Solving Domain-Independent Dynamic Programming Problems with Anytime Heuristic Search

Ryo Kuroiwa, J. Christopher Beck

Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Canada, ON M5S 3G8
ryo.kuroiwa@mail.utoronto.ca, jcb@mie.utoronto.ca

Abstract

Domain-independent dynamic programming (DIDP) is a recently proposed model-based paradigm for combinatorial optimization where a problem is formulated as dynamic programming (DP) and solved by a generic solver. In this paper, we develop anytime heuristic search solvers for DIDP, which quickly find a feasible solution and continuously improve it to prove optimality. We implement six anytime heuristic search algorithms previously used as problem-specific methods and evaluate them on nine different problem classes. Our experimental results show that most of the anytime DIDP solvers outperform an existing A*-based solver, mixed-integer programming, and constraint programming in proving optimality, solution quality, and primal integral across multiple problem classes. In particular, complete anytime beam search (CABS) performs the best, improving on the best-known solution for one instance of traveling salesperson problem with time windows and closing five instances of one-to-one multi-commodity pick-and-delivery traveling salesperson problems.

Introduction

Many challenges in planning and scheduling arise from combinatorial optimization, which requires making discrete decisions to optimize an objective function. In model-based approaches for combinatorial optimization, a problem is formulated as a mathematical model and solved by a generic solver. Such approaches are seen as the Holy Grail of computer programming, and, indeed, AI planning, where a user just needs to state the problem to solve it (Freuder 1997).

While mixed-integer programming (MIP) and constraint programming (CP) are widely used model-based approaches for combinatorial optimization, domain-independent dynamic programming (DIDP) is a recently proposed model-based paradigm based on dynamic programming (DP) (Kuroiwa and Beck 2023b). DIDP is also related to AI planning, in that the modeling language, DyPDL, is inspired by PDDL (Ghallab et al. 1998), and the existing prototype solver, CAASDy, employs heuristic search.

Although CAASDy outperforms commercial MIP and CP solvers on multiple problem classes, it has a serious limitation since it uses A* (Hart, Nilsson, and Raphael 1968): it either finds an optimal solution or does not find any solution at all. As many combinatorial optimization problems are NP-hard, optimally solving large-scale problems requires prohibitive computational time and/or space. In contrast, MIP and CP solvers are usually anytime, i.e., they often quickly find a feasible solution and continuously improve until, if given enough time, proving optimality. With an anytime solver, a user can trade off the computational time for solution quality while still proving optimality given enough run-time.

In this paper, we develop solvers for DIDP using anytime heuristic search algorithms. We show the state-of-the-art performance of our solvers through a comprehensive evaluation. Concretely, our contributions are as follows:

1. We develop DIDP solvers using six anytime heuristic search algorithms previously used in problem-specific settings.
2. We evaluate the anytime solvers on nine problem classes, demonstrating state-of-the-art performance across most problem classes.
3. We demonstrate that the best solver, complete anytime beam search (CABS), outperforms CAASDy, MIP, and CP on multiple problem classes, closing a number of open problem instances.

Domain-Independent Dynamic Programming

In DP, the cost of a problem is represented by recursive equations defining a *value function*, which maps a state to a numeric value. To formulate recursive equations as a DP model, DIDP uses DyPDL, a modeling formalism independent of solving algorithms (Kuroiwa and Beck 2023b). A DyPDL model is a tuple $\langle \mathcal{V}, S^0, \mathcal{K}, \mathcal{T}, \mathcal{B}, \mathcal{C}, h \rangle$, where $\mathcal{V} = \{v_1, ..., v_n\}$ is the set of *state variables*, S^0 is the *target state*, $\mathcal{K}$ is the set of *constants*, $\mathcal{T}$ is the set of *transitions*, $\mathcal{B}$ is the set of *base cases*, $\mathcal{C}$ is the set of *state constraints*, and h is the *dual bound*. Each state variable $v_i \in \mathcal{V} = \{v_1, ..., v_n\}$ is either of a *set*, an *element*, or a *numeric variable* and has a domain D_i. A set or an element variable v_i is associated with a set of n_i *objects* $N_i = \{0, ..., n_i - 1\}$, $D_i = 2^{N_i}$ if v_i is a set variable, and $D_i = N_i$ if v_i is an element variable. If v_i is a numeric variable, $D_i = \mathbb{R} \cup \{\infty\}$. A *state* is a value assignment to the variables, represented by a tuple $\langle d_1, ..., d_n \rangle \in \mathcal{D}$ where $\mathcal{D}$ is the cartesian product of $D_1, ..., D_n$. We denote the value of a variable v_i in a state

S by $S[v_i]$. For an element or a numeric variable, *preference*, either of less or more, can be specified, and such a variable is called a *resource variable*. Let $\mathcal{D}_{\leq}$ be resource variables where less is preferred, $\mathcal{D}_{\geq}$ be resource variables where more is preferred, and $\mathcal{D}_{=}$ be the other state variables. For two states S and S', S dominates S', i.e., S leads to an equal or better solution than S', denoted by $S' \preceq S$, if $\forall v \in \mathcal{D}_{\leq}, S[v] \leq S'[v]$, $\forall v \in \mathcal{D}_{\geq}, S[v] \geq S'[v]$, and $\forall v \in \mathcal{D}_{=}, S[v] = S'[v]$.

The target state $S^0 \in \mathcal{D}$ is a state for which we want to compute the value function. A constant in $\mathcal{K}$ can be a *set*, an *element*, or a *numeric constant*. A set constant is a subset of N_i for some v_i, an element constant is an element of N_i, and a numeric constant is a real value. A transition $\tau \in \mathcal{T}$ is a 4-tuple $\langle \mathsf{eff}_\tau, \mathsf{cost}_\tau, \mathsf{pre}_\tau, \mathsf{forced}_\tau \rangle$. The *effect* $\mathsf{eff}_\tau : \mathcal{D} \to \mathcal{D}$ maps a state S to another state $S[\![\tau]\!]$, the *cost expression* $\mathsf{cost}_\tau : \mathbb{R} \cup \{\infty\} \times \mathcal{D} \to \mathbb{R} \cup \{\infty\}$ maps a real value r and a state S to a value $\mathsf{cost}_\tau(r, S)$. *Preconditions* pre_τ are conditions on state variables, e.g., $v \geq 0$ for a numeric variable v, and τ is *applicable* in a state S only if all preconditions are satisfied, denoted by $S \models \mathsf{pre}_\tau$. The last element, forced_τ, is a boolean, and τ is a *forced transition* when $\mathsf{forced}_\tau = \top$. When a forced transition is applicable, all other transitions are ignored. The set of applicable transitions in a state S is

$$\mathcal{T}(S) = \begin{cases} \{\tau\} & \text{if } \exists \tau \in \mathcal{T}, S \models \mathsf{pre}_\tau \wedge \mathsf{forced}_\tau \\ \{\tau \in \mathcal{T} \mid S \models \mathsf{pre}_\tau\} & \text{otherwise.} \end{cases}$$

If $S \models \mathsf{pre}_\tau$ for multiple forced transitions τ, we assume that the first defined one is used. Effects, the cost expression, and preconditions are described by *set*, *element*, *numeric*, or *boolean expressions*, predefined mathematical operations on variables and constants. For example, $v + k$ with a numeric variable v and a constant k is a numeric expression, and $U \setminus \{j\}$ with a set variable U and a constant j is a set expression.

Base cases in $\mathcal{B}$ and state constraints in $\mathcal{C}$ are conditions. When a base case is satisfied by a state, the state is called a *base state*. The dual bound $h : \mathcal{D} \to \mathbb{R} \cup \{\infty\}$ maps a state S to a real value $h(S)$. Base cases, state constraints, and the dual bound are also described by expressions.

While DyPDL can be used for maximization, we focus on minimization in this paper. The optimal cost of a DyPDL model is computed by the following recursive equations.

$$\text{compute } V(S^0) \tag{1}$$
$$V(S) = \infty \qquad \text{if } S \not\models \mathcal{C} \tag{2}$$
$$V(S) = 0 \qquad \text{if } \exists B \in \mathcal{B}, S \models B \tag{3}$$
$$V(S) = \min_{\tau \in \mathcal{T}(S)} \mathsf{cost}_t(V(S[\![\tau]\!]), S) \tag{4}$$
$$V(S) \leq V(S') \qquad \text{if } S' \preceq S \tag{5}$$
$$V(S) \geq h(S). \tag{6}$$

When multiple conditions are satisfied by S, the first defined one is active. In Equation (4), we assume that $\min_{\tau \in \mathcal{T}(S)} \mathsf{cost}_\tau(V(S[\![\tau]\!]), S) = \infty$ if $\mathcal{T}(S) = \emptyset$.

Given a sequence of transitions $x = \langle x_1, ..., x_m \rangle$, let $\langle S^0, ..., S^m \rangle$ be a sequence of states with $S^i = S^{i-1}[\![x_i]\!]$. Let $C_i(x) = \mathsf{cost}_{x_{i+1}}(C_{i+1}(x), S^i)$ for $i = 0, ..., m-1$ with $C_m(x) = 0$. If $V(S^0) < \infty$, then $V(S^0)$ is the minimum $C_0(x)$ satisfying the following conditions: S^m is a base state; S^i satisfies the state constraints for $i = 0, ..., m$; $x_{i+1} \in \mathcal{T}(S^i)$ for $i = 0, ..., m-1$. Therefore, if x satisfies the above conditions, we call x a *solution* of the DyPDL model. If $C_0(x) = V(S^0)$, we call x an *optimal solution*.

Cost-Algebraic Heuristic Search Solver for DyPDL

A solution of a DyPDL model can be considered a path in a *state space*, a directed graph $\langle V, E \rangle$, where $V = \{S \in \mathcal{D} \mid S \models \mathcal{C}\}$ and $E = \{(S, S') \in V \times V \mid \exists \tau \in \mathcal{T}(S), S' = S[\![\tau]\!]\}$. When the cost expressions satisfy *cost-algebra* (Edelkamp, Jabbar, and Lafuente 2005), the optimal solution can be computed by a cost-algebraic search algorithm (Kuroiwa and Beck 2023b). For minimization of a non-negative real value, a cost-algebra requires that each transition τ has a cost expression in the form of $\mathsf{cost}_\tau(r, S) = w_\tau(S) \times r$ where $w_\tau(S) \in \mathbb{R}_0^+ \cup \{\infty\}$ is a numeric expression depending on τ and $\times$ is an isotonic binary operator such that $\langle \mathbb{R}_0^+ \cup \{\infty\}, \times, 0 \rangle$ is a monoid. For example, binary operators $+$ and $\max$ satisfy the conditions. The cost of a solution x corresponds to the path cost in a weighted graph with weight $w_\tau(S)$ for $(S, S[\![\tau]\!]) \in E$, i.e., $\bigtimes_{i=1}^{m} w_{x_i}(S^{i-1})$. Based on this observation, Kuroiwa and Beck (2023b) proposed CAASDy, an exact solver using the cost-algebraic version of A* (Hart, Nilsson, and Raphael 1968; Edelkamp, Jabbar, and Lafuente 2005). A* uses an *admissible heuristic function*, which underestimates the shortest path cost from a node. In CAASDy, the dual bound h defined in a DyPDL model is used as a heuristic function.

Anytime Heuristic Search Algorithms

Since CAASDy uses A*, the first solution found is optimal. However, a user may prefer an anytime algorithm, which continuously improves the solution quality over time and eventually proves optimality. In this section, we introduce six anytime heuristic search algorithms previously used for combinatorial optimization in problem-specific settings.

Generic pseudo-code of a heuristic search algorithm is shown in Algorithm 1. In this algorithm, a set of generated states G and an *open list* O are maintained; both initially contain S^0. For each state S, the *g-value* $g(S)$ is the path cost from S^0 to S, the *h-value* $h(S)$ is a lower bound of $V(S)$, the *f-value* $f(S)$ is a lower bound of the cost of a path from S^0 to a base state via S, and $x(S)$ is the path from S^0 to S. In addition, the best found solution $\overline{x}$ and its cost $\overline{f}$ are maintained. At each step, a search algorithm removes a state S from O according to some criterion in lines 5 and 6. If S is a base state, then the path from S^0 to S is a solution, and the best solution is updated if $g(S) < \overline{f}$. Since the f-values are lower bounds of the optimal cost, states having higher f-values than the new solution cost are removed from O. Otherwise, in lines 12-20, *successor states* are generated by applicable transitions and inserted into O if their f-values are less than $\overline{f}$. This procedure is called the *expansion* of S, and we say that the algorithm *expands* S. In line 17, the path to a successor state $S[\![\tau]\!]$ is computed by extending the path to S with τ. When O becomes empty, $\overline{x}$ is returned. The problem is infeasible if $\overline{x}$ is NULL.

If a state S having the minimum f-value in O is selected

Algorithm 1: Cost-Algebraic Heuristic Search for DyPDL.

```
1: O, G ← {S^0}.
2: g(S^0) ← 0, f(S^0) ← h(S^0), x(S^0) ← ⟨⟩.
3: f̄ ← ∞, x̄ ← NULL.
4: while O ≠ ∅ do
5:    Let S ∈ O.
6:    O ← O \ {S}.
7:    if ∃B ∈ B, S ⊨ B then
8:       if g(S) < f̄ then
9:          f̄ ← g(S), x̄ ← x(S).
10:         O ← {S' ∈ O | f(S') < f̄}.
11:   else
12:      for all τ ∈ T(S) : S⟦τ⟧ ⊨ C do
13:         g^τ ← g(S) × w_τ(S).
14:         if ∄S' ∈ G, S⟦τ⟧ ⪯ S' ∧ g^τ ≥ g(S') then
15:            g(S⟦τ⟧) ← g^τ.
16:            f(S⟦τ⟧) ← g(S⟦τ⟧) × h(S⟦τ⟧).
17:            x(S⟦τ⟧) ← ⟨x(S); τ⟩.
18:            if f(S⟦τ⟧) < f̄ then
19:               G ← G ∪ {S⟦τ⟧}.
20:               O ← O ∪ {S⟦τ⟧}.
21: return x̄.
```

in line 5, it results in cost-algebraic A*. When multiple states have the same f-value in O, the selection depends on the tie-breaking strategy. In CAASDy, the state minimizing the h-value is selected, and if multiple states minimize the f-value and the h-value, the tie is broken according to a data structure or an algorithm used in the implementation (binary heap for CAASDy). In the following anytime algorithms, when states are ordered by the f-values, we use the same tie-breaking strategy.

Depth-First Branch-and-Bound

Depth-first branch-and-bound (DFBnB) performs depth-first search. In line 5, the state maximizing the number of edges in a path from S^0 is selected. When multiple states have the maximum depth, the state having the minimum f-value is selected. Practically, a stack is used to implement O, and successor states are pushed onto the stack in the descending order of the f-values. For example, for combinatorial optimization, DFBnB has been applied to sequential order problems (SOP) (Libralesso et al. 2020), traveling salesperson problems (TSP), and single machine scheduling (Vadlamudi et al. 2012; Vadlamudi, Aine, and Chakrabarti 2016).

Cyclic-Best First Search

Cyclic-best first search (CBFS) (Kao, Sewell, and Jacobson 2009) maintains a priority queue $O_i \subseteq O$ for each depth 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 5 and inserts successor states into O_{i+1} in line 20. Then CBFS increases i by 1 or sets i to 0 if it is the maximum depth. 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 10 or $O_j = \emptyset$ for all $j \geq i$. While different priorities are used in problem specific settings such as single machine scheduling (Kao, Sewell, and Jacobson 2009) and simple assembly line balancing problems (Sewell and Jacobson 2012; Morrison, Sewell, and Jacobson 2014), our implementation selects the state minimizing the f-value. We implement each priority queue using a binary heap.

Anytime Column Search

Anytime column search (ACS) (Vadlamudi et al. 2012) can be considered a generalized version of CBFS, expanding b states at each depth. While CBFS increases the depth i after each expansion, ACS increases i by 1 after expanding the best b states from the priority queue O_i or when O_i becomes empty, where b is a parameter.

Anytime column progressive search (ACPS) (Vadlamudi et al. 2012) 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 when a new best solution is found or $O_j = \emptyset$ for all $j \geq i$. Our implementation of ACPS expands the b states minimizing the f-values and uses a binary heap for each priority queue. For combinatorial optimization, ACS and ACPS were evaluated on TSP (Vadlamudi et al. 2012).

Anytime Pack Search

Anytime pack search (APS) (Vadlamudi, Aine, and Chakrabarti 2016) maintains the set of the best b states $O_b \subseteq O$, initialized with $\{S^0\}$, the set of the best successor states O_c, and a suspend list $O_s \subseteq O$. APS removes states from O_b in line 6 and inserts the best b successor states according to the 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.

Anytime pack progressive search (APPS) (Vadlamudi, Aine, and Chakrabarti 2016) 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. Our implementation inserts the b states minimizing the f-values into O_c and uses binary heaps for O_b, O_c, and O_s. We use $\delta = 1$ and $\overline{b} = \infty$ following the configuration in TSP and single machine scheduling (Vadlamudi, Aine, and Chakrabarti 2016).

Discrepancy-Based Search

Discrepancy-based search (Harvey and Ginsberg 1995) 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) (Beck and Perron 2000) 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. Our implementation uses the f-value as the priority, where smaller f-values are preferred. Two stacks $O_0 \subseteq O$ and $O_1 \subseteq O$ are maintained, $O_0 = \{S^0\}$ and $O_1 = \emptyset$ at the beginning, and a state is popped from O_0 in line 6. Successor states with the discrepancy between $(i-1)k$ and $ik-1$ are pushed into O_0, and other states are pushed into O_1. Here, the successor states are pushed onto the stacks in the descending order of the f-values. When O_0 becomes empty, it is swapped with O_1, and i is increased by 1. In this implementation, 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 $(i+1)k-1$. We use $k = 1$ for our experiments. Discrepancy-based search was originally proposed as tree search (Harvey and Ginsberg 1995; Beck and Perron 2000) and later applied to state space search for SOP (Libralesso et al. 2020).

Complete Anytime Beam Search

Beam search maintains only the best b states at each depth and terminates at the depth where a solution is found. While beam search itself is not anytime and has no guarantee of optimality, complete anytime beam search (CABS) (Zhang 1998) is an anytime algorithm based on beam search. CABS sequentially performs beam search with increasing b until a stopping condition is satisfied. Following the configuration used in SOP (Libralesso et al. 2020) and the flowshop permutation (Libralesso et al. 2022), we double b at each iteration. CABS proves optimality when it searches all states S with $f(S) < \overline{f}$ and finds no better solution. As CABS does not fit into Algorithm 1, we show its pseudo-code in Algorithm 2. At each depth, CABS keeps successor states in G and moves the b best states to O. Our implementation keeps b states minimizing the f-values in a binary heap.

Other than CABS, beam search can be combined with A* to yield another anytime heuristic search algorithm, which was used for a job sequencing problem (Horn, Raidl, and Blum 2019).

Algorithm 2: Complete Anytime Beam Search for DyPDL.

```
1: g(S^0) ← 0, f(S^0) ← h(S^0), x(S^0) ← ⟨⟩.
2: f̄ ← ∞, x̄ ← NULL.
3: b ← 1.
4: complete ← ⊥, new_solution ← ⊥.
5: while ¬complete ∨ new_solution do
6:    complete ← ⊤, new_solution ← ⊥.
7:    O ← {S^0}.
8:    while ¬new_solution ∧ O ≠ ∅ do
9:       G ← ∅.
10:      for all S ∈ O do
11:         if ∃B ∈ B, S ⊨ B then
12:            if g(S) < f̄ then
13:               f̄ ← g(S), x̄ ← x(S).
14:               new_solution ← ⊤.
15:         else
16:            for all τ ∈ T(S) : S⟦τ⟧ ⊨ C do
17:               g^τ ← g(S) × w_τ(S).
18:               if ∄S' ∈ G, S⟦τ⟧ ⪯ S' ∧ g^τ ≥ g(S') then
19:                  g(S⟦τ⟧) ← g^τ.
20:                  f(S⟦τ⟧) ← g(S⟦τ⟧) × h(S⟦τ⟧).
21:                  x(S⟦τ⟧) ← ⟨x(S); τ⟩.
22:                  if f(S⟦τ⟧) < f̄ then
23:                     G ← G ∪ {S⟦τ⟧}.
24:      if |G| ≤ b then
25:         O ← G.
26:      else
27:         O ← the best b states in G.
28:         complete ← ⊥
29:   b ← 2b.
30: return x̄.
```

Empirical Evaluation

We experimentally compare the six anytime heuristic search algorithms using combinatorial optimization problems. In addition, we evaluate CAASDy and MIP and CP models.

Benchmark Problems and Models

For benchmark problems, Kuroiwa and Beck (2023b) used six combinatorial optimization problems: traveling salesperson problems with time windows (TSPTW), capacitated vehicle routing problems (CVRP), simple assembly line balancing problems (SALBP-1), bin packing, the minimization of open stacks problem (MOSP) (Yuen and Richardson 1995), and graph-clear (Kolling and Carpin 2007). We use the same DP models for SALBP-1, bin packing, MOSP, and graph-clear. For TSPTW and CVRP, we present new DP models in the appendix (Kuroiwa and Beck 2023a). While the previous work used the trivial dual bounds of zero, we replace them with more sophisticated dual bounds. In addition, we add a state constraint based on the capacity in the DP model for CVRP. The MIP and CP models are the same as Kuroiwa and Beck (2023b) for all problems except for graph-clear, where we use CPS (Morin et al. 2018) as the CP model because it tends to find better solutions in less time. We also use the same instances while adding larger instance sets for CVRP: set M (Christofides, Mingozzi, and Toth 1979), set X (Uchoa et al. 2017), and DIMACS (DIMACS 2021).

In addition, we use three more problems: one-to-one multi-commodity pick and delivery traveling salesperson problems, talent scheduling, and single machine scheduling to minimize the total weighted tardiness.

Multi-Commodity Pick and Delivery TSP A one-to-one multi-commodity pick and delivery traveling salesperson problem (m-PDTSP) (Hernández-Pérez and Salazar-González 2009) is to pick 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} 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 $0 \in N$, visits each customer once, and stops at $n+1 \in N$. The objective is to minimize the total travel time.

We propose a DP model based on the 1-PDTSP reduction (Gouveia and Ruthmair 2015). 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. A numeric constant $\delta_j = \sum_{k \in M: p_k = j} w_k - \sum_{k \in M: d_k = j} w_k$ represents the net change of the load at customer j. A set constant $P_j = \{p_k \mid k \in M : d_k = j\}$ represents the customers that must be visited before j. The following set expression gives the set of customers that can be visited next.

$$R(U, i, l) = \{j \in U \mid (i, j) \in A \wedge l + \delta_j \leq q \wedge P_j \cap U = \emptyset\}.$$

We also use a lower bound adapted from SOP (Libralesso et al. 2020). A numeric constant $c_j^{\text{in}} = \min_{k \in N: (k,j) \in A} c_{kj}$ is the minimum travel time to customer j, and $c_j^{\text{out}} = \min_{k \in N: (j,k) \in A} c_{jk}$ is the minimum travel time from j. We use $c_0^{\text{in}} = c_{n+1}^{\text{out}} = 0$. Since all customers in U must be arrived at and departed from, the sum of c_j^{in} or c_j^{out} over $j \in U$ gives a lower bound. We have the following DP model.

$$\begin{aligned}
&\text{compute } V(N \setminus \{0, n+1\}, 0, 0) \\
&V(\emptyset, i, l) = c_{i,n+1} \\
&V(U, i, l) = \min_{j \in R(U,i,l)} c_{ij} + V(U \setminus \{j\}, j, l + \delta_j) \\
&V(U, i, l) \leq V(U, i, l') \qquad \text{if } l \leq l' \\
&V(U, i, l) \geq \max \left\{ \sum_{j \in (U \cup \{n+1\}) \setminus \{i\}} c_j^{\text{in}}, \sum_{j \in U \cup \{i\}} c_j^{\text{out}} \right\}.
\end{aligned}$$

The second line is implemented by a transition to visit $n+1$ with a precondition $U = \emptyset$ and a base case $U = \emptyset \wedge i = n+1$.

For MIP, we use the MCF2C+IP formulation (Letchford and Salazar-González 2016). We use the CP model proposed by Castro, Cire, and Beck (2020) with an improved implementation as described in the appendix. In all models, unnecessary edges are removed from A by a preprocessing method (Letchford and Salazar-González 2016). For benchmarks, we use class1, class2, and class3 instances (Hernández-Pérez and Salazar-González 2009).

Talent Scheduling The talent scheduling problem 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.

We use the DP model proposed by Garcia de la Banda and Stuckey (2007). Let Q be a set variable representing a set of unscheduled scenes. In DyPDL, A_s is a set constant, and d_s and c_s are numeric constants. At each step, a scene s to shoot is selected from Q. A set expression $L(s, Q) = A_s \cup (\bigcup_{s' \in Q \setminus \{s\}} A_{s'} \cap \bigcup_{s' \in N \setminus (Q \cup \{s\})} A_{s'})$ 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 $s_2 \preceq s_1$. If all A_s are different, this relationship is a partial order: it is reflective 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 $s_3 \preceq s_1$. Since two scenes with the same set of actors are merged into a single scene in preprocessing without losing optimality, the following expression gives a non-empty set of candidate scenes to shoot next.

$$R(Q) = \{s_1 \in Q \mid \nexists s_2 \in Q \setminus \{s_1\}, s_2 \preceq s_1\}.$$

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 DP model.

$$\begin{aligned}
&\text{compute } V(N) \\
&V(\emptyset) = 0 \\
&V(Q) = d_s \sum_{a \in L(s,Q)} c_a + V(Q \setminus \{s\}) \\
&\qquad\qquad \text{if } \exists s \in Q, A_s = L(Q) \\
&V(Q) = \min_{s \in R(Q)} d_s \sum_{a \in L(s,Q)} c_a + V(Q \setminus \{s\}) \quad \text{otherwise} \\
&V(Q) \geq \sum_{s \in Q} d_s b_s.
\end{aligned}$$

We use a MIP model described in Qin et al. (2016). For CP, we extend the model used in Chu and Stuckey (2015) with the AllDifferent global constraint, which is redundant but slightly improves the performance in practice, as described in the appendix. In all models, a problem is simplified by preprocessing as described in Garcia de la Banda and Stuckey (2007). For a benchmark set, while Garcia de la Banda and Stuckey (2007) generated 100 random instances for each of 200 configurations, we use the first 5 instances for each configuration, resulting in 1000 instances in total.

Single Machine Total Weighted Tardiness In single machine scheduling to minimize the total weighted tardiness ($1||\sum w_i T_i$), 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. 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.

We use the DP model described in Abdul-Razaq, Potts, and Wassenhove (1990), where one job is scheduled at each step. Let U be a set variable representing the set of unscheduled jobs. In DyPDL, p_i, d_i, and w_i are numeric constants. A

numeric expression $T(i, U) = \max\{0, p_i + \sum_{j \in N \setminus U} p_j - d_i\}$ represents the tardiness of i when it is scheduled after $N \setminus U$. We introduce a set constant P_i, representing the set of jobs scheduled before i. While it is not defined in the problem, P_i can be extracted in preprocessing using precedence theorems without losing optimality (Emmons 1969).

compute $V(N)$

$$V(U) = \begin{cases} 0 & \text{if } U = \emptyset \\ \min\limits_{i \in U : P_i \cap U = \emptyset} w_i T(i, U) + V(U \setminus i) & \text{if } U \neq \emptyset. \end{cases}$$

For MIP, we use the formulation with assignment and positional date variables (F4) (Keha, Khowala, and Fowler 2009). For CP, we formulate a model using interval variables and precedence constraints, as described in the appendix. We extract precedence relations between jobs using the method proposed by Kanet (2007) and incorporate them in the DP and CP models but not in the MIP model as its performance is not improved. For a benchmark set, we use instances in OR-Library (Beasley 1990) with 40, 50, and 100 jobs.

Evaluation Measures

Coverage The performance of an exact method is evaluated by the number of instances where an optimal solution is found and its optimality is proved within time and memory limits. We also include the number of instances where its infeasibility is proved. We call this metric *coverage*.

Primal Integral To evaluate the performance of anytime solvers, we use the *primal integral* (Berthold 2013), which considers the balance between the solution quality and computational time. For an optimization problem, let x^t be a solution found by an algorithm at time t, x^* be an optimal (or best-known) solution, and f be a function mapping a solution to its cost. The *primal gap* function p is

$$p(t) = \begin{cases} 1 & \text{if no } x^t \text{ or } f(x^t) f(x^*) < 0 \\ 0 & \text{if } f(x^t) = f(x^*) = 0 \\ \frac{|f(x^*) - f(x^t)|}{\max\{|f(x^*)|, |f(x^t)|\}} & \text{otherwise.} \end{cases}$$

It takes a value in $[0, 1]$, and lower is better. We use $p(T)$, the primal gap at the time limit T, to measure the final solution quality. Let $t_i \in [0, T]$ for $i = 1, ..., l-1$ be a time point when a new better solution is found by an algorithm, $t_0 = 0$, 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. $P(T)$ 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 $p(t) = 0$, corresponding to the time to prove infeasibility.

Experimental Results

We implement the anytime algorithms using Rust 1.59.0.[1] We use Python 3.10.2 scripts to convert a problem instance to a YAML file of a DP model. We implement MIP models with Gurobi 9.5.0 and CP models with CP Optimizer from CPLEX Optimization Studio 22.1.0 using Python 3.10.2. 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 (Tange 2011).

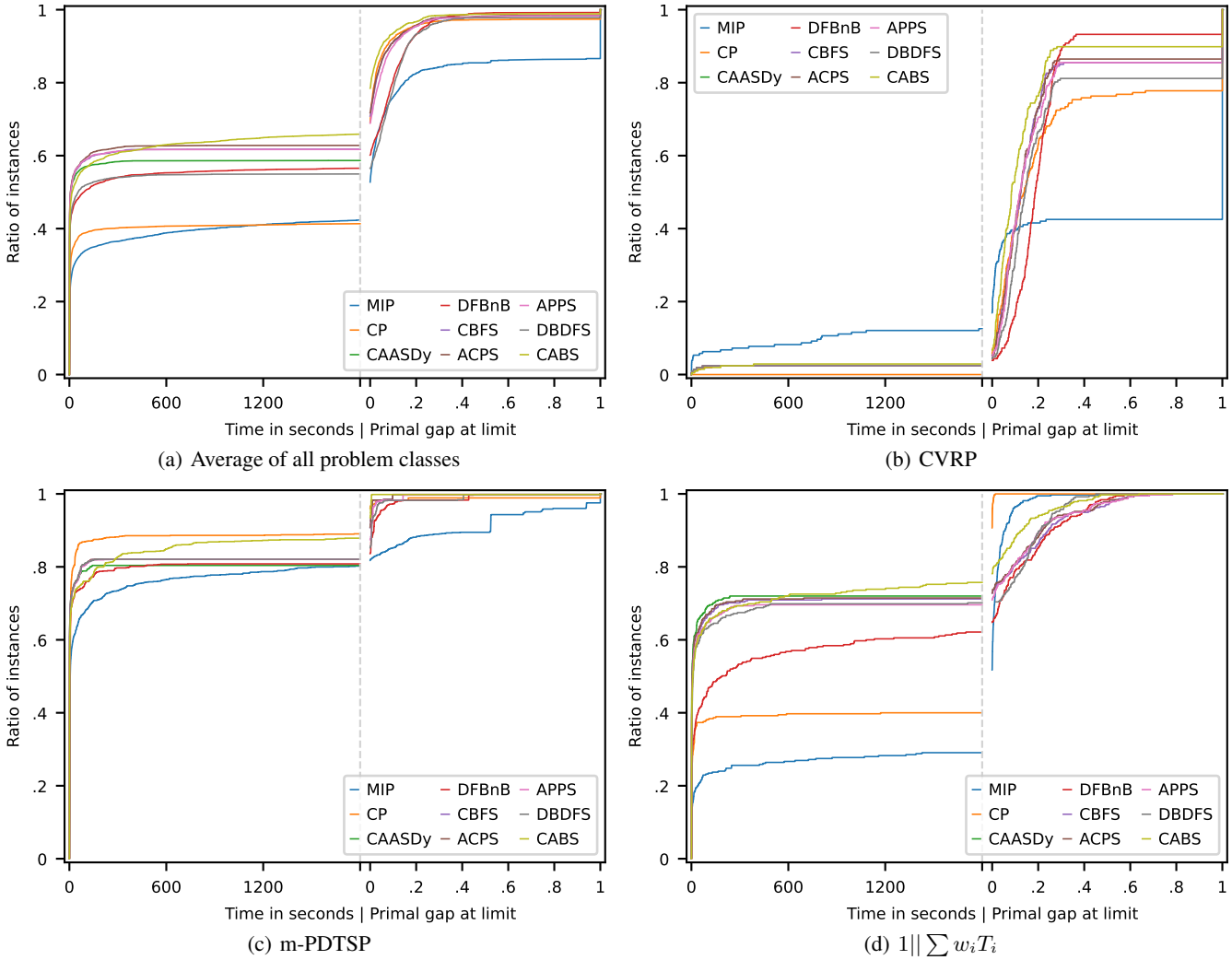
We show the total coverage on each problem class in Table 1. The instance set-wise coverage is shown in the appendix. CABS solves the largest number of instances on average and in six out of nine problem classes. Heuristic search algorithms almost always outperform MIP and CP on TSPTW, SALBP-1, MOSP, graph-clear, talent scheduling, and $1||\sum w_i T_i$. Among the heuristic search algorithms, CABS is the best in all problem classes except for bin packing (see the appendix). The other anytime heuristic search algorithms have lower coverage than CAASDy in SALBP-1, talent scheduling, and $1||\sum w_i T_i$. In m-PDTSP, CABS is the best among all the methods in the class1 instances while CP is the best in class3. For open instances in class1, all the heuristic search algorithms prove the infeasibility of p43.3Q10max5 and p43.3Q4max1 and find optimal solutions for p43.3Q15max5 and p43.3Q6max1, and CABS finds an optimal solution for p43.3Q7max1 additionally. The CP model finds optimal solutions for p43.3Q20max5 and p43.3Q8max1.

Figure 1 presents the ratio of the coverage over the number of instances against time and the ratio of instances against the primal gap at the time limit. Each line shows the cumulative ratio of instances for one method, so higher and more left is better. To compute the primal gap, for the best-known cost $f(x^*)$, we use the best solution cost found by methods evaluated in our experiment and the best solution cost reported by previous work, which is available online for TSPTW, CVRP, and $1||\sum w_i T_i$. Figure 1(a) shows the average of the ratios over all problem classes. While the results for TSPTW, SALBP-1, MOSP, and graph-clear are similar to Figure 1(a), other problems have different tendencies. We show the plots for CVRP, m-PDTSP, whose result is similar to bin packing, and $1||\sum w_i T_i$, whose result is similar to talent scheduling. The plots for the remaining problems are included in the appendix. CBFS, ACPS, and APPS have similar performance, outperforming CAASDy, DFBnB, and DBDFS. While CAASDy, CBFS, ACPS, and APPS have higher coverage at first, CABS eventually outperforms them because it spends more time proving optimality but consumes less memory. While the others generate a state only once, CABS generates the same state in multiple iterations. However, CABS only stores states at a particular depth while the others store all generated states. In addition, once a solution with the cost $\hat{f}$ is found, CABS avoids generating states having higher f-values in subsequent iterations. The other anytime heuristic search algorithms also benefit from this pruning, which results in higher coverage than CAASDy in some problem classes, but they still store already generated states S with $f(S) \geq \hat{f}$. Indeed, while CABS rarely runs out of memory, the other heuristic search algorithms often reach the memory limit before the time limit (see Table 5 in the appendix).

[1] https://github.com/domain-independent-dp/didp-rs

	MIP	CP	CAASDy	DFBnB	CBFS	ACPS	APPS	DBDFS	CABS
TSPTW (340)	227	47	257	239	257	257	257	256	**259**
CVRP (207)	**26**	0	5	5	5	5	5	5	6
m-PDTSP (1178)	945	**1049**	947	952	967	967	967	967	1035
SALBP-1 (2100)	1357	1584	1653	1618	1466	1609	1625	1393	**1801**
Bin Packing (1615)	1157	**1234**	922	517	1109	1138	1032	424	1163
MOSP (570)	224	437	483	524	523	524	523	522	**527**
Graph-Clear (135)	24	1	76	99	100	100	99	82	**103**
Talent Scheduling (1000)	0	0	207	189	206	206	193	199	**237**
$1\|\|\sum w_i T_i$ (375)	109	150	270	233	267	268	261	263	**284**
Average ratio	0.424	0.416	0.587	0.565	0.628	0.628	0.617	0.549	**0.659**

Table 1: Coverage. 'Average ratio' is the ratio of the coverage to the number of instances averaged over all problem classes.



(a) Average of all problem classes

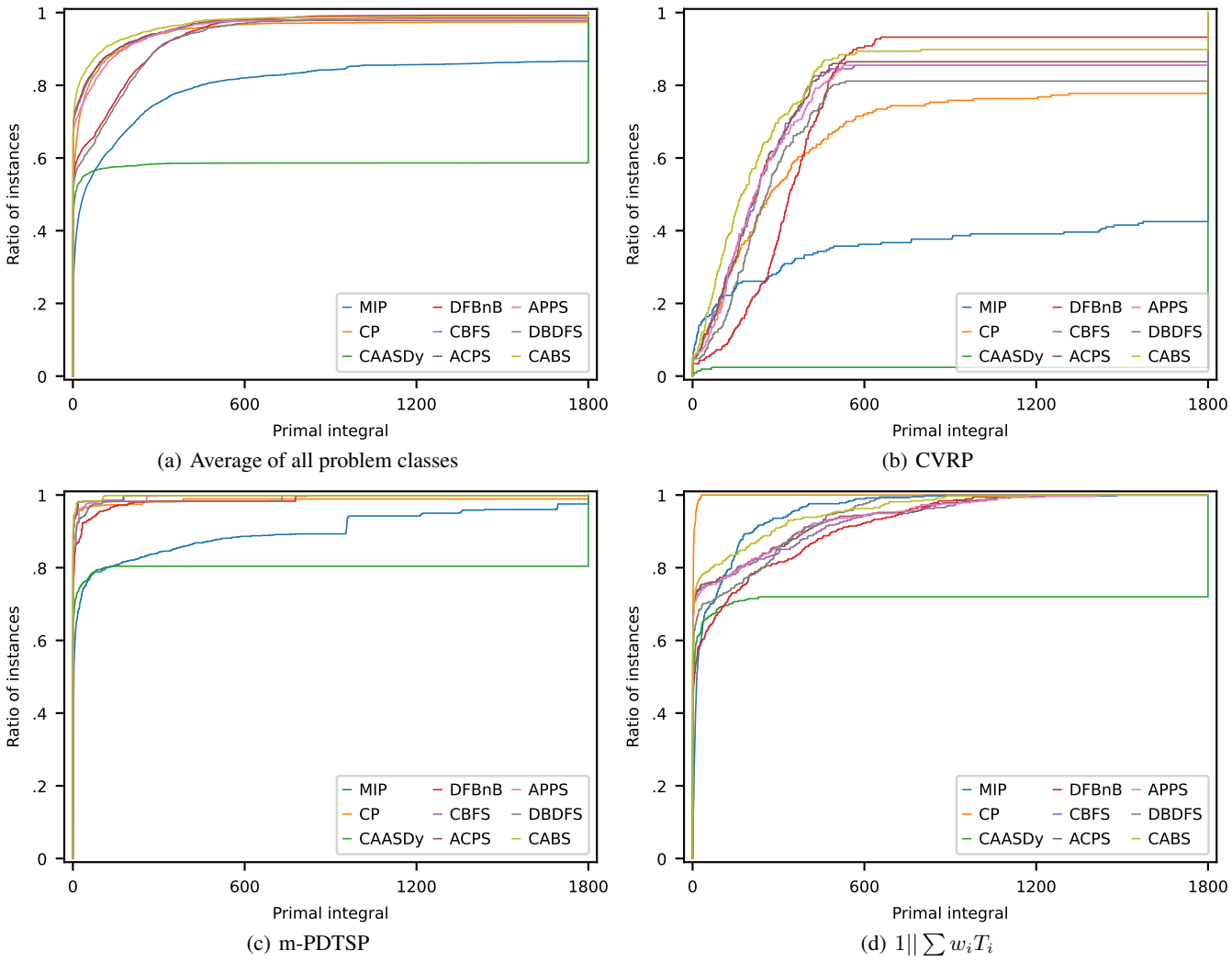
(b) CVRP

(c) m-PDTSP

(d) $1||\sum w_i T_i$

Figure 1: The ratio of the coverage over the number of instances against time (left) and the ratio of instances against the primal gap at the time limit (right). Higher and left is better. The figure is best viewed in color.

CABS has a smaller primal gap at the time limit than the other methods on average. CBFS, ACPS, and CABS improve the solution cost for rbg193.2 in the AFG set of TSPTW from 12142 to 12139. In CVRP, MIP performs the best in small instances such as set A and set B but it fails to find a feasible solution for almost all instances of DI-



(a) Average of all problem classes

(b) CVRP

(c) m-PDTSP

(d) $1||\sum w_i T_i$

Figure 2: The ratio of instances against the primal integral. Higher and left is better. The figure is best viewed in color.

MACS, set M, and set X. As a result, heuristic search algorithms have more instances with a gap smaller than 0.1. In $1||\sum w_i T_i$ and talent scheduling, although heuristic search algorithms have higher coverage, CP is the best in the primal gap. In contrast, CABS has a smaller primal gap in m-PDTSP while CP has higher coverage. In bin packing, CP is the best in coverage and close to CABS in the primal gap (see the appendix).

We also show the ratio of instances against the primal integral in Figure 2. The tendency is similar to that of the primal gap. While CABS outperforms the others on average, CP is the best in $1||\sum w_i T_i$ and talent scheduling. In CVRP, MIP has more instances when the primal gap is around 50 while CABS has more instances in other regions.

Overall, CABS is the best anytime solver for DIDP on average in all the measures, showing a significant improvement over CAASDy. CBFS, ACPS, and APPS also outperform CAASDy in the primal gap and the primal integral since CAASDy does not provide a solution when optimality is not proved. However, CAASDy has higher coverage than CBFS, ACPS, and APPS on multiple problems.

Conclusion

We proposed solvers for domain-independent dynamic programming using anytime heuristic search. Our solvers, especially complete anytime beam search (CABS), show good performance on multiple combinatorial optimization problems in terms of proving optimality, solution quality, and primal integral, outperforming mixed-integer programming (MIP), constraint programming (CP), and A*. However, in talent scheduling and single machine scheduling, although CABS solves more instances to optimality, CP is better in solution quality and primal integral. While we focused on anytime heuristic search algorithms that eventually find an optimal solution, developing a heuristic solver or a primal heuristic, which may not have an optimality guarantee but finds a good solution quickly, and combining it with another solver to prove optimality, is part of our future work.

Acknowledgments

This work was partially supported by the Natural Sciences and Engineering Research Council of Canada.

References

Abdul-Razaq, T. S.; Potts, C. N.; and Wassenhove, L. N. V. 1990. A Survey of Algorithms for the Single Machine Total Weighted Tardiness Scheduling Problem. *Discret. Appl. Math.*, 26: 235–253.

Beasley, J. E. 1990. OR-Library: Distributing Test Problems by Electronic Mail. *J. Oper. Res. Soc.*, 41: 1069–1072.

Beck, J. C.; and Perron, L. 2000. Discrepancy-Bounded Depth First Search. In *CPAIOR 2000*.

Berthold, T. 2013. Measuring the Impact of Primal Heuristics. *Oper. Res. Lett.*, 41: 611–614.

Castro, M. P.; Cire, A. A.; and Beck, J. C. 2020. An MDD-Based Lagrangian Approach to the Multicommodity Pickup-and-Delivery TSP. *INFORMS J. Comput.*, 32: 263–278.

Christofides, N.; Mingozzi, A.; and Toth, P. 1979. *Combinatorial Optimization*, volume 1, chapter The Vehicle Routing Problem, 315–338. Wiley Interscience.

Chu, G.; and Stuckey, P. J. 2015. Learning Value Heuristics for Constraint Programming. In *Proc. CPAIOR 2015*, 108–123.

DIMACS. 2021. the 12th DIMACS Implementaion Challenge. http://dimacs.rutgers.edu/programs/challenge/vrp/cvrp. Accessed: 2023-02-24.

Edelkamp, S.; Jabbar, S.; and Lafuente, A. L. 2005. Cost-Algebraic Heuristic Search. In *Proc. AAAI*, 1362–1367.

Emmons, H. 1969. One-Machine Sequencing to Minimize Certain Functions of Job Tardiness. *Oper. Res.*, 17: 701–715.

Freuder, E. 1997. In Pursuit of the Holy Grail. *Constraints*, 2: 57–61.

Garcia de la Banda, M.; and Stuckey, P. J. 2007. Dynamic Programming to Minimize the Maximum Number of Open Stacks. *INFORMS J. Comput.*, 19(4): 607–617.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL - The Planning Domain Definition Language.

Gouveia, L.; and Ruthmair, M. 2015. Load-Dependent and Precedence-Based Models for Pickup and Delivery Problems. *Comput. Oper. Res.*, 63: 56–71.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Syst. Sci. Cybern.*, 4(2): 100–107.

Harvey, W. D.; and Ginsberg, M. L. 1995. Limited Discrepancy Search. In *Proc. IJCAI*, 607–613.

Hernández-Pérez, H.; and Salazar-González, J. J. 2009. The Multi-Commodity One-to-One Pickup-and-Delivery Traveling Salesman Problem. *Eur. J. Oper. Res.*, 196: 987–995.

Horn, M.; Raidl, G.; and Blum, C. 2019. Job Sequencing with One Common and Multiple Secondary Resources: An A∗/Beam Search Based Anytime Algorithm. *Artif. Intell.*, 277.

Kanet, J. J. 2007. New Precedence Theorems for One-Machine Weighted Tardiness. *Math. Oper. Res.*, 32: 579–588.

Kao, G. K.; Sewell, E. C.; and Jacobson, S. H. 2009. A Branch, Bound, and Remember Algorithm for the $1|r_t|\sum t_i$ Scheduling Problem. *J. Sched.*, 12: 163–175.

Keha, A. B.; Khowala, K.; and Fowler, J. W. 2009. Mixed Integer Programming Formulations for Single Machine Scheduling Problems. *Comput. Ind. Eng.*, 56: 357–367.

Kolling, A.; and Carpin, S. 2007. The GRAPH-CLEAR Problem: Definition, Theoretical Properties and its Connections to Multi-robot Aided Surveillance. In *Proc. IROS*, 1003–1008.

Kuroiwa, R.; and Beck, J. C. 2023a. Appendix for Solving Domain-Independent Dynamic Programming Problems with Anytime Heuristic Search. https://tidel.mie.utoronto.ca/pubs/Appendix_Anytime_ICAPS23.pdf. Accessed: 2023-02-24.

Kuroiwa, R.; and Beck, J. C. 2023b. Domain-Independent Dynamic Programming: Generic State Space Search for Combinatorial Optimization. In *Proc. ICAPS*.

Letchford, A. N.; and Salazar-González, J. J. 2016. Stronger Multi-Commodity Flow Formulations of the (Capacitated) Sequential Ordering Problem. *Eur. J. Oper. Res.*, 251: 74–84.

Libralesso, L.; Bouhassoun, A.-M.; Cambazard, H.; and Jost, V. 2020. Tree search for the Sequential Ordering Problem. In *Proc. ECAI*, 459–465.

Libralesso, L.; Focke, P. A.; Secardin, A.; and Jost, V. 2022. Iterative beam search algorithms for the permutation flowshop. *Eur. J. Oper. Res.*, 301: 217–234.

Morin, M.; Castro, M. P.; Booth, K. E.; Tran, T. T.; Liu, C.; and Beck, J. C. 2018. Intruder Alert! Optimization Models for Solving the Mobile Robot Graph-Clear Problem. *Constraints*, 23(3): 335–354.

Morrison, D. R.; Sewell, E. C.; and Jacobson, S. H. 2014. An Application of the Branch, Bound, and Remember Algorithm to a New Simple Assembly Line Balancing Dataset. *Eur. J. Oper. Res.*, 236(2): 403–409.

Qin, H.; Zhang, Z.; Lim, A.; and Liang, X. 2016. An Enhanced Branch-and-Bound Algorithm for the Talent Scheduling Problem. *Eur. J. Oper. Res.*, 250: 412–426.

Sewell, E. C.; and Jacobson, S. H. 2012. A Branch, Bound, and Remember Algorithm for the Simple Assembly Line Balancing Problem. *INFORMS J. Comput.*, 24(3): 433–442.

Tange, O. 2011. GNU Parallel - The Command-Line Power Tool. *;login: The USENIX Magazine*, 36: 42–47.

Uchoa, E.; Pecin, D.; Pessoa, A.; Poggi, M.; Vidal, T.; and Subramanian, A. 2017. New Benchmark Instances for the Capacitated Vehicle Routing Problem. *Eur. J. Oper. Res.*, 257(3): 845–858.

Vadlamudi, S. G.; Aine, S.; and Chakrabarti, P. P. 2016. Anytime Pack Search. *Nat. Comput.*, 15: 395–414.

Vadlamudi, S. G.; Gaurav, P.; Aine, S.; and Chakrabarti, P. P. 2012. Anytime Column Search. In *AI 2012: Advances in Artificial Intelligence*, 254–265.

Yuen, B. J.; and Richardson, K. V. 1995. Establishing the Optimality of Sequencing Heuristics for Cutting Stock Problems. *Eur. J. Oper. Res.*, 84: 590–598.

Zhang, W. 1998. Complete Anytime Beam Search. In *Proc. AAAI*, 425–430.