: *Proceedings of the 21st International Florida Artificial Intelligence Research Society Conference (FLAIRS)*. AAAI Press, 2008, pp. 525–530.

[21] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. "Learning to Act Using Real-Time Dynamic Programming". In: *Artificial Intelligence* 72.1 (1995), pp. 81–138. DOI: 10.1016/0004-3702(94)00011-O.

[22] İlker Baybars. "A Survey of Exact Algorithms for the Simple Assembly Line Balancing Problem". In: *Management Science* 32.8 (1986), pp. 909–932. DOI: 10.1287/mnsc.32.8.909.

[23] J. E. Beasley. "OR-Library: Distributing Test Problems by Electronic Mail". In: *The Journal of the Operational Research Society* 41.11 (1990), pp. 1069–1072. DOI: 10.2307/2582903.

[24] J. Christopher Beck. "Solution-Guided Multi-Point Constructive Search for Job Shop Scheduling". In: *Journal of Artificial Intelligence Research* 29 (2007), pp. 49–77. DOI: 10.1613/jair.2169.

[25] J. Christopher Beck and Mark S. Fox. "A Generic Framework for Constraint-Directed Search and Scheduling". In: *AI Magazine* 19.4 (1998), p. 103. DOI: 10.1609/aimag.v19i4.1426.

[26] J. Christopher Beck and Laurent Perron. "Discrepancy-Bounded Depth First Search". In: *Second International Workshop on Integration of AI and OR Technologies for Combinatorial Optimization Problems, CPAIOR 2000*. 2000.

[27] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[28] Richard Bellman. "Dynamic Programming Treatment of the Travelling Salesman Problem". In: *Journal of the ACM* 9.1 (1962), 61–63. DOI: 10.1145/321105.321111.

[29] Richard Bellman. "On a Routing Problem". In: *Quarterly of Applied Mathematics* 16.1 (1958), pp. 87–90. DOI: 10.1090/qam/102435.

[30] J. F. Benders. "Partitioning Procedures for Solving Mixed-Variables Programming Problems". In: *Numerische Mathematik* 4.1 (1962), pp. 238–252. DOI: 10.1007/BF01386316.

[31] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. "Machine Learning for Combinatorial Optimization: A Methodological Tour d'Horizon". In: *European Journal of Operational Research* 290.2 (2021), pp. 405–421. DOI: 10.1016/j.ejor.2020.07.063.

[32] David Bergman and Andre A. Cire. "Decomposition Based on Decision Diagrams". In: *Integration of AI and OR Techniques in Constraint Programming – 13th International Conference, CPAIOR 2016*. Cham: Springer International Publishing, 2016, pp. 45–54. DOI: 10.1007/978-3-319-33954-2_4.

[33] David Bergman and Andre A. Cire. "Discrete Nonlinear Optimization by State-Space Decompositions". In: *Management Science* 64.10 (2018), pp. 4700–4720. DOI: 10.1287/mnsc.2017.2849.

[34] David Bergman, Andre A. Cire, Ashish Sabharwal, Horst Samulowitz, Vijay Saraswat, and Willem-Jan van Hoeve. "Parallel Combinatorial Optimization with Decision Diagrams". In: *Integration of AI and OR Techniques in Constraint Programming – 11th International Conference, CPAIOR 2014*. Cham: Springer International Publishing, 2014, pp. 351–367. DOI: 10.1007/978-3-319-07046-9_25.

[35] David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and J. N. Hooker. "Discrete Optimization with Decision Diagrams". In: *INFORMS Journal on Computing* 28.1 (2016), pp. 47–66. DOI: 10.1287/ijoc.2015.0648.

[36] David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and J. N. Hooker. "Optimization Bounds from Binary Decision Diagrams". In: *INFORMS Journal on Computing* 26.2 (2014), pp. 253–268. DOI: 10.1287/ijoc.2013.0561.

[37] David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and Tallys Yunes. "BDD-Based Heuristics for Binary Optimization". In: *Journal of Heuristics* 20.2 (2014), pp. 211–234. DOI: 10.1007/s10732-014-9238-1.

[38] Donald A. Berry and Bert Fristedt. *Bandit Problems: Sequential Allocation of Experiments*. London: Chapman and Hall, 1985.

[39] Timo Berthold. "Measuring the Impact of Primal Heuristics". In: *Operations Research Letters* 41.6 (2013), pp. 611–614. DOI: 10.1016/j.orl.2013.08.007.

[40] Timo Berthold. "Primal Heuristics for Mixed Integer Programs". Master's Thesis. Technische Universität Berlin, 2006.

[41] Christian Bessiere. "Constraint Propagation". In: *Handbook of Constraint Programming*. Ed. by Francesca Rossi, Peter van Beek, and Toby Walsh. Vol. 2. Foundations of Artificial Intelligence. Elsevier, 2006. Chap. 3, pp. 29–83. DOI: 10.1016/S1574-6526(06)80007-6.

[42] R. S. Bird. "Tabulation Techniques for Recursive Programs". In: *ACM Computing Surveys* 12.4 (1980), pp. 403–417. DOI: 10.1145/356827.356831.

[43] Johannes Bisschop and Marcel Roelofs. "The Modeling Language AIMMS". In: *Modeling Languages in Mathematical Optimization*. Ed. by Josef Kallrath. Boston, MA: Springer, 2004, pp. 71–104. DOI: 10.1007/978-1-4613-0215-5_6.

[44] Gustav Björdal, Jean-Noël Monette, Pierre Flener, and Justin Pearson. "A Constraint-Based Local Search Backend for MiniZinc". In: *Constraints* 20.3 (2015), pp. 325–345. DOI: 10.1007/s10601-015-9184-z.

[45] Manuel Bodirsky. *Complexity of Infinite-Domain Constraint Satisfaction*. Vol. 52. Cambridge University Press, 2021.

[46] Natashia Boland, John Dethridge, and Irina Dumitrescu. "Accelerated Label Setting Algorithms for the Elementary Resource Constrained Shortest Path Problem". In: *Operations Research Letters* 34.1 (2006), pp. 58–68. DOI: 10.1016/j.orl.2004.11.011.

[47] Blai Bonet. "An Admissible Heuristic for SAS+ Planning Obtained from the State Equation". In: *Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI-13*. Menlo Park, California: AAAI Press/International Joint Conferences on Artificial Intelligence Orginization, 2013, pp. 2268–2274.

[48] Blai Bonet and Héctor Geffner. "Planning as Heuristic Search". In: *Artificial Intelligence* 129.1 (2001), pp. 5–33. DOI: 10.1016/S0004-3702(01)00108-4.

[49] Kyle E.C. Booth, Tony T. Tran, Goldie Nejat, and J. Christopher Beck. "Mixed-Integer and Constraint Programming Techniques for Mobile Robot Task Planning". In: *IEEE Robotics and Automation Letters* 1.1 (2016), pp. 500–507. DOI: 10.1109/LRA.2016.2522096.

[50] Djallel Bouneffouf, Irina Rish, and Charu Aggarwal. "Survey on Applications of Multi-Armed and Contextual Bandits". In: *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE Press, 2020, pp. 1–8. DOI: 10.1109/CEC48606.2020.9185782.

[51] Frédéric Boussemart, Christophe Lecoutre, Gilles Audemard, and Cédric Piette. *XCSP3-core: A Format for Representing Constraint Satisfaction/Optimization Problems*. 2022. arXiv: 2009.00514 [cs.AI].

[52] Craig Boutilier, Ray Reiter, and Bob Price. "Symbolic Dynamic Programming for First-Order MDPs". In: *Proceedings of the 17th International Joint Conference on Artificial Intelligence, IJCAI-01*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 690–697.

[53] Nils Boysen, Philipp Schulze, and Armin Scholl. "Assembly Line Balancing: What Happened in the Last Fifteen Years?" In: *European Journal of Operational Research* 301.3 (2021), pp. 797–814. DOI: 10.1016/j.ejor.2021.11.043.

[54] Yossi Bukchin and Tal Raviv. "Constraint Programming for Solving Various Assembly Line Balancing Problems". In: *Omega* 78 (2018), pp. 57–68. DOI: 10.1016/j.omega.2017.06.008.

[55] Ethan Burns, Sofia Lemons, Wheeler Ruml, and Rong Zhou. "Best-First Heuristic Search for Multicore Machines". In: *Journal of Artificial Intelligence Research* 39 (2010), pp. 689–743. DOI: 10.1613/jair.3094.

[56] Michael R. Bussieck and Alex Meeraus. "General Algebraic Modeling System (GAMS)". In: *Modeling Languages in Mathematical Optimization*. Ed. by Josef Kallrath. Boston, MA: Springer, 2004, pp. 137–157. DOI: 10.1007/978-1-4613-0215-5_8.

[57] Tom Bylander. "A Probabilistic Analysis of Prepositional STRIPS Planning". In: *Artificial Intelligence* 81.1 (1996). Frontiers in Problem Solving: Phase Transitions and Complexity, pp. 241–271. DOI: 10.1016/0004-3702(95)00055-0.

[58] Tom Bylander. "The Computational Complexity of Propositional STRIPS Planning". In: *Artificial Intelligence* 69.1 (1994), pp. 165–204. DOI: 10.1016/0004-3702(94)90081-7.

[59] Michael L. Bynum, Gabriel A. Hackebeil, William E. Hart, Carl D. Laird, Bethany L. Nicholson, John D. Siirola, Jean-Paul Watson, and David L. Woodruff. *Pyomo–Optimization Modeling in Python*. Third Edition. Vol. 67. Springer Optimization and Its Applications. Cham: Springer, 2021. DOI: 10.1007/978-3-030-68928-5.

[60] Nicolás Cabrera, Andrés L. Medaglia, Leonardo Lozano, and Daniel Duque. "An Exact Bidirectional Pulse Algorithm for the Constrained Shortest Path". In: *Networks* 76.2 (2020), pp. 128–146. DOI: 10.1002/net.21960.

[61] Valentina Cacchiani, Manuel Iori, Alberto Locatelli, and Silvano Martello. "Knapsack Problems — An Overview of Recent Advances. Part II: Multiple, Multidimensional, and Quadratic Knapsack Problems". In: *Computers & Operations Research* 143 (2022), p. 105693. DOI: 10.1016/j.cor.2021.105693.

[62] Quentin Cappart, Thierry Moisan, Louis-Martin Rousseau, Isabeau Prémont-Schwarz, and Andre A. Cire. "Combining Reinforcement Learning and Constraint Programming for Combinatorial Optimization". In: *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI)*. Palo Alto, California USA: AAAI Press, 2021, pp. 3677–3687. DOI: 10.1609/aaai.v35i5.16484.

[63] Matteo Cardellini, Enrico Giunchiglia, and Marco Maratea. "Symbolic Numeric Planning with Patterns". In: *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI)*. Washington, DC, USA: AAAI Press, 2024, pp. 20070–20077. DOI: 10.1609/aaai.v38i18.29985.

[64] Marco Antonio Moreira De Carvalho and Nei Yoshihiro Soma. "A Breadth-First Search Applied to the Minimization of the Open Stacks". In: *Journal of the Operational Research Society* 66.6 (2015), pp. 936–946. DOI: 10.1057/jors.2014.60.

[65] Margarita P. Castro, Andre A. Cire, and J. Christopher Beck. "An MDD-Based Lagrangian Approach to the Multicommodity Pickup-and-Delivery TSP". In: *INFORMS Journal on Computing* 32.2 (2020), pp. 263–278. DOI: 10.1287/ijoc.2018.0881.

[66] Margarita Paz Castro, Chiara Piacentini, Andre Augusto Cire, and J. Christopher Beck. "Solving Delete Free Planning with Relaxed Decision Diagram Based Heuristics". In: *Journal of Artificial Intelligence Research* 67 (2020). DOI: 10.1613/jair.1.11659.

[67] Alfonso Catalano, Stefania Gnesi, and Ugo Montanari. "Shortest Path Problems and Tree Grammars: An Algebraic Framework". In: *Graph-Grammars and Their Application to Computer Science and Biology*. Berlin, Heidelberg: Springer, 1979, pp. 167–179. DOI: 10.1007/BFb0025719.

[68] P.P. Chakrabarti, S. Ghose, A. Pandey, and S.C. De Sarkar. "Increasing Search Efficiency Using Multiple Heuristics". In: *Information Processing Letters* 30.1 (1989), pp. 33–36. DOI: 10.1016/0020-0190(89)90171-3.

[69] Félix Chalumeau, Ilan Coulon, Quentin Cappart, and Louis-Martin Rousseau. "SeaPearl: A Constraint Programming Solver Guided by Reinforcement Learning". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research – 18th International Conference, CPAIOR 2021*. Cham: Springer International Publishing, 2021, pp. 392–409. DOI: 10.1007/978-3-030-78230-6_25.

[70] Rohit Chandra, Leo Dagum, David Kohr, Ramesh Menon, Dror Maydan, and Jeff McDonald. *Parallel Programming in OpenMP*. Morgan kaufmann, 2001.

[71] Peter Cheeseman, Bob Kanefsky, and William M. Taylor. "Where the Really Hard Problems Are". In: *Proceedings of the 12th International Joint Conference on Artificial Intelligence, IJCAI-91*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1991, pp. 331–337.

[72] Dillon Z. Chen, Sylvie Thiébaux, and Felipe Trevizan. "Learning Domain-Independent Heuristics for Grounded and Lifted Planning". In: *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI)*. Washington, DC, USA: AAAI Press, 2024, pp. 20078–20086. DOI: 10.1609/aaai.v38i18.29986.

[73] Yixin Chen, Ruoyun Huang, Zhao Xing, and Weixiong Zhang. "Long-Distance Mutual Exclusion for Planning". In: *Artificial Intelligence* 173.2 (2009), pp. 365–391. DOI: 10.1016/j.artint.2008.11.004.

[74] T. C. E. Cheng, J. E. Diamond, and B. M. T. Lin. "Optimal Scheduling in Film Production to Minimize Talent Hold Cost". In: *Journal of Optimization Theory and Applications* 79.3 (1993), pp. 479–492. DOI: 10.1007/BF00940554.

[75] Antonia Chmiela, Ambros Gleixner, Pawel Lichocki, and Sebastian Pokutta. "Online Learning for Scheduling MIP Heuristics". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research – 20th International Conference, CPAIOR 2023*. Cham: Springer Nature Switzerland, 2023, pp. 114–123. DOI: 10.1007/978-3-031-33271-5_8.

[76] Nicos Christofides, A. Mingozzi, and P. Toth. "State-Space Relaxation Procedures for the Computation of Bounds to Routing Problems". In: *Networks* 11.2 (1981), pp. 145–164. DOI: 10.1002/net.3230110207.

[77] Geoffrey Chu, Maria Garcia de la Banda, and Peter J. Stuckey. "Exploiting Subproblem Dominance in Constraint Programming". In: *Constraints* 17 (2012), pp. 1–38. DOI: 10.1007/s10601-011-9112-9.

[78] Geoffrey Chu and Peter J. Stuckey. "Learning Value Heuristics for Constraint Programming". In: *Integration of AI and OR Techniques in Constraint Programming – 12th International Conference, CPAIOR 2015*. Cham: Springer International Publishing, 2015, pp. 108–123. DOI: 10.1007/978-3-319-18008-3_8.

[79] Geoffrey Chu and Peter J. Stuckey. "Minimizing the Maximum Number of Open Stacks by Customer Search". In: *Principles and Practice of Constraint Programming – CP 2009*. Berlin, Heidelberg: Springer, 2009, pp. 242–257. DOI: 10.1007/978-3-642-04244-7_21.

[80] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. "Weak, Strong, and Strong Cyclic Planning via Symbolic Model Checking". In: *Artificial Intelligence* 147.1 (2003). Planning with Uncertainty and Incomplete Information, pp. 35–84. DOI: 10.1016/S0004-3702(02)00374-0.

[81] Andre A. Cire and Willem-Jan van Hoeve. "Multivalued Decision Diagrams for Sequencing Problems". In: *Operations Research* 61.6 (2013), pp. 1411–1428. DOI: 10.1287/opre.2013.1221.

[82] Eldan Cohen and J. Christopher Beck. "Fat- and Heavy-Tailed Behavior in Satisficing Planning". In: *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*. Palo Alto, California USA: AAAI Press, 2018, pp. 6136–6143. DOI: 10.1609/aaai.v32i1.12092.

[83] Eldan Cohen and J. Christopher Beck. "Local Minima, Heavy Tails, and Search Effort for GBFS". In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence, IJCAI-18*. Main track. International Joint Conferences on Artificial Intelligence Organization, 2018, pp. 4708–4714. DOI: 10.24963/ijcai.2018/654.

[84] Eldan Cohen and J. Christopher Beck. "Problem Difficulty and the Phase Transition in Heuristic Search". In: *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*. Palo Alto, California USA: AAAI Press, 2017, pp. 780–786. DOI: 10.1609/aaai.v31i1.10658.

[85] Amanda Coles, Andrew Coles, Maria Fox, and Derek Long. "A Hybrid LP-RPG Heuristic for Modelling Numeric Resource Flows in Planning". In: *Journal of Artificial Intelligence Research* 46.1 (2013), pp. 343–412. DOI: 10.1613/jair.3788.

[86] Amanda Coles, Andrew Coles, Maria Fox, and Derek Long. "COLIN: Planning with Continuous Linear Numeric Change". In: *Journal of Artificial Intelligence Research* 44 (2012), pp. 1–96. DOI: 10.1613/jair.3608.

[87] Vianney Coppé, Xavier Gillard, and Pierre Schaus. "Boosting Decision Diagram-Based Branch-And-Bound by Pre-Solving with Aggregate Dynamic Programming". In: *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 13:1–13:17. DOI: 10.4230/LIPIcs.CP.2023.13.

[88] Vianney Coppé, Xavier Gillard, and Pierre Schaus. "Decision Diagram-Based Branch-and-Bound with Caching for Dominance and Suboptimality Detection". In: *INFORMS Journal on Computing* (2024). DOI: 10.1287/ijoc.2022.0340.

[89] Vianney Coppé, Xavier Gillard, and Pierre Schaus. "Modeling and Exploiting Dominance Rules for Discrete Optimization with Decision Diagrams". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research – 21st International Conference, CPAIOR 2024*. 2024.

[90] Jean-François Cordeau, Michel Gendreau, and Gilbert Laporte. "A Tabu Search Heuristic for Periodic and Multi-Depot Vehicle Routing Problems". In: *Networks* 30.2 (1997), pp. 105–119. DOI: 10.1002/(SICI)1097-0037(199709)30:2<105::AID-NET5>3.0.CO;2-G.

[91] "CPlan: A Constraint Programming Approach to Planning". In: *Proceedigns of the 16th National Conference on Artificial Intelligence (AAAI)*. Menlo Park, California: AAAI Press, 1999, pp. 585–590.

[92] Marco Daniele, Paolo Traverso, and Moshe Y. Vardi. "Strong Cyclic Planning Revisited". In: *Recent Advances in AI Planning, Proceedings of the Fifth European Conference on Planning (ECP)*. Berlin, Heidelberg: Springer, 1999, pp. 35–48. DOI: 10.1007/10720246_3.

[93] Emilie Danna and Laurent Perron. "Structured vs. Unstructured Large Neighborhood Search: A Case Study on Job-Shop Scheduling Problems with Earliness and Tardiness Costs". In: *Principles and Practice of Constraint Programming – CP 2003*. Berlin, Heidelberg: Springer, 2003, pp. 817–821. DOI: 10.1007/978-3-540-45193-8_59.

[94] Emilie Danna, Edward Rothberg, and Claude Le Pape. "Exploring Relaxation Induced Neighborhoods to Improve MIP Solutions". In: *Mathematical Programming* 102 (2005), pp. 71–90. DOI: 10.1007/s10107-004-0518-7.

[95] G. B. Dantzig and J. H. Ramser. "The Truck Dispatching Problem". In: *Management Science* 6.1 (1959), pp. 80–91. DOI: 10.1287/mnsc.6.1.80.

[96] George B. Dantzig. "Discrete-Variable Extremum Problems". In: *Operations Research* 5.2 (1957), pp. 266–277. DOI: 10.1287/opre.5.2.266.

[97] Rina Dechter. "Bucket Elimination: A Unifying Framework for Reasoning". In: *Artificial Intelligence* 113.1 (1999), pp. 41–85. DOI: 10.1016/S0004-3702(99)00059-4.

[98] Rina Dechter and Judea Pearl. "Generalized Best-First Search Strategies and the Optimality of A*". In: *Journal of the ACM* 32.3 (1985), pp. 505–536. DOI: 10.1145/3828.3830.

[99] Maxence Delorme, Manuel Iori, and Silvano Martello. "Bin Packing and Cutting Stock Problems: Mathematical Models and Exact Algorithms". In: *European Journal of Operational Research* 255.1 (2016), pp. 1–20. DOI: 10.1016/j.ejor.2016.04.030.

[100] Maxence Delorme, Manuel Iori, and Silvano Martello. "BPPLIB: A Library for Bin Packing and Cutting Stock Problems". In: *Optimization Letters* 12.2 (2018), pp. 235–250. DOI: 10.1007/s11590-017-1192-z.

[101] Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon. *Column Generation*. New York, NY: Springer. DOI: 10.1007/b135457.

[102] Guy Desaulniers, Oli B. G. Madsen, and Stefan Ropke. "The Vehicle Routing Problem with Time Windows". In: *Vehicle Routing: Problems, Methods, and Applications*. Ed. by Paolo Toth and Daniele Vigo. Second Edition. Society for Industrial and Applied Mathematics, 2014. Chap. 5, pp. 119–159. DOI: 10.1137/1.9781611973594.

[103] Martin Desrochers, Jacques Desrosiers, and Marius Solomon. "A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows". In: *Operations Research* 40.2 (1992), pp. 342–354. DOI: 10.1287/opre.40.2.342.

[104] Martin Desrochers and Francois Soumis. "A Generalized Permanent Labelling Algorithm For The Shortest Path Problem With Time Windows". In: *INFOR: Information Systems and Operational Research* 26.3 (1988), pp. 191–212. DOI: 10.1080/03155986.1988.11732063.

[105] Edsger W Dijkstra. "A Note on Two Problems in Connexion with Graphs". In: *Numerische Mathematik* 1 (1959), pp. 269–271. DOI: 10.1007/BF01386390.

[106] Minh Binh Do and Subbarao Kambhampati. "Planning as Constraint Satisfaction: Solving the Planning Graph by Compiling It Into CSP". In: *Artificial Intelligence* 132.2 (2001), pp. 151–182. DOI: 10.1016/S0004-3702(01)00128-X.

[107] J. E. Doran and D. Michie. "Experiments with the Graph Traverser Program". In: *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences* 294.1437 (1966), pp. 235–259. DOI: 10.1098/rspa.1966.0205.

[108] Dominik Drexler, Daniel Gnad, Paul Höft, Jendrik Seipp, David Speck, and Simon Ståhlberg. *Ragnarok*. 2023. URL: https://ipc2023-classical.github.io/abstracts/planner17_ragnarok.pdf.

[109] Yvan Dumas, Jacques Desrosiers, Eric Gelinas, and Marius M Solomon. "An Optimal Algorithm for the Traveling Salesman Problem with Time Windows". In: *Operations Research* 43.2 (1995), pp. 367–371. DOI: 10.1287/opre.43.2.367.

[110] S. Dutt and N.R. Mahapatra. "Scalable Load Balancing Strategies for Parallel A* Algorithms". In: *Journal of Parallel and Distributed Computing* 22.3 (1994), pp. 488–505. DOI: 10.1006/jpdc.1994.1106.

[111] Stefan Edelkamp. "External-Memory State Space Search". In: *Algorithm Engineering: Selected Results and Surveys*. Ed. by Lasse Kliemann and Peter Sanders. Cham: Springer International Publishing, 2016, pp. 185–225. DOI: 10.1007/978-3-319-49487-6_6.

[112] Stefan Edelkamp. "Planning with Pattern Databases". In: *Proceedings of the Sixth E (ECP)uropean Conference on Planning (ECP)*. 2001, pp. 13–24.

[113] Stefan Edelkamp and Jörg Hoffmann. *PDDL2.2: The Language for the Classical Part of the 4th International Planning Competitions*. Tech. rep. No. 195. Institut für Informatik, Albert-Ludwigs-Universität Freiburg, 2004.

[114] Stefan Edelkamp, Shahid Jabbar, and Alberto Lluch Lafuente. "Cost-Algebraic Heuristic Search". In: *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*. AAAI Press, 2005, pp. 1362–1367.

[115] Stefan Edelkamp and Stefan Schrödl. "Automatically Created Heuristics". In: *Heuristic Search*. San Francisco: Morgan Kaufmann, 2012. Chap. 4, pp. 161–192. DOI: 10.1016/B978-0-12-372512-7.00004-3.

[116] Stefan Edelkamp and Stefan Schrödl. *Heuristic Search: Theory and Applications*. San Francisco: Morgan Kaufmann, 2012. DOI: 10.1016/C2009-0-16511-X.

[117] Stefan Edelkamp and Stefan Schrödl. "Linear-Space Search". In: *Heuristic Search*. San Francisco: Morgan Kaufmann, 2012. Chap. 5, pp. 195–225. DOI: 10.1016/B978-0-12-372512-7.00005-5.

[118] Stefan Edelkamp and Stefan Schrödl. "Memory-Restricted Search". In: *Heuristic Search*. San Francisco: Morgan Kaufmann, 2012. Chap. 6, pp. 227–281. DOI: 10.1016/B978-0-12-372512-7.00006-7.

[119] Stefan Edelkamp and Stefan Schrödl. "Symbolic Search". In: *Heuristic Search*. San Francisco: Morgan Kaufmann, 2012. Chap. 7, pp. 283–318. DOI: 10.1016/B978-0-12-372512-7.00007-9.

[120] Stefan Edelkamp, Damian Sulewski, and Cengizhan Yücel. "Perfect Hashing for State Space Exploration on the GPU". In: *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press, 2010, pp. 57–64. DOI: 10.1609/icaps.v20i1.13414.

[121] Jason Eisner, Eric Goldlust, and Noah A. Smith. "Compiling Comp Ling: Weighted Dynamic Programming and the Dyna Language". In: *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*. USA: Association for Computational Linguistics, 2005, pp. 281–290. DOI: 10.3115/1220575.1220611.

[122] Dwidier El Baz, Bilal Fakih, Romeo Sanchez Nigenda, and Vincent Boyer. "Parallel Best-First Search Algorithms for Planning Problems on Multi-Core Processors". In: *The Journal of Supercomputing* 78.3 (2022), pp. 3122–3151. DOI: 10.1007/s11227-021-03986-z.

[123] Hamilton Emmons. "One-Machine Sequencing to Minimize Certain Functions of Job Tardiness". In: *Operations Research* 17.4 (1969), pp. 701–715. DOI: 10.1287/opre.17.4.701.

[124] Gary D. Eppen and R. Kipp Martin. "Solving Multi-Item Capacitated Lot-Sizing Problems Using Variable Redefinition". In: *Operations Research* 35.6 (1987), pp. 832–848. DOI: 10.1287/opre.35.6.832.

[125] L.F. Escudero. "An Inexact Algorithm for the Sequential Ordering Problem". In: *European Journal of Operational Research* 37.2 (1988), pp. 236–249. DOI: 10.1016/0377-2217(88)90333-5.

[126] Jason Evans. "A Scalable Concurrent malloc(3) Implementation for FreeBSD". In: *Proceedigns of BSDCan 2006*. 2006.

[127] M. Evett, J. Hendler, A. Mahanti, and D. Nau. "PRA*: Massively Parallel Heuristic Search". In: *Journal of Parallel and Distributed Computing* 25.2 (1995), pp. 133–143. DOI: 10.1006/jpdc.1995.1036.

[128] Patrick Eyerich, Robert Mattmüller, and Gabriele Röger. "Using the Context-enhanced Additive Heuristic for Temporal and Numeric Planning". In: *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press, 2009, pp. 130–137. DOI: 10.1609/icaps.v19i1.13373.

[129] Enrico Faggioli and Carlo Alberto Bentivoglio. "Heuristic and Exact Methods for the Cutting Sequencing Problem". In: *European Journal of Operational Research* 110.3 (1998), pp. 564–575. DOI: 10.1016/S0377-2217(97)00268-3.

[130] Emanuel Falkenauer. "A Hybrid Grouping Genetic Algorithm for Bin Packing". In: *Journal of Heuristics* 2.1 (1996), pp. 5–30. DOI: 10.1007/BF00226291.

[131] Dieqiao Feng, Carla P Gomes, and Bart Selman. "Left Heavy Tails and the Effectiveness of the Policy and Value Networks in DNN-based best-first search for Sokoban Planning". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 35. Curran Associates, Inc., 2022, pp. 36295–36307.

[132] Patrick Ferber, Florian Geißer, Felipe Trevizan, Malte Helmert, and Jörg Hoffmann. "Neural Network Heuristic Functions for Classical Planning: Bootstrapping and Comparison to Other Methods". In: *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS)*. 2022, pp. 583–587. DOI: 10.1609/icaps.v32i1.19845.

[133] Patrick Ferber, Florian Geißer, Felipe Trevizan, Malte Helmert, and Jörg Hoffmann. "Neural Network Heuristic Functions for Classical Planning: Reinforcement Learning and Comparison to Other Methods". In: *PRL Workshop Series – Bridging the Gap Between AI Planning and Reinforcement Learning*. 2021. URL: https://prl-theworkshop.github.io/prl2021/papers/PRL2021_paper_20.pdf.

[134] Patrick Ferber, Malte Helmert, and Jörg Hoffmann. "Neural Network Heuristics for Classical Planning: A Study of Hyperparameter Space". In: *ECAI 2020 – 24th European Conference on Artificial Intelligence*. Vol. 325. Frontiers in Artificial Intelligence and Applications. IOS Press, 2020, pp. 2346–2353. DOI: 10.3233/FAIA200364.

[135] Patrick Ferber, Malte Helmert, and Jörg Hoffmann. "Reinforcment Learning for Planning Heuristics". In: *Proceedings of the First Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL)*. 2020, pp. 119–126.

[136] Maximilian Fickert, Tianyi Gu, and Wheeler Ruml. "New Results in Bounded-Suboptimal Search". In: *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI)*. Palo Alto, California USA: AAAI Press, 2022, pp. 10166–10173. DOI: 10.1609/aaai.v36i9.21256.

[137] Richard E. Fikes and Nils J. Nilsson. "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving". In: *Artificial Intelligence* 2.3 (1971), pp. 189–208. DOI: 10.1016/0004-3702(71)90010-5.

[138] L. R. Ford. *Network Flow Theory*. Santa Monica, CA: RAND Corporation, 1956.

[139] Robert Fourer, David M. Gay, and Brian W. Kernighan. "AMPL: A Mathematical Programming Language". In: *Algorithms and Model Formulations in Mathematical Programming*. Ed. by Stein W. Wallace. Berlin, Heidelberg: Springer, 1989, pp. 150–151.

[140] Maria Fox and Derek Long. "Modelling Mixed Discrete-Continuous Domains for Planning". In: *Jorunal of Artifiicial Intelligence Research* 27 (2006), pp. 235–297. DOI: 10.1613/jair.2044.

[141] Maria Fox and Derek Long. "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains". In: *Journal of Artificial Intelligence Research* 20 (2003), pp. 61–124. DOI: 10.1613/jair.1129.

[142] Guillem Francès and Hector Geffner. "Modeling and Computation in Planning: Better Heuristics from More Expressive Languages". In: *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*. 2015, pp. 70–78. DOI: 10.1609/icaps.v25i1.13722.

[143] Guillem Francès, Miquel Ramirez, and Collaborators. *Tarski: An AI Planning Modeling Framework*. https://github.com/aig-upf/tarski. 2018.

[144] M. Frantzeskakis and C. D. T. Watson-Gandy. "The Use of State Space Relaxation for the Dynamic Facility Location Problem". In: *Annals of Operations Research* 18.1 (1989), pp. 187–211. DOI: 10.1007/BF02097803.

[145] Eugene Freuder. "In Pursuit of the Holy Grail". In: *Constraints* 2.1 (1997), pp. 57–61. DOI: 10.1023/A:1009749006768.

[146] Rafael de Magalhães Dias Frinhani, Marco Antonio Moreira de Carvalho, and Nei Yoshihiro Soma. "A PageRank-Based Heuristic for the Minimization of Open Stacks Problem". In: *PLoS ONE* 13.8 (2018), pp. 1–24. DOI: 10.1371/journal.pone.0203076.

[147] Alan M. Frisch, Warwick Harvey, Chris Jefferson, Bernadette Martínez-Hernández, and Ian Miguel. "ESSENCE: A Constraint Language for Specifying Combinatorial Problems". In: *Constraints* 13.3 (2008), pp. 268–306. DOI: 10.1007/s10601-008-9047-y.

[148] Nikolaus Frohner, Jan Gmys, Nouredine Melab, Günther Raidl, and El-Ghazali Talbi. "Parallel Beam Search for Combinatorial Optimization". In: *Workshop Proceedings of the 51st International Conference on Parallel Processing*. 2023. DOI: 10.1145/3547276.3548633.

[149] David A Furcy. "ITSA*: Iterative Tunneling Search with A*". In: *Heuristic Search, Memory-Based Heuristics and Their Applications: Papers from AAAI Workshop*. Menlo Park, California: AAAI Press, 2006, pp. 21–26.

[150] Éric Fusy. "Uniform Random Sampling of Planar Graphs in Linear Time". In: *Random Structures and Algorithms* 35.4 (2009), pp. 464–522. DOI: 10.1002/rsa.20275.

[151] S. L. Gadegaard and J. Lysgaard. "A Symmetry-Free Polynomial Formulation of the Capacitated Vehicle Routing Problem". In: *Discrete Applied Mathematics* 296 (2021), pp. 179–192. DOI: 10.1016/j.dam.2020.02.012.

[152] Maria Garcia de la Banda and Peter J. Stuckey. "Dynamic Programming to Minimize the Maximum Number of Open Stacks". In: *INFORMS Journal on Computing* 19.4 (2007), pp. 607–617. DOI: 10.1287/ijoc.1060.0205.

[153] Maria Garcia de la Banda, Peter J. Stuckey, and Geoffrey Chu. "Solving Talent Scheduling with Dynamic Programming". In: *INFORMS Journal on Computing* 23.1 (2011), pp. 120–137. DOI: 10.1287/ijoc.1090.0378.

[154] Ángel García Olaya, Sergio Jiménez, and Carlos Linares López. *The 2011 International Planning Competition - Description of Participant Planners*. 2011. URL: http://www.plg.inf.uc3m.es/ipc2011-deterministic/attachments/ParticipatingPlanners/ipc2011-booklet.pdf.

[155] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman and Company, 1979.

[156] Antonio Garrido, Marlene Arangu, and Eva Onaindia. "A Constraint Programming Formulation for Planning: from Plan Scheduling to Plan Generation". In: *Journal of Scheduling* 12.3 (2009), pp. 227–256. DOI: 10.1007/s10951-008-0083-7.

[157] Bezalel Gavish and Stephen C. Graves. *The Travelling Salesman Problem and Related Problems*. Tech. rep. Working Paper OR 078-78. Operations Research Center, Massachusetts Institute of Technology, 1978.

[158] Dongdong Ge, Qi Huangfu, Zizhuo Wang, Jian Wu, and Yinyu Ye. *Cardinal Optimizer (COPT) User Guide*. https://guide.coap.online/copt/en-doc. 2022.

[159] Héctor Geffner. "Functional Strips: A More Flexible Language for Planning and Problem Solving". In: *Logic-Based Artificial Intelligence*. Ed. by Jack Minker. Boston, MA: Springer US, 2000, pp. 187–209. DOI: 10.1007/978-1-4615-1567-8_9.

[160] Michel Gendreau, Alain Hertz, Gilbert Laporte, and Mihnea Stan. "A Generalized Insertion Heuristic for the Traveling Salesman Problem with Time Windows". In: *Operations Research* 46.3 (1998), pp. 330–346. DOI: 10.1287/opre.46.3.330.

[161] Ian P. Gent, Chris Jefferson, and Ian Miguel. "MINION: A Fast, Scalable, Constraint Solver". In: *ECAI 2006 – 17th European Conference on Artificial Intelligence*. Vol. 141. Frontiers in Artificial Intelligence and Applications. NLD: IOS Press, 2006, pp. 98–102.

[162] Rebecca Gentzel, Laurent Michel, and W.-J. van Hoeve. "HADDOCK: A Language and Architecture for Decision Diagram Compilation". In: *Principles and Practice of Constraint Programming – CP 2020*. Cham: Springer International Publishing, 2020, pp. 531–547. DOI: 10.1007/978-3-030-58475-7_31.

[163] Alfonso Gereveni and Derek Long. *Plan Constraints and Preferences in PDDL3*. Tech. rep. Department of Electronics for Automation, University of Brescia, 2005.

[164] Malik Ghallab, Adele Howe, Craig Knoblock, Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. *PDDL - The Planning Domain Definition Language*. Tech. rep. CVC TR-98-003/DCS TR-1165. Yale Center for Computational Vison and Control, 1998.

[165] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning*. The Morgan Kaufmann Series in Artificial Intelligence. Burlington: Morgan Kaufmann, 2004. DOI: 10.1016/B978-1-55860-856-6.X5000-5.

[166] Nina Ghanbari Ghooshchi, Majid Namazi, M. A. Hakim Newton, and Abdul Sattar. "Encoding Domain Transitions for Constraint-Based Planning". In: *Journal of Artificial Intelligence Research* 58 (2017), pp. 905–966. DOI: 10.1613/jair.5378.

[167] Robert Giegerich and Carsten Meyer. "Algebraic Dynamic Programming". In: *Proceedings of the Ninth Algebraic Methodology and Software Technology*. Berlin, Heidelberg: Springer, 2002, pp. 349–364. DOI: 10.1007/3-540-45719-4_24.

[168] Nicola Gigante and Enrico Scala. "On the Compilability of Bounded Numeric Planning". In: *Proceedings of the 32nd International Joint Conference on Artificial Intelligence, IJCAI-23*. Main track. International Joint Conferences on Artificial Intelligence Organization, 2023, pp. 5341–5349. DOI: 10.24963/ijcai.2023/593.

[169] Xavier Gillard, Vianney Coppé, Pierre Schaus, and André Augusto Cire. "Improving the Filtering of Branch-and-Bound MDD Solver". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research – 18th International Conference, CPAIOR 2021*. Cham: Springer International Publishing, 2021, pp. 231–247. DOI: 10.1007/978-3-030-78230-6_15.

[170] Xavier Gillard and Pierre Schaus. "Large Neighborhood Search with Decision Diagrams". In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI-22*. Main track. International Joint Conferences on Artificial Intelligence Organization, 2022, pp. 4754–4760. DOI: 10.24963/ijcai.2022/659.

[171] Xavier Gillard, Pierre Schaus, and Vianney Coppé. "Ddo, a Generic and Efficient Framework for MDD-Based Optimization". In: *Proceedings of the 29th International Joint Conference on Artificial Intelligence, IJCAI-20*. Demos. International Joint Conferences on Artificial Intelligence Organization, 2020, pp. 5243–5245. DOI: 10.24963/ijcai.2020/757.

[172] Michael Gimelfarb, Ayal Taitler, and Scott Sanner. "JaxPlan and GurobiPlan: Optimization Baselines for Replanning in Discrete and Mixed Discrete and Continuous Probabilistic Domains". In: *Proceedings in the 34th International Conference on Automated Planning and Scheduling (ICAPS)*. Washington, DC, USA: AAAI Press, 2024, pp. 230–238. DOI: 10.1609/icaps.v34i1.31480.

[173] Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, Marco Lübbecke, Hans D. Mittelmann, Derya Ozyurt, Ted K. Ralphs, Domenico Salvagnin, and Yuji Shinano. "MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library". In: *Mathematical Programming Computation* 13 (2021), pp. 443–490. DOI: 10.1007/s12532-020-00194-3.

[174] Daniel Gnad, Malte Helmert, Peter Jonsson, and Alexander Shleyfman. "Planning over Integers: Compilations and Undecidability". In: *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS)*. Palo Alto, California USA: AAAI Press, 2023, pp. 148–152. DOI: 10.1609/icaps.v33i1.27189.

[175] Stefania Gnesi, Ugo Montanari, and Alberto Martelli. "Dynamic Programming as Graph Searching: An Algebraic Approach". In: *Journal of the ACM* 28.4 (1981), 737–751. DOI: 10.1145/322276.322285.

[176] Bruce L. Golden, Larry Levy, and Rakesh Vohra. "The Orienteering Problem". In: *Naval Research Logistics (NRL)* 34.3 (1987), pp. 307–318. DOI: 10.1002/1520-6750(198706)34:3<307::AID-NAV3220340302>3.0.CO;2-D.

[177] Arnoosh Golestanian, Giovanni Lo Bianco, Chengyu Tao, and J. Christopher Beck. "Optimization Models for Pickup-And-Delivery Problems with Reconfigurable Capacities". In: *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*. Vol. 280. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 17:1–17:17. DOI: 10.4230/LIPIcs.CP.2023.17.

[178] Carla P. Gomes and Bart Selman. "Algorithm Portfolios". In: *Artificial Intelligence* 126.1 (2001). Tradeoffs under Bounded Resources, pp. 43–62. DOI: 10.1016/S0004-3702(00)00081-3.

[179] Carla P. Gomes, Bart Selman, and Nuno Crato. "Heavy-Tailed Distributions in Combinatorial Search". In: *Principles and Practice of Constraint Programming – CP97*. Berlin, Heidelberg: Springer, 1997, pp. 121–135. DOI: 10.1007/BFb0017434.

[180] Carla P. Gomes, Bart Selman, Nuno Crato, and Henry Kautz. "Heavy-Tailed Phenomena in Satisfiability and Constraint Satisfaction Problems". In: *Journal of Automated Reasoning* 24.1 (2000), pp. 67–100. DOI: 10.1023/A:1006314320276.

[181] Carla P. Gomes, Bart Selman, and Henry Kautz. "Boosting Combinatorial Search Through Randomization". In: *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI)*. AAAI Press, 1998, pp. 431–437.

[182] Jaime E. González, Andre A. Cire, Andrea Lodi, and Louis-Martin Rousseau. "Integrated Integer Programming and Decision Diagram Search Tree with an Application to the Maximum Independent Set Problem". In: *Constraints* 25.1 (2020), pp. 23–46. DOI: 10.1007/s10601-019-09306-w.

[183] Luis Gouveia, Ana Paias, and Stefan Voß. "Models for a Traveling Purchaser Problem with Additional Side-Constraints". In: *Computers & Operations Research* 38.2 (2011), pp. 550–558. DOI: 10.1016/j.cor.2010.07.016.

[184] Luis Gouveia and Mario Ruthmair. "Load-Dependent and Precedence-Based Models for Pickup and Delivery Problems". In: *Computers & Operations Research* 63 (2015), pp. 56–71. DOI: 10.1016/j.cor.2015.04.008.

[185] Peter Gregory, Derek Long, and Maria Fox. "Constraint Based Planning with Composable Substate Graphs". In: *ECAI 2010 – 19th European Conference on Artificial Intelligence*. Vol. 215. Frontiers in Artificial Intelligence and Applications. IOS Press, 2010, pp. 453–458. DOI: 10.3233/978-1-60750-606-5-453.

[186] Peter Gregory, Derek Long, Maria Fox, and J. Christopher Beck. "Planning Modulo Theories: Extending the Planning Paradigm". In: *Proceedings of the 22nd International Conference on Automated Planning and Scheduling*. AAAI Press, 2012, pp. 65–73. DOI: 10.1609/icaps.v22i1.13505.

[187] J. Gromicho, J. J. Van Hoorn, A. L. Kok, and J. M.J. Schutten. "Restricted Dynamic Programming: A Flexible Framework for Solving Realistic VRPs". In: *Computers & Operations Research* 39.5 (2012), pp. 902–909. DOI: 10.1016/j.cor.2011.07.002.

[188] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2023. URL: https://www.gurobi.com.

[189] Timothy L. Harris. "A Pragmatic Implementation of Non-blocking Linked-Lists". In: *Proceedings of the 15th International Conference on Distributed Computing. DISC 2001*. Berlin, Heidelberg: Springer-Verlag, 2001, pp. 300–314. DOI: 10.1007/3-540-45414-4_21.

[190] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. DOI: 10.1109/TSSC.1968.300136.

[191] William E Hart, Jean-Paul Watson, and David L Woodruff. "Pyomo: Modeling and Solving Mathematical Programs in Python". In: *Mathematical Programming Computation* 3.3 (2011), pp. 219–260.

[192] William D. Harvey and Matthew L. Ginsberg. "Limited Discrepancy Search". In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI-95*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, pp. 607–613.

[193] Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig. "Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning". In: *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Press, 2007, pp. 1007–1012.

[194] Patrik Haslum and Héctor Geffner. "Admissible Heuristics for Optimal Planning". In: *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS)*. AAAI Press, 2000, pp. 140–149.

[195] Emmanuel Hebrard. "Mistral, a Constraint Satisfaction Library". In: *Proceedings of the Third International CSP Solver Competition*. 2008, pp. 31–39.

[196] Emmanuel Hebrard, Eoin O'Mahony, and Barry O'Sullivan. "Constraint Programming and Combinatorial Optimisation in Numberjack". In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems – 7th International Conference, CPAIOR 2010*. Berlin, Heidelberg: Springer, 2010, pp. 181–185. DOI: 10.1007/978-3-642-13520-0_22.

[197] Stefan Heinz, Wen-Yang Ku, and J. Christopher Beck. "Recent Improvements Using Constraint Integer Programming for Resource Allocation and Scheduling". In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR 2013*. Berlin, Heidelberg: Springer, 2013, pp. 12–27. DOI: 10.1007/978-3-642-38171-3_2.

[198] Michael Held and Richard M. Karp. "A Dynamic Programming Approach to Sequencing Problems". In: *Journal of the Society for Industrial and Applied Mathematics* 10.1 (1962), pp. 196–210. DOI: 10.1137/0110015.

[199] Michael Held and Richard M. Karp. "The Traveling-Salesman Problem and Minimum Spanning Trees". In: *Operations Research* 18.6 (1970), pp. 1138–1162. DOI: 10.1287/opre.18.6.1138.

[200] Daniel Heller, Patrick Ferber, Julian Bitterwolf, Matthias Hein, and Jörg Hoffmann. "Neural Network Heuristic Functions: Taking Confidence into Account". In: *Proceedings of the 17th International Symposium on Combinatorial Search (SoCS)*. AAAI Press, 2022, pp. 223–228. DOI: 10.1609/socs.v15i1.21771.

[201] Paul Helman. "A New Theory of Dynamic Programming." PhD thesis. University of Michigan, Ann Arbor, 1982.

[202] Malte Helmert. "Decidability and Undecidability Results for Planning with Numerical State Variables". In: *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems (AIPS)*. AAAI Press, 2002, pp. 44–53.

[203] Malte Helmert. "The Fast Downward Planning System". In: *Journal of Artificial Intelligence Research* 26 (2006), pp. 191–246. DOI: 10.1613/jair.1705.

[204] Malte Helmert, Patrik Haslum, Jörg Hoffmann, and Raz Nissim. "Merge-and-Shrink Abstraction". In: *Journal of the ACM* 61.3 (2014), pp. 1–63. DOI: 10.1145/2559951.

[205] Gregor Hendel. "Adaptive Large Neighborhood Search for Mixed Integer Programming". In: *Mathematical Programming Computation* 14 (2022), pp. 185–221. DOI: 10.1007/s12532-021-00209-7.

[206] Maurice Herlihy. "A Methodology for Implementing Highly Concurrent Data Objects". In: *ACM Transactions on Programing Languages Systems* 15.5 (1993), pp. 745–770. DOI: 10.1145/161468.161469.

[207] István T. Hernádvölgyi, Robert C. Holte, and Toby Walsh. "Experiments with Automatically Created Memory-Based Heuristics". In: *Abstraction, Reformulation, and Approximation. SARA 2000*. Berlin, Heidelberg: Springer, 2000, pp. 281–290. DOI: 10.1007/3-540-44914-0_18.

[208] Hipólito Hernández-Pérez and Juan José Salazar-González. "The Multi-Commodity One-to-One Pickup-and-Delivery Traveling Salesman Problem". In: *European Journal of Operational Research* 196.3 (2009), pp. 987–995. DOI: 10.1016/j.ejor.2008.05.009.

[209] Frederick S Hillier and Gerald J Lieberman. *Introduction to Operations Research*. McGraw-Hill, 2015.

[210] Samid Hoda, Willem-Jan van Hoeve, and J. N. Hooker. "A Systematic Approach to MDD-Based Constraint Programming". In: *Principles and Practice of Constraint Programming – CP 2010*. Berlin, Heidelberg: Springer, 2010, pp. 266–280. DOI: 10.1007/978-3-642-15396-9_23.

[211] Jesse Hoey, Robert St-Aubin, Alan J. Hu, and Craig Boutilier. "SPUDD: Stochastic Planning using Decision Diagrams". In: *UAI '99: Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 279–288.

[212] Jörg Hoffmann. "The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables". In: *Journal of Artificial Intelligence Research* 20 (2003), pp. 291–341. DOI: 10.1613/jair.1144.

[213] Jörg Hoffmann and Bernhard Nebel. "The FF Planning System: Fast Plan Generation Through Heuristic Search". In: *Journal of Artificial Intelligence Research* 14 (2001), pp. 253–302. DOI: 10.1613/jair.855.

[214] Robert Holte and Gaojin Fan. "State Space Abstraction in Artificial Intelligence and Operations Research". In: *Planning, Search, and Optimization: Papers from the 2015 AAAI Workshop*. 2015, pp. 55–60.

[215] J. N. Hooker and G. Ottosson. "Logic-Based Benders Decomposition". In: *Mathematical Programming* 96.1 (2003), pp. 33–60. DOI: 10.1007/s10107-003-0375-9.

[216] John Hooker. "Nonserial Dynamic Programming". In: *Logic-Based Methods for Optimization*. John Wiley & Sons, Ltd, 2000. Chap. 20, pp. 423–441. DOI: 10.1002/9781118033036.ch20.

[217] John N. Hooker. "Decision Diagrams and Dynamic Programming". In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems – 10th International Conference, CPAIOR 2013*. Berlin, Heidelberg: Springer, 2013, pp. 94–110. DOI: 10.1007/978-3-642-38171-3_7.

[218] Holger H. Hoos and Edward Tsang. "Local Search Methods". In: *Handbook of Constraint Programming*. Ed. by Francesca Rossi, Peter van Beek, and Toby Walsh. Vol. 2. Foundations of Artificial Intelligence. Elsevier, 2006. Chap. 5, pp. 135–167. DOI: 10.1016/S1574-6526(06)80009-X.

[219] Ronald A. Howard. "Dynamic Programming and Markov Process". In: New York: John Wiley & Sons, Inc., 1960.

[220] Ruoyun Huang, Yixin Chen, and Weixiong Zhang. "A Novel Transition Based Encoding Scheme for Planning as Satisfiability". In: *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI)*. Palo Alto, California USA: AAAI Press, 2010, pp. 89–94. DOI: 10.1609/aaai.v24i1.7544.

[221] Q. Huangfu and J. A. J. Hall. "Parallelizing the Dual Revised Simplex Method". In: *Mathematical Programming Computation* 10.1 (2018), pp. 119–142. DOI: 10.1007/s12532-017-0130-5.

[222] Philipp Hungerländer and Christian Truden. "Efficient and Easy-to-Implement Mixed-Integer Linear Programs for the Traveling Salesperson Problem with Time Windows". In: *Transportation Research Procedia* 30 (2018), pp. 157–166. DOI: 10.1016/j.trpro.2018.09.018.

[223] Daniel Höller and Gregor Behnke. "Encoding Lifted Classical Planning in Propositional Logic". In: *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS)*. Palo Alto, California USA: AAAI Press, 2022, pp. 134–144. DOI: 10.1609/icaps.v32i1.19794.

[224] Daniel Höller, Gregor Behnke, Pascal Bercher, Susanne Biundo, Humbert Fiorino, Damien Pellier, and Ron Alford. "HDDL: An Extension to PDDL for Expressing Hierarchical Planning Problems". In: *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*. Palo Alto, California USA: AAAI Press, 2020, pp. 9883–9891. DOI: 10.1609/aaai.v34i06.6542.

[225] Toshihide Ibaraki. "Branch-and-Bound Procedure and State-Space Representation of Combinatorial Optimization Problems". In: *Information and Control* 36.1 (1978), pp. 1–27. DOI: 10.1016/S0019-9958(78)90197-3.

[226] Toshihide Ibaraki. "Classes of Discrete Optimization Problems and Their Decision Problems". In: *Journal of Computer and System Sciences* 8.1 (1974), pp. 84–116. DOI: 10.1016/S0022-0000(74)80024-3.

[227] Toshihide Ibaraki. "Finite State Representations of Discrete Optimization Problems". In: *SIAM Journal on Computing* 2.3 (1973), pp. 193–210. DOI: 10.1137/0202016.

[228] Toshihide Ibaraki. "Representation Theorems for Equivalent Optimization Problems". In: *Information and Control* 21.5 (1972), pp. 397–435. DOI: 10.1016/S0019-9958(72)90125-8.

[229] Toshihide Ibaraki. "Solvable Classes of Discrete Dynamic Programming". In: *Journal of Mathematical Analysis and Applications* 43.3 (1973), pp. 642–693. DOI: 10.1016/0022-247X(73)90283-7.

[230] IBM Corporation, White Plains, N.Y. *Mathematical Programming System/360 Version 2, Linear and Separable Programming—User's Manual*. Publication H20-0476-2. 1969.

[231] Tatsuya Imai and Alex Fukunaga. "On a Practical, Integer-Linear Programming Model for Delete-FreeTasks and its Use as a Heuristic for Cost-Optimal Planning". In: *Journal of Artificial Intelligence Research* 5 (2015), pp. 631–677. DOI: 10.1613/jair.4936.

[232] Stefan Irnich and Guy Desaulniers. "Shortest Path Problems with Resource Constraints". In: *Column Generation*. Ed. by Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon. Boston, MA: Springer, 2005, pp. 33–65. DOI: 10.1007/0-387-25486-2_2.

[233] James R. Jackson. "A Computing Procedure for a Line Balancing Problem". In: *Management Science* 2.3 (1956), pp. 261–271. DOI: 10.1287/mnsc.2.3.261.

[234] Siddhartha Jain and Pascal Van Hentenryck. "Large Neighborhood Search for Dial-a-Ride Problems". In: *Principles and Practice of Constraint Programming – CP 2011*. Berlin, Heidelberg: Springer, 2011, pp. 400–413. DOI: 10.1007/978-3-642-23786-7_31.

[235] Yuu Jinnai and Alex Fukunaga. "On Hash-Based Work Distribution Methods for Parallel Best-First Search". In: *Journal of Artifiicial Intelligence Research* 60 (2017), pp. 491–548. DOI: 10.1613/jair.5225.

[236] Roger V. Johnson. "Optimally Balancing Large Assembly Lines with 'Fable'". In: *Management Science* 34.2 (1988), pp. 240–253. DOI: 10.1287/mnsc.34.2.240.

[237] J. J. Kanet. "New Precedence Theorems for One-Machine Weighted Tardiness". In: *Mathematics of Operations Research* 32.3 (2007), pp. 579–588. DOI: 10.1287/moor.1070.0255.

[238] Marisa G. Kantor and Moshe B. Rosenwein. "The Orienteering Problem with Time Windows". In: *The Journal of the Operational Research Society* 43.6 (1992), pp. 629–635. DOI: 10.1057/jors.1992.88.

[239] Gio K. Kao, Edward C. Sewell, and Sheldon H. Jacobson. "A Branch, Bound, and Remember Algorithm for the $1|r_t|\sum t_i$ Scheduling Problem". In: *Journal of Scheduling* 12.2 (2009), pp. 163–175. DOI: 10.1007/s10951-008-0087-3.

[240] Richard M. Karp. "Reducibility among Combinatorial Problems". In: *Complexity of Computer Computations: Proceedings of a Symposium on the Complexity of Computer Computations*. Boston, MA: Springer US, 1972, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2_9.

[241] Richard M. Karp and Michael Held. "Finite-State Processes and Dynamic Programming". In: *SIAM Journal on Applied Mathematics* 15.3 (1967), pp. 693–718. DOI: 10.1137/0115060.

[242] Michael Katz and Carmel Domshlak. "Optimal Admissible Composition of Abstraction Heuristics". In: *Artificial Intelligence* 174.12 (2010), pp. 767–798. DOI: 10.1016/j.artint.2010.04.021.

[243] Henry Kautz and Bart Selman. "Planning as Satisfiability". In: *ECAI '92: Proceedings of the 10th European Conference on Artificial Intelligence*. USA: John Wiley & Sons, Inc., 1992, pp. 359–363.

[244] Henry Kautz and Bart Selman. "Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search". In: *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI)*. 1996, pp. 1194–1201.

[245] Henry Kautz and Bart Selman. "Unifying SAT-Based and Graph-Based Planning". In: *Proceedings of the 16th International Joint Conference on Artifical Intelligence, IJCAI-99*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 318–325.

[246] Henry Kautz and Joachim P. Walser. "State-Space Planning by Integer Optimization". In: *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI)*. AAAI Press, 1999, pp. 526–533.

[247] Henry A. Kautz, David A. McAllester, and Bart Selman. "Encoding Plans in Propositional Logic". In: *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996, pp. 374–384.

[248] Ahmet B. Keha, Ketan Khowala, and John W. Fowler. "Mixed Integer Programming Formulations for Single Machine Scheduling Problems". In: *Computers & Industrial Engineering* 56.1 (2009), pp. 357–367. DOI: 10.1016/j.cie.2008.06.008.

[249] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Berlin, Heidelberg: Springer, 2004. DOI: 10.1007/978-3-540-24777-7.

[250] Emil Keyder and Héctor Geffner. "Heuristics for Planning with Action Costs Revisited". In: *ECAI 2008 – 18th European Conference on Artificial Intelligence*. Vol. 178. Frontiers in Artificial Intelligence and Applications. NLD: IOS Press, 2008, pp. 588–592. DOI: 10.3233/978-1-58603-891-5-588.

[251] L.G. Khachiyan. "Polynomial Algorithms in Linear Programming". In: *USSR Computational Mathematics and Mathematical Physics* 20.1 (1980), pp. 53–72. DOI: 10.1016/0041-5553(80)90061-0.

[252] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. "Optimization by Simulated Annealing". In: *Science* 220.4598 (1983), pp. 671–680. DOI: 10.1126/science.220.4598.671.

[253] Akihiro Kishimoto, Alex Fukunaga, and Adi Botea. "Evaluation of a Simple, Scalable, Parallel Best-First Search Strategy". In: *Artificial Intelligence* 195 (2013), pp. 222–248. DOI: 10.1016/j.artint.2012.10.007.

[254] Yoshikazu Kobayashi, Akihiro Kishimoto, and Osamu Watanabe. "Evaluations of Hash Distributed A* in Optimal Sequence Alignment". In: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI-11*. Menlo Park, California: AAAI Press/International Joint Conferences on Artificial Intelligence Orginization, 2011, pp. 584–590. DOI: 10.5591/978-1-57735-516-8/IJCAI11-105.

[255] Walter H. Kohler and Kenneth Steiglitz. "Characterization and Theoretical Comparison of Branch-and-Bound Algorithms for Permutation Problems". In: *Journal of the ACM* 21.1 (1974), pp. 140–156. DOI: 10.1145/321796.321808.

[256] Andreas Kolling and Stefano Carpin. "The GRAPH-CLEAR Problem: Definition, Theoretical Properties and its Connections to Multirobot Aided Surveillance". In: *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*. 2007, pp. 1003–1008. DOI: 10.1109/IROS.2007.4399368.

[257] Wouter Kool, Herke van Hoof, Joaquim Gromicho, and Max Welling. "Deep Policy Dynamic Programming for Vehicle Routing Problems". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research – 19th International Conference, CPAIOR 2022*. Berlin, Heidelberg: Springer-Verlag, 2022, pp. 190–213. DOI: 10.1007/978-3-031-08011-1_14.

[258] Richard E Korf. "Depth-First Iterative-Deepening: An Optimal Admissible Tree Search". In: *Artificial Intelligence* 27.1 (1985), pp. 97–109. DOI: 10.1016/0004-3702(85)90084-0.

[259] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Sixth Edition. Berlin, Heidelberg: Springer, 2018. DOI: 10.1007/978-3-662-56039-6.

[260] Robert A. Kowalski. "Predicate Logic as Programming Language". In: *Information Processing, Proceedings of the Sixth IFIP Congress 1974*. North-Holland, 1974, pp. 569–574.

[261] Bjarni Kristjansson and Denise Lee. "The MPL Modeling System". In: *Modeling Languages in Mathematical Optimization*. Ed. by Josef Kallrath. Boston, MA: Springer, 2004, pp. 239–266. DOI: 10.1007/978-1-4613-0215-5_13.

[262] K. Kuchcinski and R. Szymanek. *JaCoP - Java Constraint Programming Solver*. Abstract from CP Solvers: Modeling, Applications, Integration, and Standardization, co-located with the 19th International Conference on Principles and Practice of Constraint Programming – CP 2013. 2013.

[263] Vipin Kumar and Laveen N. Kanal. "The CDP: A Unifying Formulation for Heuristic Search, Dynamic Programming, and Branch-and-Bound". In: *Search in Artificial Intelligence*. Ed. by Laveen Kanal and Vipin Kumar. New York, NY: Springer, 1988, pp. 1–27. DOI: 10.1007/978-1-4613-8788-6_1.

[264] Vipin Kumar, K. Ramesh, and V. Nageshwara Rao. "Parallel Best-First Search of State-Space Graphs: A Summary of Results". In: *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI)*. AAAI Press, 1988, pp. 122–127.

[265] Ryo Kuroiwa and J. Christopher Beck. "A Branch-and-Cut Approach for a Mixed Integer Linear Programming Compilation of Optimal Numeric Planning". In: *ICAPS 2021 Workshop on Heuristics and Search for Domain-independent Planning (HSDIP)*. 2021. URL: https://openreview.net/forum?id=vKLrYM4t_tB.

[266] Ryo Kuroiwa and J. Christopher Beck. *Domain-Independent Dynamic Programming*. 2024. arXiv: 2401.13883 [cs.AI].

[267] Ryo Kuroiwa and J. Christopher Beck. "Domain-Independent Dynamic Programming: Generic State Space Search for Combinatorial Optimization". In: *Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS)*. Palo Alto, California USA: AAAI Press, 2023, pp. 236–244. DOI: 10.1609/icaps.v33i1.27200.

[268] Ryo Kuroiwa and J. Christopher Beck. "Large Neighborhood Beam Search for Domain-Independent Dynamic Programming". In: *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*. Vol. 280. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 23:1–23:22. DOI: 10.4230/LIPIcs.CP.2023.23.

[269] Ryo Kuroiwa and J. Christopher Beck. "Parallel Beam Search Algorithms for Domain-Independent Dynamic Programming". In: *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI)*. Washington, DC, USA: AAAI Press, 2024, pp. 20743–20750. DOI: 10.1609/aaai.v38i18.30062.

[270] Ryo Kuroiwa and J. Christopher Beck. "Solving Domain-Independent Dynamic Programming Problems with Anytime Heuristic Search". In: *Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS)*. Palo Alto, California USA: AAAI Press, 2023, pp. 245–253. DOI: 10.1609/icaps.v33i1.27201.

[271] Ryo Kuroiwa and Alex Fukunaga. "Analyzing and Avoiding Pathological Behavior in Parallel Best-First Search". In: *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press, 2020, pp. 175–183. DOI: 10.1609/icaps.v30i1.6659.

[272] Ryo Kuroiwa and Alex Fukunaga. "On the Pathological Search Behavior of Distributed Greedy Best-First Search". In: *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS)*. 2019, pp. 255–263. DOI: 10.1609/icaps.v29i1.3485.

[273] Ryo Kuroiwa, Alexander Shleyfman, and J. Christopher Beck. "Extracting and Exploiting Bounds of Numeric Variables for Optimal Linear Numeric Planning". In: *ECAI 2023 – 26th European Conference on Artificial Intelligence*. Vol. 372. Frontiers in Artificial Intelligence and Applications. IOS Press, 2023, pp. 1332–1339. DOI: 10.3233/FAIA230409.

[274] Ryo Kuroiwa, Alexander Shleyfman, and J. Christopher Beck. "LM-Cut Heuristics for Optimal Linear Numeric Planning". In: *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS)*. Palo Alto, California USA: AAAI Press, 2022, pp. 203–212. DOI: 10.1609/icaps.v32i1.19803.

[275] Ryo Kuroiwa, Alexander Shleyfman, and J. Christopher Beck. *NLM-CutPlan*. 2023. URL: https://ipc2023-numeric.github.io/abstracts/NLM_CutPlan_Abstract.pdf.

[276] Ryo Kuroiwa, Alexander Shleyfman, Chiara Piacentini, Margarita P. Castro, and J. Christopher Beck. "The LM-Cut Heuristic Family for Optimal Numeric Planning with Simple Conditions". In: *Journal of Artificial Intelligence Research* 75 (2022), pp. 1477–1548. DOI: 10.1613/jair.1.14034.

[277] Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím. "IBM ILOG CP Optimizer for Scheduling". In: *Constraints* 23.2 (2018), pp. 210–250. DOI: 10.1007/s10601-018-9281-x.

[278] Ten-Hwang Lai and Sartaj Sahni. "Anomalies in Parallel Branch-and-Bound Algorithms". In: *Communications of the ACM* (1984), pp. 594–602. DOI: 10.1145/358080.358103.

[279] A. H. Land and A. G. Doig. "An Automatic Method of Solving Discrete Programming Problems". In: *Econometrica* 28.3 (1960), pp. 497–520. DOI: 10.2307/1910129.

[280] Jena-Lonis Lauriere. "A Language and a Program for Stating and Solving Combinatorial Problems". In: *Artificial Intelligence* 10.1 (1978), pp. 29–127. DOI: 10.1016/0004-3702(78)90029-2.

[281] Christophe Lecoutre. *ACE, A Generic Constraint Solver*. 2023. arXiv: 2302.05405 [cs.AI].

[282] Christophe Lecoutre and Nicolas Szczepanski. *PyCSP3: Modeling Combinatorial Constrained Problems in Python*. 2023. arXiv: 2009.00326 [cs.AI].

[283] C. Y. Lee. "Representation of Switching Circuits by Binary-Decision Programs". In: *The Bell System Technical Journal* 38.4 (1959), pp. 985–999. DOI: 10.1002/j.1538-7305.1959.tb01585.x.

[284] Daniel B. Leifker and Laveen N. Kanal. "A Hybrid SSS*/Alpha-Beta Algorithm for Parallel Search of Game Trees". In: *Proceedings of the Ninth International Joint Conference on Artificial Intelligence, IJCAI-85*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1985, pp. 1044–1046.

[285] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. "Complexity of Machine Scheduling Problems". In: Annals of Discrete Mathematics 1 (1977), pp. 343–362. DOI: 10.1016/S0167-5060(08)70743-X.

[286] Francesco Leofante, Enrico Giunchiglia, Erika Ábráham, and Armando Tacchella. "Optimal Planning Modulo Theories". In: *Proceedings of the 29th International Joint Conference on Artificial Intelligence, IJCAI-20*. Main track. International Joint Conferences on Artificial Intelligence Organization, 2020, pp. 4128–4134. DOI: 10.24963/ijcai.2020/571.

[287] Adam N. Letchford and Juan-José Salazar-González. "Stronger Multi-Commodity Flow Formulations of the (Capacitated) Sequential Ordering Problem". In: *European Journal of Operational Research* 251.1 (2016), pp. 74–84. DOI: 10.1016/j.ejor.2015.11.001.

[288] Art Lew and Holger Mauch. *Dynamic Programming: A Computational Tool*. Berlin, Heidelberg: Springer, 2006. DOI: 10.1007/978-3-540-37014-7.

[289] Dongxu Li, Enrico Scala, Patrik Haslum, and Sergiy Bogomolov. "Effect-Abstraction Based Relaxation for Linear Numeric Planning". In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence, IJCAI-18*. Main track. International Joint Conferences on Artificial Intelligence Organization, 2018, pp. 4787–4793. DOI: 10.24963/ijcai.2018/665.

[290] Jiaoyang Li, Zhe Chen, Daniel Harabor, Peter J. Stuckey, and Sven Koenig. "Anytime Multi-Agent Path Finding via Large Neighborhood Search". In: *Proceedings of the 30th International Joint Conference on Artificial Intelligence, IJCAI-21*. Main track. International Joint Conferences on Artificial Intelligence Organization, 2021, pp. 4127–4135. DOI: 10.24963/ijcai.2021/568.

[291] Luc Libralesso, Abdel-Malik Bouhassoun, Hadrien Cambazard, and Vincent Jost. "Tree Search for the Sequential Ordering Problem". In: *ECAI 2020 – 24th European Conference on Artificial Intelligence*. Vol. 325. Frontiers in Artificial Intelligence and Applications. IOS Press, 2020, pp. 459–465. DOI: 10.3233/FAIA200126.

[292] Luc Libralesso, Pablo Andres Focke, Aurélien Secardin, and Vincent Jost. "Iterative Beam Search Algorithms for the Permutation Flowshop". In: *European Journal of Operational Research* 301.1 (2022), pp. 217–234. DOI: 10.1016/j.ejor.2021.10.015.

[293] Shu Lin, Na Meng, and Wenxin Li. "Optimizing Constraint Solving via Dynamic Programming". In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI-19*. Main track. International Joint Conferences on Artificial Intelligence Organization, 2019, pp. 1146–1154. DOI: 10.24963/ijcai.2019/160.

[294] Alexandre Linhares and Horacio Hideki Yanasse. "Connections Between Cutting-Pattern Sequencing, VLSI Design, and Flexible Machines". In: *Computers & Operations Research* 29.12 (2002), pp. 1759–1772. DOI: 10.1016/S0305-0548(01)00054-5.

[295] Nir Lipovetzky and Hector Geffner. "Best-First Width Search: Exploration and Exploitation in Classical Planning". In: *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*. Palo Alto, California USA: AAAI Press, 2017. DOI: 10.1609/aaai.v31i1.11027.

[296] Yaxin Liu, Sven Koenig, and David Furcy. "Speeding Up the Calculation of Heuristics for Heuristic Search-Based Planning". In: *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*. AAAI Press, 2002, pp. 484–491.

[297] Adriana Lopez and Fahiem Bacchus. "Generalizing GraphPlan by Formulating Planning as a CSP". In: *Proceedings of the 18th International Joint Conference on Artificial Intelligence, IJCAI-03*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003, pp. 954–960.

[298] Leonardo Lozano, Daniel Duque, and Andrés L. Medaglia. "An Exact Algorithm for the Elementary Shortest Path Problem with Resource Constraints". In: *Transportation Science* 50.1 (2016), pp. 348–357. DOI: 10.1287/trsc.2014.0582.

[299] Leonardo Lozano and Andrés L. Medaglia. "On an Exact Method for the Constrained Shortest Path Problem". In: *Computers & Operations Research* 40.1 (2013), pp. 378–384. DOI: 10.1016/j.cor.2012.07.008.

[300] Leonardo Lozano and J. Cole Smith. "A Binary Decision Diagram Based Algorithm for Solving a Class of Binary Two-Stage Stochastic Programs". In: *Mathematical Programming* 191.1 (2022), pp. 381–404. DOI: 10.1007/s10107-018-1315-z.

[301] Miles Lubin, Oscar Dowson, Joaquim Dias Garcia, Joey Huchette, Benoît Legat, and Juan Pablo Vielma. "JuMP 1.0: Recent Improvements to a Modeling Language for Mathematical Optimization". In: *Mathematical Programming Computation* (2023). DOI: 10.1007/s12532-023-00239-3.

[302] Alan K. Mackworth. "Consistency in Networks of Relations". In: *Artificial Intelligence* 8.1 (1977), pp. 99–118. DOI: 10.1016/0004-3702(77)90007-8.

[303] N.R. Mahapatra and S. Dutt. "Scalable Duplicate Pruning Strategies for Parallel A* Graph Search". In: *Parallel and Distributed Processing, IEEE Symposium on*. Los Alamitos, CA, USA: IEEE Computer Society, 1993, pp. 290–297. DOI: 10.1109/SPDP.1993.395520.

[304] A. Martelli and U. Montanari. "From Dynamic Programming to Search Algorithms with Functional Costs". In: *Proceedings of the Fourth International Joint Conference on Artificial Intelligence, IJCAI-75*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1975, pp. 345–350.

[305] A. Martelli and U. Montanari. "On the Foundations of Dynamic Programming". In: *Topics in Combinatorial Optimization*. Ed. by Sergio Rinaldi. Vienna: Springer, 1975, pp. 145–163. DOI: 10.1007/978-3-7091-3291-3_9.

[306] Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. New York, NY, USA: John Wiley & Sons, Inc., 1990.

[307] Mateus Martin, Horacio Hideki Yanasse, and Maria José Pinto. "Mathematical Models for the Minimization of Open Stacks Problem". In: *International Transactions in Operational Research* 29.5 (2021), pp. 2944–2967. DOI: 10.1111/itor.13053.

[308] Drew M. McDermott. "The 1998 AI Planning Systems Competition". In: *AI Magazine* 21.2 (2000), pp. 35–55. DOI: 10.1609/aimag.v21i2.1506.

[309] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 4.1*. 2023. URL: https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report.pdf.

[310] Maged M. Michael. "High Performance Dynamic Lock-Free Hash Tables and List-Based Sets". In: *Proceedings of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*. New York, NY, USA: Association for Computing Machinery, 2002, pp. 73–82. DOI: 10.1145/564870.564881.

[311] L. Michel, P. Schaus, and P. Van Hentenryck. "MiniCP: a Lightweight Solver for Constraint Programming". In: *Mathematical Programming Computation* 13.1 (2021), pp. 133–184. DOI: 10.1007/s12532-020-00190-7.

[312] Donald Michie. ""Memo" Functions and Machine Learning". In: *Nature* 218.5136 (1968), pp. 19–22. DOI: 10.1038/218019a0.

[313] Eduardo Álvarez Miranda and Jordi Pereira. "On the Complexity of Assembly Line Balancing Problems". In: *Computers & Operations Research* 108 (2019), pp. 182–186. DOI: 10.1016/j.cor.2019.04.005.

[314] Federico Miretti, Daniela Misul, and Ezio Spessa. "DynaProg: Deterministic Dynamic Programming Solver for Finite Horizon Multi-Stage Decision Problems". In: *SoftwareX* 14 (2021), p. 100690. DOI: 10.1016/j.softx.2021.100690.

[315] Roberto Montemanni and Luca Maria Gambardella. "Ant Colony System for Team Orienteering Problems with Time Windows". In: *Foundations of Computing and Decision Sciences* 34 (2009), pp. 287–306.

[316] Jairo R. Montoya-Torres, Julián López Franco, Santiago Nieto Isaza, Heriberto Felizzola Jiménez, and Nilson Herazo-Padilla. "A Literature Review on the Vehicle Routing Problem with Multiple Depots". In: *Computers & Industrial Engineering* 79 (2015), pp. 115–129. DOI: 10.1016/j.cie.2014.10.029.

[317] Michael Morin, Margarita P. Castro, Kyle E.C. Booth, Tony T. Tran, Chang Liu, and J. Christopher Beck. "Intruder Alert! Optimization Models for Solving the Mobile Robot Graph-Clear Problem". In: *Constraints* 23.3 (2018), pp. 335–354. DOI: 10.1007/s10601-018-9288-3.

[318] Thomas L. Morin and Roy E. Marsten. "Branch-And-Bound Strategies for Dynamic Programming". In: *Operations Research* 24.4 (1976), pp. 611–627. DOI: 10.1287/opre.24.4.611.

[319] David R. Morrison, Edward C. Sewell, and Sheldon H. Jacobson. "An Application of the Branch, Bound, and Remember Algorithm to a New Simple Assembly Line Balancing Dataset". In: *European Journal of Operational Research* 236.2 (2014), pp. 403–409. DOI: 10.1016/j.ejor.2013.11.033.

[320] Shohin Mukherjee, Sandip Aine, and Maxim Likhachev. "ePA*SE: Edge-Based Parallel A* for Slow Evaluations". In: *Proceedings of the 17th International Symposium on Combinatorial Search (SoCS)*. AAAI Press, 2022, pp. 136–144. DOI: 10.1609/socs.v15i1.21761.

[321] Shohin Mukherjee, Sandip Aine, and Maxim Likhachev. "MPLP: Massively Parallelized Lazy Planning". In: *IEEE Robotics and Automation Letters* 7.3 (2022), pp. 6067–6074. DOI: 10.1109/LRA.2022.3157544.

[322] Shohin Mukherjee and Maxim Likhachev. "GePA*SE: Generalized Edge-Based Parallel A* for Slow Evaluations". In: *Proceedings of the 18th International Symposium on Combinatorial Search (SoCS)*. Washington, DC, USA: AAAI Press, 2023, pp. 153–157. DOI: 10.1609/socs.v16i1.27295.

[323] R.V. Nageshwara and V. Kumar. "Concurrent Access of Priority Queues". In: *IEEE Transactions on Computers* 37.12 (1988), pp. 1657–1665. DOI: 10.1109/12.9744.

[324] Hootan Nakhost and Martin Müller. "Action Elimination and Plan Neighborhood Graph Search: Two Algorithms for Plan Improvement". In: *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press, 2010, pp. 121–128. DOI: 10.1609/icaps.v20i1.13402.

[325] Dana S. Nau, Vipin Kumar, and Laveen Kanal. "General Branch and Bound, and its Relation to A* and AO*". In: *Artificial Intelligence* 23.1 (1984), pp. 29–58. DOI: 10.1016/0004-3702(84)90004-3.

[326] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. "MiniZinc: Towards a Standard CP Modelling Language". In: *Principles and Practice of Constraint Programming – CP 2007*. Berlin, Heidelberg: Springer, 2007, pp. 529–543.

[327] Peter Norvig. "Techniques for Automatic Memoization with Applications to Context-Free Parsing". In: *Computational Linguistics* 17.1 (1991), pp. 91–98. URL: https://aclanthology.org/J91-1004.

[328] Jeffrey W. Ohlmann and Barrett W. Thomas. "A Compressed-Annealing Heuristic for the Traveling Salesman Problem with Time Windows". In: *INFORMS Journal on Computing* 19.1 (2007), pp. 80–90. DOI: 10.1287/ijoc.1050.0145.

[329] Laurent Orseau and Levi H. S. Lelis. "Policy-Guided Heuristic Search with Guarantees". In: *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI)*. Palo Alto, California USA: AAAI Press, 2021, pp. 12382–12390. DOI: 10.1609/aaai.v35i14.17469.

[330] OscaR Team. *OscaR: Scala in OR*. Available from https://bitbucket.org/oscarlib/oscar. 2012.

[331] Peter S. Pacheco and Matthew Malensek. *An Introduction to Parallel Programming*. Second Edition. Philadelphia: Morgan Kaufmann, 2022. DOI: 10.1016/B978-0-12-804605-0.00010-5.

[332] Dario Pacino and Pascal Van Hentenryck. "Large Neighborhood Search and Adaptive Randomized Decompositions for Flexible Jobshop Scheduling". In: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI-11*. Menlo Park, California: AAAI Press/Intertnational Joint Conferences on Artificial Intelligence Organization, 2011, pp. 1997–2002. DOI: 10.5591/978-1-57735-516-8/IJCAI11-333.

[333] Stefan Panjkovic and Andrea Micheli. "Abstract Action Scheduling for Optimal Temporal Planning via OMT". In: *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI)*. Washington, DC, USA: AAAI Press, 2024, pp. 20222–20229. DOI: 10.1609/aaai.v38i18.30002.

[334] Christos H. Papadimitriou. "The Euclidean Travelling Salesman Problem is NP-Complete". In: *Theoretical Computer Science* 4.3 (1977), pp. 237–244. DOI: 10.1016/0304-3975(77)90012-3.

[335] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. USA: Addison-Wesley Longman Publishing Co., Inc., 1984. DOI: 10.5555/525.

[336] Judea Pearl and Jin H. Kim. "Studies in Semi-Admissible Heuristics". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-4.4 (1982), pp. 392–399. DOI: 10.1109/TPAMI.1982.4767270.

[337] Laurent Perron, Frédéric Didier, and Steven Gay. "The CP-SAT-LP Solver". In: *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 3:1–3:2. DOI: 10.4230/LIPIcs.CP.2023.3.

[338] Thomy Phan, Taoan Huang, Bistra Dilkina, and Sven Koenig. "Adaptive Anytime Multi-Agent Path Finding Using Bandit-Based Large Neighborhood Search". In: *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI)*. Washington, DC, USA: AAAI Press, 2024, pp. 17514–17522. DOI: 10.1609/aaai.v38i16.29701.

[339] Mike Phillips, Maxim Likhachev, and Sven Koenig. "PA*SE: Parallel A* for Slow Expansions". In: *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press, 2014, pp. 208–216. DOI: 10.1609/icaps.v24i1.13652.

[340] Chiara Piacentini, Margarita Castro, Andre Cire, and J. Christopher Beck. "Compiling Optimal Numeric Planning to Mixed Integer Linear Programming". In: *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press, 2018, pp. 383–387. DOI: 10.1609/icaps.v28i1.13919.

[341] Chiara Piacentini, Margarita P. Castro, Andre A. Cire, and J. Christopher Beck. "Linear and Integer Programming-Based Heuristics for Cost-Optimal Numeric Planning". In: *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*. Palo Alto, California USA: AAAI Press, 2018, pp. 6254–6261. DOI: 10.1609/aaai.v32i1.12082.

[342] Michael L. Pinedo. *Planning and Scheduling in Manufacturing and Services*. Second Edition. New York, NY: Springer, 2009. DOI: 10.1007/978-1-4419-0910-7.

[343] David Pisinger and Stefan Ropke. "Large Neighborhood Search". In: *Handbook of Metaheuristics*. Ed. by Michel Gendreau and Jean-Yves Potvin. Cham: Springer International Publishing, 2019, pp. 99–127. DOI: 10.1007/978-3-319-91086-4_4.

[344] Florian Pommerening, Gabriele Röger, Malte Helmert, and Blai Bonet. "LP-Based Heuristics for Cost-Optimal Planning". In: *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press, 2014, pp. 226–234. DOI: 10.1609/icaps.v24i1.13621.

[345] Patrick Prosser. "An Empirical Study of Phase Transitions in Binary Constraint Satisfaction Problems". In: *Artificial Intelligence* 81.1 (1996). Frontiers in Problem Solving: Phase Transitions and Complexity, pp. 81–109. DOI: 10.1016/0004-3702(95)00048-8.

[346] Charles Prud'homme and Jean-Guillaume Fages. "Choco-solver: A Java Library for Constraint Programming". In: *Journal of Open Source Software* 7.78 (2022), p. 4708. DOI: 10.21105/joss.04708.

[347] Charles Prud'homme, Xavier Lorca, and Narendra Jussien. "Explanation-Based Large Neighborhood Search". In: *Constraints* 19.4 (2014), pp. 339–379. DOI: 10.1007/s10601-014-9166-6.

[348] Jakob Puchinger and Peter J. Stuckey. "Automating Branch-and-Bound for Dynamic Programs". In: *Proceedings of the 2008 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*. PEPM '08. New York, NY, USA: Association for Computing Machinery, 2008, 81–89. DOI: 10.1145/1328408.1328421.

[349] Hu Qin, Zizhen Zhang, Andrew Lim, and Xiaocong Liang. "An Enhanced Branch-and-Bound Algorithm for the Talent Scheduling Problem". In: *European Journal of Operational Research* 250.2 (2016), pp. 412–426. DOI: 10.1016/j.ejor.2015.10.002.

[350] Bochra Rabbouch, Foued Saâdaoui, and Rafaa Mraihi. "Constraint Programming Based Algorithm for Solving Large-Scale Vehicle Routing Problems". In: *Hybrid Artificial Intelligent Systems*. Cham: Springer International Publishing, 2019, pp. 526–539. DOI: 10.1007/978-3-030-29859-3_45.

[351] Miquel Ramirez, Nir Lipovetzky, and Christian Muise. *Lightweight Automated Planning ToolKiT*. http://lapkt.org/. tccessed: 2020. 2015.

[352] Daniel Ratner and Ira Pohl. "Joint and LPA*: Combination of Approximation and Search". In: *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI)*. AAAI Press, 1986, pp. 173–177.

[353] A. Reinefeld and T.A. Marsland. "Enhanced Iterative-Deepening Search". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16.7 (1994), pp. 701–710. DOI: 10.1109/34.297950.

[354] Silvia Richter and Matthias Westphal. "The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks". In: *Journal of Artificial Intelligence Research* 39 (2010), pp. 127–177. DOI: 10.1613/jair.2972.

[355] Giovanni Righini and Matteo Salani. "Decremental State Space Relaxation Strategies and Initialization Heuristics for Solving the Orienteering Problem with Time Windows with Dynamic Programming". In: *Computers & Operations Research* 36.4 (2009), pp. 1191–1203. DOI: 10.1016/j.cor.2008.01.003.

[356] Giovanni Righini and Matteo Salani. *Dynamic Programming for the Orienteering Problem with Time Windows.* Tech. rep. 91. Crema, Italy: Dipartimento di Tecnologie dell'Informazione, Universita degli Studi Milano, 2006.

[357] Giovanni Righini and Matteo Salani. "New Dynamic Programming Algorithms for the Resource Constrained Elementary Shortest Path Problem". In: *Networks* 51.3 (2008), pp. 155–170. DOI: 10.1002/net.20212.

[358] Giovanni Righini and Matteo Salani. "Symmetry Helps: Bounded Bi-Directional Dynamic Programming for the Elementary Shortest Path Problem with Resource Constraints". In: *Discrete Optimization* 3.3 (2006). Graphs and Combinatorial Optimization, pp. 255–273. DOI: 10.1016/j.disopt.2006.05.007.

[359] Jussi Rintanen. "Phase Transitions in Classical Planning: an Experimental Study". In: *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS).* AAAI Press, 2004, pp. 101–110.

[360] Jussi Rintanen. "Planning as Satisfiability: Heuristics". In: *Artificial Intelligence* 193 (2012), pp. 45–86. DOI: 10.1016/j.artint.2012.08.001.

[361] Jussi Rintanen. "Temporal Planning with Clock-Based SMT Encodings". In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence, IJCAI-17.* Main track. International Joint Conferences on Artificial Intelligence Orginization, 2017, pp. 743–749. DOI: 10.24963/ijcai.2017/103.

[362] Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä. "Planning as Satisfiability: Parallel Plans and Algorithms for Plan Search". In: *Artificial Intelligence* 170.12 (2006), pp. 1031–1080. DOI: 10.1016/j.artint.2006.08.002.

[363] Marcus Ritt and Alysson M. Costa. "Improved Integer Programming Models for Simple Assembly Line Balancing and Related Problems". In: *International Transactions in Operational Research* 25.4 (2018), pp. 1345–1359. DOI: 10.1111/itor.12206.

[364] Herbert Robbins. "Some Aspects of the Sequential Design of Experiments". In: *Bulletin of the American Mathematical Society* 58 (1952).

[365] Roberto Roberti and Aristide Mingozzi. "Dynamic ng-Path Relaxation for the Delivery Man Problem". In: *Transportation Science* 48.3 (2014), pp. 413–424. DOI: 10.1287/trsc.2013.0474.

[366] Nathan Robinson, Charles Gretton, Duc-Nghia Pham, and Abdul Sattar. "A Compact and Efficient SAT Encoding for Planning". In: *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS).* AAAI Press, 2008, pp. 296–303.

[367] J.W. Romein, H.E. Bal, J. Schaeffer, and A. Plaat. "A Performance Analysis of Transposition-Table-Driven Work Scheduling in Distributed Search". In: *IEEE Transactions on Parallel and Distributed Systems* 13.5 (2002), pp. 447–459. DOI: 10.1109/TPDS.2002.1003855.

[368] Stefan Ropke and David Pisinger. "An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows". In: *Transportation Science* 40.4 (2006), pp. 455–472. DOI: 10.1287/trsc.1050.0135.

[369] Francesca Rossi, Peter van Beek, and Toby Walsh, eds. *Handbook of Constraint Programming.* Vol. 2. Foundations of Artificial Intelligence. Elsevier, 2006.

[370] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Fourth Edition. Pearson, 2020. URL: http://aima.cs.berkeley.edu/.

[371] Stuart Russell and Peter Norvig. "Making Complx Decisions". In: *Artificial Intelligence: A Modern Approach*. Fourth Edition. Pearson, 2020. Chap. 17, pp. 562–598. URL: http://aima.cs.berkeley.edu/.

[372] Stuart Russell and Peter Norvig. "Solving Problems by Searching". In: *Artificial Intelligence: A Modern Approach*. Fourth Edition. Pearson, 2020. Chap. 3, pp. 63–109. URL: http://aima.cs.berkeley.edu/.

[373] Ruslan Sadykov, Eduardo Uchoa, and Artur Pessoa. "A Bucket Graph-Based Labeling Algorithm with Application to Vehicle Routing". In: *Transportation Science* 55.1 (2021), pp. 4–28. DOI: 10.1287/trsc.2020.0985.

[374] M. E. Salveson. "The Assembly-Line Balancing Problem". In: *The Journal of Industrial Engineering* 6.3 (1955), pp. 18–25. DOI: 10.1115/1.4014559.

[375] Scott Sanner. "Relational Dynamic Influence Diagram Language (RDDL): Language Description". 2010. URL: http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf.

[376] Scott Sanner and Craig Boutilier. "Approximate Linear Programming for First-Order MDPs". In: *UAI 05, Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence*. Arlington, Virginia, USA: AUAI Press, 2005, pp. 509–517.

[377] M. W. P. Savelsbergh. "Local Search in Routing Problems with Time Windows". In: *Annals of Operations Research* 4.1 (1985), pp. 285–305. DOI: 10.1007/BF02022044.

[378] Enrico Scala, Patrik Haslum, Daniele Magazzeni, and Sylvie Thiébaux. "Landmarks for Numeric Planning Problems". In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence, IJCAI-17*. Main track. International Joint Conferences on Artificial Intelligence Organization, 2017, pp. 4384–4390. DOI: 10.24963/ijcai.2017/612.

[379] Enrico Scala, Patrik Haslum, Sylvie Thiébaux, and Miquel Ramírez. "Interval-Based Relaxation for General Numeric Planning". In: *ECAI 2016 – 22nd European Conference on Artificial Intelligence*. Vol. 285. Frontiers in Artificial Intelligence and Applications. IOS Press, 2016, pp. 655–663. DOI: 10.3233/978-1-61499-672-9-655.

[380] Enrico Scala, Patrik Haslum, Sylvie Thiébaux, and Miquel Ramírez. "Subgoaling Techniques for Satisficing and Optimal Numeric Planning". In: *Journal of Artificial Intellingence Research* 68 (2020), pp. 691–752. DOI: 10.1613/jair.1.11875.

[381] Enrico Scala, Miquel Ramírez, Patrik Haslum, and Sylvie Thiebaux. "Numeric Planning with Disjunctive Global Constraints via SMT". In: *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press, 2016, pp. 276–0284. DOI: 10.1609/icaps.v26i1.13766.

[382] J. E. Schoenfield. *Fast, Exact Solution of Open Bin Packing Problems Without Linear Programming*. Tech. rep. Hutsville, Alabama, USA: US Army Space and Missile Defense Command, 2002.

[383] Armin Scholl and Robert Klein. "SALOME: A Bidirectional Branch-and-Bound Procedure for Assembly Line Balancing". In: *INFORMS Journal on Computing* 9.4 (1997), pp. 319–335. DOI: 10.1287/ijoc.9.4.319.

[384] Armin Scholl, Robert Klein, and Christian Jürgens. "Bison: A Fast Hybrid Procedure for Exactly Solving the One-Dimensional Bin Packing Problem". In: *Computers & Operations Research* 24.7 (1997), pp. 627–645. DOI: 10.1016/S0305-0548(96)00082-2.

[385] Petra Schwerin and Gerhard Wäscher. "The Bin-Packing Problem: A Problem Generator and Some Numerical Experiments with FFD Packing and MTP". In: *International Transactions in Operational Research* 4.5 (1997), pp. 377–389. DOI: 10.1016/S0969-6016(97)00025-7.

[386] Jendrik Seipp and Malte Helmert. "Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning". In: *Journal of Artificial Intelligence Research* 62 (2018), pp. 535–577. DOI: 10.1613/jair.1.11217.

[387] Bart Selman, David G. Mitchell, and Hector J. Levesque. "Generating Hard Satisfiability Problems". In: *Artificial Intelligence* 81.1 (1996). Frontiers in Problem Solving: Phase Transitions and Complexity, pp. 17–29. DOI: 10.1016/0004-3702(95)00045-3.

[388] E. C. Sewell and S. H. Jacobson. "A Branch, Bound, and Remember Algorithm for the Simple Assembly Line Balancing Problem". In: *INFORMS Journal on Computing* 24.3 (2012), pp. 433–442. DOI: 10.1287/ijoc.1110.0462.

[389] Lei Shang, Vincent T'Kindt, and Federico Della Croce. "Branch & Memorize Exact Algorithms for Sequencing Problems: Efficient Embedding of Memorization into Search Trees". In: *Computers & Operations Research* 128 (2021), p. 105171. DOI: 10.1016/j.cor.2020.105171.

[390] C. E. Shannon. "A Mathematical Theory of Communication". In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[391] Paul Shaw. "A Constraint for Bin Packing". In: *Principles and Practice of Constraint Programming – CP 2004*. Berlin, Heidelberg: Springer, 2004, pp. 648–662. DOI: 10.1007/978-3-540-30201-8_47.

[392] Paul Shaw. "Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems". In: *Principles and Practice of Constraint Programming – CP98*. Berlin, Heidelberg: Springer, 1998, pp. 417–431. DOI: 10.1007/3-540-49481-2_30.

[393] William Shen, Felipe Trevizan, and Sylvie Thiébaux. "Learning Domain-Independent Planning Heuristics with Hypergraph Networks". In: *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press, 2020, pp. 574–584. DOI: 10.1609/icaps.v30i1.6754.

[394] Alfonso Shimbel. "Structural Parameters of Communication Networks". In: *The Bulletin of Mathematical Biophysics* 15.4 (1953), pp. 501–507. DOI: 10.1007/BF02476438.

[395] Takumi Shimoda and Alex Fukunaga. "Improved Exploration of the Bench Transition System in Parallel Greedy Best First Search". In: *Proceedings of the 16th International Symposium on Combinatorial Search (SoCS)*. Washington, DC, USA: AAAI Press, 2023, pp. 100–107. DOI: 10.1609/socs.v16i1.27285.

[396] Yuji Shinano, Tobias Achterberg, Timo Berthold, Stefan Heinz, and Thorsten Koch. "ParaSCIP: A Parallel Extension of SCIP". In: *Competence in High Performance Computing 2010*. Berlin, Heidelberg: Springer, 2012, pp. 135–148.

[397] Yuji Shinano, Tobias Achterberg, Timo Berthold, Stefan Heinz, Thorsten Koch, and Michael Winkler. "Solving Open MIP Instances with ParaSCIP on Supercomputers Using up to 80,000 Cores". In: *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2016, pp. 770–779. DOI: 10.1109/IPDPS.2016.56.

[398] Yuji Shinano, Timo Berthold, and Stefan Heinz. "ParaXpress: an Experimental Extension of the FICO Xpress-Optimizer to Solve Hard MIPs on Supercomputers". In: *Optimization Methods and Software* 33.3 (2018), pp. 530–539. DOI: 10.1080/10556788.2018.1428602.

[399] Alexander Shleyfman, Daniel Gnad, and Peter Jonsson. "Structurally Restricted Fragments of Numeric Planning – a Complexity Analysis". In: *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI)*. Washington, DC, USA: AAAI Press, 2023, pp. 12112–12119. DOI: 10.1609/aaai.v37i10.26428.

[400] Alexander Shleyfman, Ryo Kuroiwa, and J. Christopher Beck. "Symmetry Detection and Breaking in Linear Cost-Optimal Numeric Planning". In: *Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS)*. Palo Alto, California USA: AAAI Press, 2023, pp. 393–401. DOI: 10.1609/icaps.v33i1.27218.

[401] David Silver. "Cooperative Pathfinding". In: *Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*. AAAI Press, 2021, pp. 117–122. DOI: 10.1609/aiide.v1i1.18726.

[402] Barbara Smith and Ian Gent. *Constraint Modelling Challenge Report 2005*. https://ipg.host.cs.st-andrews.ac.uk/challenge/. 2005.

[403] Barbara M. Smith. "Caching Search States in Permutation Problems". In: *Principles and Practice of Constraint Programming – CP 2005*. Berlin, Heidelberg: Springer, 2005, pp. 637–651. DOI: 10.1007/11564751_47.

[404] Barbara M. Smith and Martin E. Dyer. "Locating the Phase Transition in Binary Constraint Satisfaction Problems". In: *Artificial Intelligence* 81.1 (1996). Frontiers in Problem Solving: Phase Transitions and Complexity, pp. 155–181. DOI: 10.1016/0004-3702(95)00052-6.

[405] Marius M. Solomon. "Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints". In: *Operations Research* 35.2 (1987), pp. 254–265. DOI: 10.1287/opre.35.2.254.

[406] A. Srinivasan, T. Ham, S. Malik, and R.K. Brayton. "Algorithms for Discrete Function Manipulation". In: *1990 IEEE International Conference on Computer-Aided Design. Digest of Technical Papers*. 1990, pp. 92–95. DOI: 10.1109/ICCAD.1990.129849.

[407] Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Roman Barták, and Eli Boyarski. "Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks". In: *Proceedings of the 12th International Symposium on Combinatorial Search (SoCS)*. AAAI Press, 2019, pp. 151–158. DOI: 10.1609/socs.v10i1.18510.

[408] Alex Stivala, Peter J. Stuckey, Maria Garcia de la Banda, Manuel Hermenegildo, and Anthony Wirth. "Lock-Free Parallel Dynamic Programming". In: *Journal of Parallel and Distributed Computing* 70.8 (2010), pp. 839–848. DOI: 10.1016/j.jpdc.2010.01.004.

[409] Damian Sulewski, Stefan Edelkamp, and Peter Kissmann. "Exploiting the Computational Power of the Graphics Card: Optimal State Space Planning on the GPU". In: *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press, 2011, pp. 242–249. DOI: 10.1609/icaps.v21i1.13464.

[410] H. Sundell and P. Tsigas. "Fast and Lock-Free Concurrent Priority Queues for Multi-Thread Systems". In: *Proceedings International Parallel and Distributed Processing Symposium*. 2003, pp. 609–627. DOI: 10.1109/ipdps.2003.1213189.

[411] Olle Sundstrom and Lino Guzzella. "A Generic Dynamic Programming Matlab Function". In: *2009 IEEE Control Applications, (CCA) & Intelligent Control, (ISIC)*. 2009, pp. 1625–1630. DOI: 10.1109/CCA.2009.5281131.

[412] Richard S. Sutton and Andrew G. Barto. "Dynamic Programming". In: *Reinforcement Learning: An Introduction*. Second Edition. Cambridge, MA, USA: A Bradford Book, 2018. Chap. 4.

[413] Richard S. Sutton and Andrew G. Barto. "Finite Markov Decision Process". In: *Reinforcement Learning: An Introduction*. Second Edition. Cambridge, MA, USA: A Bradford Book, 2018. Chap. 3.

[414] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second Edition. Cambridge, MA, USA: A Bradford Book, 2018.

[415] Hisao Tamaki and Taisuke Sato. "OLD Resolution with Tabulation". In: *Third International Conference on Logic Programming (ICLP)*. Berlin, Heidelberg: Springer, 1986, pp. 84–98.

[416] Ole Tange. "GNU Parallel - The Command-Line Power Tool". In: *;login: The USENIX Magazine* 36 (2011), pp. 42–47.

[417] Jordan T. Thayer and Wheeler Ruml. "Bounded Suboptimal Search: A Direct Approach Using Inadmissible Estimates". In: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI-11*. Menlo Park, California: AAAI Press/International Joint Conferences on Artificial Intelligence Orginization, 2011, pp. 674–679. DOI: 10.5591/978-1-57735-516-8/IJCAI11-119.

[418] Álvaro Torralba and Jörg Hoffmann. "Simulation-Based Admissible Dominance Pruning". In: *Proceedings of the 24th International Joint Conference on Artificial Intelligence, IJCAI-15*. Palo Alto, California USA: AAAI Press/International Joint Conferences on Artificial Intelligence Organization, 2015, pp. 1689–1695.

[419] Álvaro Torralba and Florian Pommerening. *Planner Abstracts for the Classical Tracks in the International Planning Competition 2018*. 2018. URL: https://ipc2018-classical.bitbucket.io/planner-abstracts/ipc-2018-planner-abstracts-classical-tracks.pdf.

[420] Alejandro Torreño, Eva Onaindia, Antonín Komenda, and Michal Štolba. "Cooperative Multi-Agent Planning: A Survey". In: *ACM Computing Surveys* 50.6 (2017). DOI: 10.1145/3128584.

[421] Paolo Toth and Daniele Vigo. "Models, Relaxations and Exact Approaches for the Capacitated Vehicle Routing Problem". In: *Discrete Applied Mathematics* 123.1 (2002), pp. 487–512. DOI: 10.1016/S0166-218X(01)00351-1.

[422] Paolo Toth and Daniele Vigo. *Vehicle Routing: Problems, Methods, and Applications*. Second Edition. Society for Industrial and Applied Mathematics, 2014. DOI: 10.1137/1.9781611973594.

[423] Long Tran-Thanh, Archie Chapman, Enrique Munoz de Cote, Alex Rogers, and Nicholas R. Jennings. "Epsilon-First Policies for Budget-Limited Multi-Armed Bandits". In: *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI)*. Palo Alto, California USA: AAAI Press, 2010, pp. 1211–1216. DOI: 10.1609/aaai.v24i1.7758.

[424] Michael A. Trick. "A Dynamic Programming Approach for Consistency and Propagation for Knapsack Constraints". In: *Annals of Operations Research* 118.1 (2003), pp. 73–84. DOI: 10.1023/A:1021801522545.

[425] Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. "New Benchmark Instances for the Capacitated Vehicle Routing Problem". In: *European Journal of Operational Research* 257.3 (2017), pp. 845–858. DOI: 10.1016/j.ejor.2016.08.012.

[426] Satya Gautam Vadlamudi, Sandip Aine, and Partha Pratim Chakrabarti. "Anytime Pack Search". In: *Natural Computing* 15.3 (2016), pp. 395–414. DOI: 10.1007/978-3-642-45062-4_88.

[427] Satya Gautam Vadlamudi, Piyush Gaurav, Sandip Aine, and Partha Pratim Chakrabarti. "Anytime Column Search". In: *AI 2012: Advances in Artificial Intelligence*. Berlin, Heidelberg: Springer, 2012, pp. 254–265. DOI: 10.1007/978-3-642-35101-3_22.

[428] Mauro Vallati, Lukáš Chrpa, and Thomas L. McCluskey. *The 2014 International Planning Competition – Description of Participant Planners of the Deterministic Track*. 2014. URL: https://helios.hud.ac.uk/scommv/IPC-14/repository/booklet2014.pdf.

[429] Peter van Beek. "Backtracking Search Algorithms". In: *Handbook of Constraint Programming*. Ed. by Francesca Rossi, Peter van Beek, and Toby Walsh. Vol. 2. Foundations of Artificial Intelligence. Elsevier, 2006. Chap. 4, pp. 85–134. DOI: 10.1016/S1574-6526(06)80008-8.

[430] Menkes van den Briel, Thomas Vossen, and Subbarao Kambhampati. "Reviving Integer Programming Approaches for AI Planning: A Branch-and-Cut Framework". In: *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press, 2005, pp. 310–319.

[431] Pascal Van Hentenryck and Laurent Michel. *Constraint-Based Local Search*. Cambridge: MIT Press, 2005.

[432] Willem-Jan van Hoeve and Irit Katriel. "Global Constraints". In: *Handbook of Constraint Programming*. Ed. by Francesca Rossi, Peter van Beek, and Toby Walsh. Vol. 2. Foundations of Artificial Intelligence. Elsevier, 2006. Chap. 6, pp. 169–208. DOI: 10.1016/S1574-6526(06)80010-6.

[433] Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. "The Orienteering Problem: A Survey". In: *European Journal of Operational Research* 209.1 (2011), pp. 1–10. DOI: 10.1016/j.ejor.2010.03.045.

[434] Pieter Vansteenwegen, Wouter Souffriau, Greet Vanden Berghe, and Dirk Van Oudheusden. "Iterated Local Search for the Team Orienteering Problem with Time Windows". In: *Computers & Operations Research* 36.12 (2009). New Developments on Hub Location, pp. 3281–3290. DOI: 10.1016/j.cor.2009.03.008.

[435] Vincent Vidal and Héctor Geffner. "Branching and Pruning: An Optimal Temporal POCL Planner Based on Constraint Programming". In: *Artificial Intelligence* 170.3 (2006), pp. 298–335. DOI: 10.1016/j.artint.2005.08.004.

[436] Vincent Vidal, Bordeaux Lucas, and Youssef Hamadi. "Adaptive K-Parallel Best-First Search: A Simple but Efficient Algorithm for Multi-Core Domain-Independent Planning". In: *Proceedings of the Third International Symposium on Combinatorial Search (SOCS)*. AAAI Press, 2010. DOI: 10.1609/socs.v1i1.18165.

[437] Tim Vieira, Matthew Francis-Landau, Nathaniel Wesley Filardo, Farzad Khorasani, and Jason Eisner. "Dyna: Toward a Self-Optimizing Declarative Language for Machine Learning Applications". In: *Proceedings of the First ACM SIGPLAN Workshop on Machine Learning and Programming Languages (MAPL)*. ACM, 2017, pp. 8–17. DOI: 10.1145/3088525.3088562.

[438] Thomas Vossen, Michael Ball, Amnon Lotem, and Dana Nau. "On the Use of Integer Programming Models in AI Planning". In: *Proceedings of the 16th International Joint Conference on Artifical Intelligence, IJCAI-99*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 304–309.

[439] Laurence Wolsey. "Cutting Plane Algorithms". In: *Integer Programming*. John Wiley & Sons, Ltd, 2020. Chap. 8, pp. 139–166. DOI: 10.1002/9781119606475.ch8.

[440] Laurence Wolsey. *Integer Programming*. Second Edition. John Wiley & Sons, Inc., 2020. DOI: 10.1002/9781119606475.

[441] Gerhard Wäscher and Thomas Gau. "Heuristics for the Integer One-Dimensional Cutting Stock Problem: A Computational Study". In: *Operations-Research-Spektrum* 18.3 (1996), pp. 131–144. DOI: 10.1007/BF01539705.

[442] Yingce Xia, Xu-Dong Zhang, Nenghai Yu, Geoffrey Holmes, and Yan Liu. "Budgeted Bandit Problems with Continuous Random Costs". In: *Proceedings of the Seventh Asian Conference on Machine Learning (ACML)*. 2015, pp. 317–332.

[443] Hanlan Yang, Shohin Mukherjee, and Maxim Likhachev. "A-ePA*SE: Anytime Edge-Based Parallel A* for Slow Evaluations". In: *Proceedings of the 18th International Symposium on Combinatorial Search (SoCS)*. Washington, DC, USA: AAAI Press, 2023, pp. 163–167. DOI: 10.1609/socs.v16i1.27297.

[444] Håkan L. S. Younes and Michael L. Littman. *PPDDL1.0: An Extension to PDDL for Expressing Planning Domains with Probabilistic Effects*. Tech. rep. CMU-CS-04-167. 2004.

[445] Liu Yu, Ryo Kuroiwa, and Fukunaga Alex. "Learning Search-Space Specific Heuristics Using Neural Network". In: *Proceedings of the 12th Workshop on Heuristics and Search for Domain-Independent Planning (HSDIP)*. 2020, pp. 1–8.

[446] Boon J Yuen and Ken V Richardson. "Establishing the Optimality of Sequencing Heuristics for Cutting Stock Problems". In: *European Journal of Operational Research* 84.3 (1995), pp. 590–598. DOI: 10.1016/0377-2217(95)00025-L.

[447] Weixiong Zhang. "Complete Anytime Beam Search". In: *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI)*. AAAI Press, 1998, pp. 425–430.

[448] Yang Zhang and Eric A. Hansen. "Parallel Breadth-First Heuristic Search on a Shared-Memory Architecture". In: *Heuristic Search, Memory-Based Heuristics and Their Applications: Papers from the AAAI Workshop*. Menlo Park, California: AAAI Press, 2006, pp. 33–38.

[449] Neng-Fa Zhou, Håkan Kjellerstrand, and Jonathan Fruhman. *Constraint Solving and Planning with Picat*. Cham: Springer, 2015. DOI: 10.1007/978-3-319-25883-6.

[450] Rong Zhou and Eric A. Hansen. "Breadth-First Heuristic Search". In: *Artificial Intelligence* 170.4 (2006), pp. 385–408. DOI: 10.1016/j.artint.2005.12.002.

[451] Rong Zhou and Eric A. Hansen. "Parallel Structured Duplicate Detection". In: *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Press, 2007, pp. 1217–1223.

[452] Yichao Zhou and Jianyang Zeng. "Massively Parallel A* Search on a GPU". In: *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Press, 2015, pp. 1248–1254.

[453] Albert L. Zobrist. *A New Hashing Method With Application for Game Playing*. Tech. rep. Department of Computer Sciences, University of Wisconsin-Madison, 1970. URL: http://digital.library.wisc.edu/1793/57624.

[454] Christian Höner zu Siederdissen, Sonja J. Prohaska, and Peter F. Stadler. "Algebraic Dynamic Programming over General Data Structures". In: *BMC Bioinformatics* 16.19 (2015). DOI: 10.1186/1471-2105-16-S19-S2.