Domain-Independent Dynamic Programming: Generic State Space Search for Combinatorial Optimization

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

For combinatorial optimization problems, model-based approaches such as mixed-integer programming (MIP) and constraint programming (CP) aim to decouple modeling and solving a problem: the 'holy grail' of declarative problem solving. We propose domain-independent dynamic programming (DIDP), a new model-based paradigm based on dynamic programming (DP). While DP is not new, it has typically been implemented as a problem-specific method. We propose Dynamic Programming Description Language (DyPDL), a formalism to define DP models, and develop Cost-Algebraic A* Solver for DyPDL (CAASDy), a generic solver for DyPDL using state space search. We formalize existing problem-specific DP and state space search methods for combinatorial optimization problems as DP models in DyPDL. Using CAASDy and commercial MIP and CP solvers, we experimentally compare the DP models with existing MIP and CP models, showing that, despite its nascent nature, CAASDy outperforms MIP and CP on a number of common problem classes.

Introduction

Combinatorial optimization is a central topic of artificial intelligence (AI) with many application fields including planning and scheduling. In model-based approaches to such problems, users formulate problems as mathematical models and use generic solvers to solve the models. While problem-specific algorithms can sometimes be more efficient, model-based approaches embody the quest for declarative general-purpose problem solving where it is sufficient to define the problem in order for it to be solved (Freuder 1997). While mixed-integer programming (MIP) and constraint programming (CP) are common in operations research (OR), domain-independent AI planning can also be considered a model-based approach.

Dynamic programming (DP) formulates a problem as a mathematical model using recursive equations with a state-based problem representation. Although problem-specific DP achieves state-of-the-art performance on multiple problems, little work has considered DP as a generic technology.

In this paper, we propose *domain-independent dynamic programming (DIDP)*, a new model-based paradigm for combinatorial optimization, based on DP. Separating DP as a problem form from DP as an algorithm, we define Dynamic Programming Description Language (DyPDL), an algorithm-independent modeling formalism for DP. We also develop Cost-Algebraic A* Solver for DyPDL (CAASDy), a DyPDL solver using a heuristic search algorithm. While we adopt a relatively simple algorithm, the cost algebraic version of A* (Edelkamp, Jabbar, and Lafuente 2005), we demonstrate that it performs better than state-of-the-art MIP and CP approaches in four out of six problem classes.

Dynamic Programming

In DP, a problem is formulated as a *state*, and the solution corresponds to a sequence of *decisions*. In a state S, one decision t is made from a set of applicable decisions $\mathcal{T}(S)$, and S is decomposed into a set of subproblems (states) $\mathcal{S}_t$. The optimal cost of a problem is given by a value function V. For a trivial subproblem S, $V(S)$ is defined by a constant. Otherwise, $V(S)$ is defined by a function F of a decision and costs of the subproblems in the following recursive equation:

$$V(S) = \min_{t \in \mathcal{T}(S)} F(t, \{V(S') \mid S' \in \mathcal{S}_t\}).$$

For maximization, $\min$ is replaced with $\max$. The equation is usually solved by a problem-specific algorithm.

DyPDL: A Modeling Formalism for DP

Dynamic Programming Description Language (DyPDL) is a solver-independent formalism for a DP model. DyPDL is inspired by AI planning formalisms such as STRIPS (Fikes and Nilsson 1971). While domain-independent AI planning takes the 'physics, not advice' approach, where a model contains only information necessary to define a problem, DyPDL allows a user to explicitly model implications of the definition. Such is the standard convention in OR and commonly exploited in DP algorithms (e.g., Dumas et al. (1995)). DyPDL enables a user to express such information in a DP model.

Example: TSPTW

In a traveling salesperson problem with time windows (TSPTW), a set of customers $N = \{0, .., n\}$ 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.

In the DP model proposed by Dumas et al. (1995), a state is a tuple of variables $\langle U, i, t \rangle$, which represents the set of unvisited customers, the current location, and the current time, respectively. In this model, one customer is visited at each step. The set of customers that can be visited next is $U' = \{j \in U \mid t + c_{ij} \leq b_j\}$, and $t^j = \max\{t + c_{ij}, a_j\}$ is the time when j is visited from the current state. Also, 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.

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

$$V(U, i, t) = \infty \qquad \text{if } \exists j \in U, t + c^*_{ij} > b_j \tag{2}$$

$$V(U, i, t) = \begin{cases} c_{i0} & \text{if } U = \emptyset \\ \min_{j \in U'} c_{ij} + V(U \setminus \{j\}, j, t^j) & \text{if } U \neq \emptyset \end{cases} \tag{3}$$

$$V(U, i, t) \leq V(U, i, t') \qquad \text{if } t \leq t' \tag{4}$$

$$V(U, i, t) \geq 0. \tag{5}$$

Objective (1) 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), the first line corresponds to returning to the depot from customer i, and the second line corresponds to visiting customer j from i. We assume that $V(U, i, t) = \infty$ if $U \neq \emptyset$ and $U' = \emptyset$.

Equation (2) and Inequalities (4) and (5) are not necessary to define a problem and are not present in the original DP model. However, they were used algorithmically by Dumas et al. (1995) as pruning rules; when enumerating states, a state $\langle U, i, t \rangle$ is ignored if $\exists j \in U, t + c^*_{ij} > b_j$ since j cannot be visited by its deadline. A state $\langle U, i, t' \rangle$ is ignored if a state $\langle U, i, t \rangle$ with $t \leq t'$ is already considered because smaller t leads to a better solution. We formulate these pruning rules as Equation (2) and Inequality (4). In addition, we use the trivial lower bound of 0 for V in Inequality (5).

A state may satisfy multiple conditions in the model; when $\exists j \in U, t + c^*_{ij} > b_j$ holds, $U \neq \emptyset$ also holds. In such a case, we assume that the first condition defined is active.

Formalism

A DyPDL model is a tuple $\langle \mathcal{V}, S^0, \mathcal{K}, \mathcal{T}, \mathcal{B}, \mathcal{C}, h \rangle$, where $\mathcal{V}$ is a set of *state variables*, S^0 is a *target state*, $\mathcal{K}$ is a set of *constants*, $\mathcal{T}$ is a set of *transitions*, $\mathcal{B}$ is a set of *base cases*, $\mathcal{C}$ is a set of *state constraints*, and h is a *dual bound*.

State Variables A state is defined by state variables, which can be *element, set*, and *numeric variables*. An element or a set variable is associated with *objects*, and the number of the objects is specified. If there are n objects, they are indexed from 0 to $n-1$, and the values of element and set variables can be $i \in \{0, ..., n-1\}$ and $M \subseteq \{0, ..., n-1\}$, respectively. The value of a numeric variable is a real number. In TSPTW, U is a set variable, i is an element variable, and t is a numeric variable. Objects $\{0, ..., n\}$ representing customers are associated with U and i.

For an element or a numeric variable, a *preference* for a greater or smaller value can be specified. If the preference is specified for a variable, it is called a *resource variable*. When all other variables are the same in two states S and S', if each resource variable in S is better than or equal to that of S' according to its preference, then $V(S)$ is assumed to be better than $V(S')$. In this case, we say that S dominates S', denoted by $S' \preceq S$. In TSPTW, $\langle U, i, t' \rangle \preceq \langle U, i, t \rangle$ if $t \leq t'$ as defined in Inequality (4). This dominance is not necessary to define a model but can help solvers if present.

Target State The target state S^0 must be specified as a full value assignment to state variables. The objective of a DP model is to compute $V(S^0)$, the value of the target state. In TSPTW, the target state is $\langle N \setminus \{0\}, 0, 0 \rangle$ as in Objective (1).

Constants A constant is a state-independent value. Constants can be *element, set, numeric*, and *boolean constants*. An element constant is a nonnegative integer representing the index of an object, and a set constant is a set of nonnegative integers representing a set of the indices of objects. A numeric constant is a real number and a boolean constant is a boolean value. Multidimensional tables of constants can be defined and indexed by objects. In TSPTW, a_i and b_i are constants in one-dimensional tables indexed by customer i, and c_{ij} and c^*_{ij} are constants in two-dimensional tables indexed by customers i and j.

Expressions Expressions are used in transitions, base cases, state constraints, and the dual bound to describe the computation of a value using the values of state variables and constants. When an expression e is evaluated given a state S, it returns a value $e(S)$. Depending on the type of the returned value, we define *element expressions*, *set expressions*, *numeric expressions*, and *conditions*.

An element expression returns the index of an object. It can refer to an element constant or variable and use arithmetic operations such as addition and subtraction on two element expressions. In TSPTW, j in Equation (3) is an element expression referring to the constant j.

A set expression returns a set of the indices of objects. It can refer to a set constant or variable, add (remove) an element expression to (from) a set expression, and take the union/intersection/difference of two set expressions. In TSPTW, $U \setminus \{j\}$ in Equation (3) is a set expression removing j from U.

A numeric expression returns a real number. It can refer to a numeric constant or variable, use arithmetic operations, and take the cardinality of a set expression. In TSPTW, $t^j = \max\{t + c_{ij}, a_j\}$ in Equation (3) is a numeric expression. It accesses a constant in a table, c_{ij}, using element expressions i and j. While j refers to a constant, i refers to a variable, so c_{ij} depends on a state. It is also possible to take the sum of numeric constants in a table using set expressions as shown below in the DP models for a simple assembly line balancing problem, bin packing, and graph-clear.

A condition returns a boolean value. For a condition c and a state S, we say $S \models c$ if $c(S) = \top$. In addition to preconditions of transitions, base cases, and state constraints, a condition can be used to define an 'if-then-else' expression, where

one expression is returned if $S \models c$ and another is returned if not. A condition can refer to a boolean constant, compare two element, numeric, or set expressions, and check whether an element is included in a set. The conjunction and the disjunction of two conditions are also conditions. In TSPTW, $\exists j \in U, t + c^*_{ij} > b_j$ in Equation (2) and $U = \emptyset$ in Equation (3) are conditions.

Transitions A transition is a decision in DP and defines a recursive formula. In DyPDL, we focus on problems where a state is transformed into another state by a decision, i.e., there is only one subproblem, so we call it a transition. A transition τ is a 4-tuple $\langle \mathsf{eff}_\tau, \mathsf{cost}_\tau, \mathsf{pre}_\tau, \mathsf{forced}_\tau \rangle$ where the set of *effects* eff_τ describes how to transform a state to another state, the *cost expression* cost_τ describes how to compute $V(S)$, the set of *preconditions* pre_τ describes when the transition is applicable, and forced_τ is a boolean indicating whether it is a *forced transition*. In TSPTW, transitions are defined in Equation (3).

For each state variable, eff_τ defines an expression to update it. By $S[\![\tau]\!]$, we denote the updated state by transition τ from state S. In TSPTW, in Equation (3), the set expression $U \setminus \{j\}$ updates the state variable U, the element expression j updates the state variable i, and the numeric expression $\max\{t + c_{ij}, a_j\}$ updates the numeric variable t.

The cost expression cost_τ is a numeric expression describing the computation of $V(S)$. In addition to variables and constants, it can use $V(S[\![\tau]\!])$, the value of the transformed state. By $\mathsf{cost}_\tau(V(S[\![\tau]\!]), S)$, we denote the value of cost_τ given $V(S[\![\tau]\!])$ and S. In TSPTW, the cost expression of a transition for $j \in U$ in Equation (3) is $c_{ij} + V(S[\![\tau]\!])$.

A precondition in pre_τ is a condition, and the transition τ is applicable in a state S only if $S \models p$ for each $p \in \mathsf{pre}_\tau$, denoted by $S \models \mathsf{pre}_\tau$. In TSPTW, $t + c_{ij} \leq b_j$ and $j \in U$ are preconditions of a transition defined in Equation (3). If $\mathsf{forced}_\tau = \top$, τ is a forced transition. When the preconditions of a forced transition are satisfied, all other transitions are ignored. Let $\mathcal{T}_f = \{\tau \in \mathcal{T} \mid \mathsf{forced}_\tau\}$ be the set of forced transitions. The set of applicable transitions in S is

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

If multiple forced transitions are applicable, the first one defined is used. A forced transition can be used to break symmetry as shown below in the DP model for bin packing.

In minimization, $V(S)$ is computed as the minimum over all applicable transitions, i.e., $V(S) = \min_{\tau \in \mathcal{T}(S)} \mathsf{cost}_\tau(V(S[\![\tau]\!]), S)$. When $\mathcal{T}(S) = \emptyset$, then $V(S) = \infty$ is assumed. In maximization, $V(S) = \max_{\tau \in \mathcal{T}(S)} \mathsf{cost}_\tau(V(S[\![\tau]\!]), S)$, and $V(S) = -\infty$ is assumed if $\mathcal{T}(S) = \emptyset$. In TSPTW, $V(S) = \min_{j \in U'} c_{ij} + V(U \setminus \{j\}, j, t^j)$ as in the second line of Equation (3).

Base Cases A base case is defined as a set of conditions to terminate the recursion. If $S \models \beta$ for every β in a base case B, denoted by $S \models B$, then $V(S) = 0$. We call such a state a *base state*. In TSPTW, since $V(U, i, t) = c_{i0}$ if $U = \emptyset$ in the first line of Equation (3), we use a base case $\{U = \emptyset, i = 0\}$ and introduce a transition with cost expression $c_{i0} + V(U, 0, t + c_{i0})$ and preconditions $U = \emptyset$ and $i \neq 0$.

$\mathcal{V}$	Type	Objects	Preference
U	set	customers N	
i	element	customers N	
t	numeric		less
$\mathcal{K}$	Type	Indices	
a_j	numeric	$j \in N$	
b_j	numeric	$j \in N$	
c_{jk}	numeric	$j, k \in N$	
c^*_{jk}	numeric	$j, k \in N$	
S^0	$\langle U = N \setminus \{0\}, i = 0, t = 0 \rangle$		
$\mathcal{B}$	$\{\{U = \emptyset, i = 0\}\}$		
$\mathcal{C}$	$\{\forall j \in U, t + c^*_{ij} \leq b_j\}$		
h	0		
$\mathcal{T}$	eff	cost	pre
visit j	$U \leftarrow U \setminus \{j\}$ $i \leftarrow j$ $t \leftarrow \max\{t + c_{ij}, a_j\}$	$c_{ij} + V(S[\![\tau]\!])$	$j \in U$ $t + c_{ij} \leq b_j$
return	$i \leftarrow 0$ $t \leftarrow t + c_{i0}$	$c_{i0} + V(S[\![\tau]\!])$	$U = \emptyset$ $i \neq 0$

Table 1: DyPDL representation of the DP model for TSPTW. No forced transition exists in this model.

State Constraints A state constraint is a condition that must be satisfied by all states. If a state does not satisfy a state constraint, it can be immediately discarded. In TSPTW, $V(U, i, t) = \infty$ if $\exists j \in U, t + c^*_{ij} > b_j$ (Equation (2)) is implemented as a state constraint $\forall j \in U, t + c^*_{ij} \leq b_j$.

Dual Bound The dual bound h is defined as a numeric expression, and $h(S)$ must be the lower (upper) bound on $V(S)$ for minimization (maximization). The dual bound is not required but can be exploited by a solver. In TSPTW, Inequality (5) defines a dual bound, $h(S) = 0$ for all S.

DyPDL Example for TSPTW

Table 1 presents the DyPDL representation of the DP model for TSPTW in (1)–(5). The transition to visit customer j is defined for all $j \in N$. There are no forced transitions, i.e., $\mathsf{forced}_\tau = \bot$ for all $\tau \in \mathcal{T}$.

YAML-DyPDL: An Implementation of DyPDL

We propose YAML-DyPDL, an implementation of DyPDL based on the YAML data format,[1] inspired by PDDL (Ghallab et al. 1998). A problem instance is represented by *domain* and *problem files*. While a domain file can be shared by multiple instances of the same problem, a problem file is specific to one problem instance. A domain file defines objects, state variables, state constraints, base cases, transitions, and the dual bound and declares tables of constants. A problem file defines the number of objects, the target state, and the values of the constants in the tables. In addition, state constraints, base cases, transitions, and dual bounds can be also defined in a problem file. We show an example of a domain file in Listing 1 and a problem file in Listing 2, which correspond to the DP model for TSPTW in (1)–(5). In this example, assuming that the triangle inequality holds, c^*_{ij} is

[1] https://yaml.org/

replaced with c_{ij}, and $t + c_{ij} \leq b_j$ is removed from preconditions of the transition to visit j because it is ensured by the state constraint $\forall j \in U, t + c_{ij} \leq b_j$.

In YAML, key-value pairs are defined, where values can be numeric values, strings, lists of values, and key-value pairs. Objects, state variables, a target state, and tables of constants are defined directly using key-value pairs and lists. In addition, `reduce: min` in line 35 of the domain file specifies to minimize the objective. In lines 3–12, state variables with names `U`, `i`, and `t` are defined, corresponding to U, i, and t, respectively. Tables `a`, `b`, and `c` in lines 13–26 correspond to a, b, and c, respectively. Numeric variables, constants, and the domain of the value function are either of `integer` or `continuous`, corresponding to integer and continuous values. In the definitions of transitions, base cases, state constraints, and the dual bound, an expression is written as a string following a LISP-like syntax. In an expression, a constant in a table is accessed by its name and indices, e.g., `(c i j)` in line 28 corresponds to a constant c_{ij}, where `i` is a state variable and `j` is a constant.

A quantifier `forall` is used to define the conjunction of conditions that are only different in element constants, which have the same object type, or are included in the same set variable. In lines 28–31, `forall` is used in the state constraint with the set variable `U`, and the element constant `j` is used in the expression, corresponding to $\forall j \in U$. Similarly, multiple transitions can be defined with `parameters`. In the example, to define transitions in the second line of Equation (3), only one definition parameterized by `j` in `U` is used in lines 38–46. Since `U` is a state variable, a precondition `(is_in j U)`, corresponding to $j \in U$, is assumed.

Note that implementations of DyPDL are not necessarily restricted to YAML-DyPDL, which adopts a style common in AI planning. Developing and improving interfaces for DyPDL is part of our future work. For example, a Python library will be useful for OR researchers and practitioners.

Listing 1: YAML-DyPDL domain file for TSPTW.

```
1  objects:
2    - customer
3  state_variables:
4    - name: U
5      type: set
6      object: customer
7    - name: i
8      type: element
9      object: customer
10   - name: t
11     type: integer
12     preference: less
13 tables:
14   - name: a
15     type: integer
16     args:
17       - customer
18   - name: b
19     type: integer
20     args:
21       - customer
22   - name: c
23     type: integer
24     args:
25       - customer
26       - customer
27 constraints:
28   - condition: (<= (+ t (c i j)) (b j))
29     forall:
30       - name: j
31         object: U
32 base_cases:
33   - - (is_empty U)
34     - (= i 0)
35 reduce: min
36 cost_type: integer
37 transitions:
38   - name: visit
39     parameters:
40       - name: j
41         object: U
42     effect:
43       U: (remove j U)
44       i: j
45       t: (max (+ t (c i j)) (a j))
46     cost: (+ cost (c i j))
47   - name: return
48     preconditions:
49       - (is_empty U)
50       - (!= i 0)
51     effect:
52       i: 0
53       t: (+ t (c i 0))
54     cost: (+ cost (c i 0))
55 dual_bounds:
56   - 0
```

CAASDy: A State Space Search Solver for DP

While various approaches can be applied to solve DyPDL, for our prototype solver, we adopt cost-algebraic heuristic search (Edelkamp, Jabbar, and Lafuente 2005). A DyPDL problem can be considered a graph search problem, where nodes correspond to states, edges correspond to transitions, and a solution corresponds to a path from S^0 to a base state. To compute the cost of a solution, we need to evaluate the cost expressions of the transitions backward from a base state to the target state. A naive approach is to perform recursion according to the recursive equations while memoizing all encountered states. However, if the cost expressions of transitions $\tau \in \mathcal{T}$ are in the form of $e_\tau(S) \times V(S[\![\tau]\!])$, where e_τ is a numeric expression and $\times$ is a binary operator, and the cost-algebra conditions are satisfied, the optimal solution can be computed by cost-algebraic search algorithms, generalized versions of shortest path algorithms such as A* (Hart, Nilsson, and Raphael 1968). For example, if the cost expression is in the form of $e_\tau(S) + V(S[\![\tau]\!])$ with $e_\tau(S) \geq 0$ for all S, and the objective is minimization, the optimal solution corresponds to the shortest path in a graph where the weight of edge $(S, S[\![\tau]\!])$ is $e_\tau(S)$. TSPTW is such an example since the cost expression of each transition is $c_{ij} + V(S[\![\tau]\!])$. In addition, a minimization problem with cost expressions in the form of $\max\{e_\tau(S), V(S[\![\tau]\!])\}$ with $e_\tau(S) \geq 0$ also satisfies the property of cost-algebra, as proved in the appendix

Listing 2: YAML-DyPDL problem file for TSPTW.

```
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,
    }
```

Algorithm 1: Cost-Algebraic A* for DyPDL

1: $g(S^0) \leftarrow 0, f(S^0) \leftarrow h(S^0), O, G \leftarrow \{S^0\}$.
2: **while** $O \neq \emptyset$ **do**
3: Let $S \in \arg\min_{S \in O} f(S)$.
4: $O \leftarrow O \setminus \{S\}$.
5: **if** $\exists B \in \mathcal{B}, S \models B$ **then**
6: **return** $g(S)$.
7: **for all** $\tau \in \mathcal{T}(S)$ **do**
8: **if** $\forall c \in \mathcal{C}, S[\![\tau]\!] \models c$ **then**
9: $g^\tau \leftarrow g(S) \times e_\tau(S)$.
10: **if** $\nexists S' \in G, S[\![\tau]\!] \preceq S' \wedge g^\tau \geq g(S')$ **then**
11: $g(S[\![\tau]\!]) \leftarrow g^\tau, f(S[\![\tau]\!]) \leftarrow g^\tau \times h(S[\![\tau]\!])$.
12: $G \leftarrow G \cup \{S[\![\tau]\!]\}, O \leftarrow O \cup \{S[\![\tau]\!]\}$.
13: **return** ∞.

(Kuroiwa and Beck 2023).

We adopt the cost-algebraic version of A* (Edelkamp, Jabbar, and Lafuente 2005) and name our solver Cost-Algebraic A* Solver for DyPDL (CAASDy). To solve a problem optimally, A* uses an admissible heuristic function, which computes a lower bound of the shortest path cost from a node. Cost-algebraic A* also uses a heuristic function, which computes a lower (upper) bound for minimization (maximization). In CAASDy, we do not use any hand-coded heuristic function. Instead, CAASDy just uses the dual bound h defined by a user in a DyPDL model as a heuristic function. In TSPTW, the trivial lower bound $V(S) \geq 0$ is defined in Inequality (5), so $h(S) = 0$ for all S is used as a heuristic function.

We show the pseudo-code of CAASDy in Algorithm 1. While we focus on minimization in the pseudo-code, it can be easily adapted to maximization as long as the cost-algebra conditions are satisfied. For each state S, the path cost from the target state $g(S)$ (the g-value), the heuristic value $h(S)$ (the h-value), and the priority $f(S) = g(S) \times h(S)$ (the f-value) are maintained. Recall that $\times$ is the binary operator used in the cost expressions satisfying the cost-algebra conditions, e.g., $+$ and $\max$. The open list O is the set of candidate states to search, and G stores all generated states. In line 4, a state S minimizing $f(S)$ is removed from O. We select the state minimizing $h(S)$ if there are multiple candidates. As discussed above, h is defined as the dual bound in a DyPDL model. If there are multiple states with the same $f(S)$ and $h(S)$, the tie-breaking depends on the binary heap implementation of O. In line 8, a state is pruned if a state constraint is not satisfied. As in line 10, a state is inserted into O only if there is no dominating state having an equal or smaller g-value in G. In the implementation, we maintain a hash table, where a key is the state variable values excluding resource variables, and a value is a list of states and their g-values. For each generated state, we check if its key exists in the hash table. If it exists, we compare the g-values and the resource variables of the state with each state in the list.

DP Models for Combinatorial Optimization

We present DP models for combinatorial optimization problems that can be represented in DyPDL. While we show recursive equations because they are succinct and easy to understand, the DyPDL representations and YAML-DyPDL files are provided in the appendix.

CVRP In a capacitated vehicle routing problem (CVRP), customers $N = \{0, ..., n\}$, where 0 is the depot, are given, and each customer $i \in N \setminus \{0\}$ has the demand d_i. 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. Visiting customer j from i incurs the travel time $c_{ij} \geq 0$, and the objective is to minimize the total travel time.

We formulate the DP model based on the giant-tour representation (Gromicho et al. 2012). 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 is visited by the current vehicle or a new vehicle. When a new vehicle is used, the customer is visited via the depot, l is reset, and k is increased. Let $U' = \{j \in U \mid l + d_j \leq q\}$ be the set of customers that can be visited next by the current vehicle, and $c'_{ij} = c_{i0} + c_{0j}$ be a numeric constant representing the travel time from i to j via the depot.

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

$$V(\emptyset, i, l, k) = c_{i0}$$

$$V(U, i, l, k) = \min \begin{cases} \min_{j \in U'} c_{ij} + V(U \setminus \{j\}, j, l + d_j, k) \\ \min_{j \in U} c'_{ij} + V(U \setminus \{j\}, j, d_j, k + 1) \end{cases} \quad \text{if } k < m$$

$$V(U, i, l, k) = \min_{j \in U'} c_{ij} + V(U \setminus \{j\}, j, l + d_j, k) \quad \text{if } k = m$$

$$V(U, i, l, k) \leq V(U, i, l', k') \quad \text{if } l \leq l' \wedge k \leq k'$$

$$V(U, i, l, k) \geq 0.$$

SALBP-1 In a simple assembly line balancing problem (SALBP), tasks $N = \{0, ..., n-1\}$ are given, and each task i has processing time t_i and predecessors $P_i \subset N$. A solution assigns tasks to a totally ordered set of stations so that the sum of processing times at each station does not exceed

the cycle time c, and all tasks in P_i are scheduled in the same station as i or an earlier station. We focus on minimizing the number of stations, which is called SALBP-1, for which branch-bound-and-remember (BB&R), a state space search algorithm, is a state-of-the-art exact method (Morrison, Sewell, and Jacobson 2014). We formulate a DP model inspired by BB&R. Let U be a set variable representing unscheduled tasks and r be a numeric variable representing the remaining time in the current station. Since having more remaining time leads to a better solution if the sets of unscheduled tasks are the same, r is a resource variable where more is preferred. Let $U' = \{i \in U \mid P_i \cap U = \emptyset \wedge r \geq t_i\}$ be tasks that can be assigned to the current station. At each step, one task is assigned to the current station from U', or a new station is opened when $U' = \emptyset$, which is called the maximal load pruning rule in the literature.

The model has a dual bound based on lower bounds of the bin packing problem obtained by ignoring predecessors (see the appendix). We define tables of numeric constants w^2, w'^2, and w^3 indexed by a task i, whose values depend on t_i.

t_i	$(0, c/2)$	$c/2$	$(c/2, c]$
w_i^2	0	0	1
$w_i'^2$	0	1/2	0

t_i	$(0, c/3)$	$c/3$	$(c/3, c/2)$	$2c/3$	$(2c/3, c]$
w_i^3	0	1/3	1/2	2/3	1

In addition, we use an 'if-then-else' expression l^2, which returns 1 if $r \geq c/2$ and 0 otherwise. Similarly, an expression l^3 returns 1 if $r \geq c/3$ and 0 otherwise. In YAML-DyPDL, l^2 is written as `(if (>= r (/ c 2.0)) 1 0)`.

compute $V(N, 0)$

$$V(U,r) = \begin{cases} 0 & \text{if } U = \emptyset \\ \min_{i \in U'} V(U \setminus \{i\}, r - t_i) & \text{if } U' \neq \emptyset \\ 1 + V(U, c) & \text{if } U' = \emptyset \end{cases}$$

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

$$V(U,r) \geq \max \begin{cases} \lceil (\sum_{i \in U} t_i - r)/c \rceil \\ \sum_{i \in U} w_i^2 + \lceil \sum_{i \in U} w_i'^2 \rceil - l^2 \\ \lceil \sum_{i \in U} w_i^3 \rceil - l^3 \end{cases}$$

In this model the sum of constants in a table, e.g., $\sum_{i \in U} t_i$, is used. In YAML-DyPDL, it is expressed as `(sum t U)`.

Bin Packing A bin packing problem is the same as SALBP-1 except that a task has no predecessors. We call a task an item and a station a bin and pack an item in a bin instead of assigning a task to a station. We adapt the DP model for SALBP-1 to bin packing. In addition to U and r, the model has 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, $U^1 = \{i \in U \mid r \geq t_i \wedge i + 1 \geq k\}$ represents items that can be packed in the current bin. When $U^1 = \emptyset$, then a new bin is opened, and any item in $U^2 = \{i \in U \mid i \geq k\}$ can be packed. The model also breaks symmetry here by selecting an arbitrary item in U^2, implemented as a forced transition.

compute $V(N, 0, 0)$

$$V(U,r,k) = \begin{cases} 0 & \text{if } U = \emptyset \\ \min_{i \in U^1} V(U \setminus \{i\}, r - t_i, k) & \text{if } U^1 \neq \emptyset \\ 1 + V(U \setminus \{i\}, c - t_i, k + 1) & \text{if } \exists i \in U^2 \\ \infty & \text{otherwise} \end{cases}$$

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

$$V(U,r,k) \geq \max \begin{cases} \lceil (\sum_{i \in U} t_i - r)/c \rceil \\ \sum_{i \in U} w_i^2 + \lceil \sum_{i \in U} w_i'^2 \rceil - l^2 \\ \lceil \sum_{i \in U} w_i^3 \rceil - l^3 \end{cases}$$

MOSP In the minimization of open stacks problem (MOSP) (Yuen and Richardson 1995), customers $C = \{0, ..., n-1\}$ and products $P = \{0, ..., m-1\}$ are given, and each customer c orders products $P_c \subseteq P$. A solution is a sequence in which products are produced. 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.

For MOSP, customer search is a state-of-the-art exact method (Chu and Stuckey 2009). 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 DP 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 a set constant representing customers that order the same product as c.

compute $V(C, \emptyset)$

$$V(R,O) = \begin{cases} 0 & \text{if } R = \emptyset \\ \min_{c \in R} \max \begin{cases} V(R \setminus \{c\}, O \cup N_c) \\ |(O \cap R) \cup (N_c \setminus O)| \end{cases} \end{cases}$$

$$V(R,O) \geq 0$$

Graph-Clear In a graph-clear problem (Kolling and Carpin 2007), 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.

Previous work (Morin et al. 2018) developed a state-based formula as the basis for MIP and CP models, but no DP model was defined. Here, we propose such a model. A set variable C represents swept nodes, and one node in $\overline{C} = N \setminus C$ is swept at each step. Weights a_i and b_{ij} are defined as numeric constants, assuming that $b_{ij} = 0$ if $\{i, j\} \notin E$, and N is defined as a set constant.

compute $V(\emptyset)$

$$V(C) = \begin{cases} 0 & \text{if } C = N \\ \min_{c \in \overline{C}} \max \begin{cases} V(C \cup \{c\}) \\ a_c + \sum_{i \in N} b_{ci} + \sum_{i \in C} \sum_{j \in \overline{C} \setminus \{c\}} b_{ij} \end{cases} \end{cases}$$

$$V(C) \geq 0.$$

The model takes the sum of b_{ij} over all combinations of $i \in C$ and $j \in \overline{C} \setminus \{c\}$. In YAML-DyPDL, it is described as `(sum b C (remove c ~C))`.

Experimental Evaluation

We experimentally compare DP, MIP, and CP models for the problems presented above. Most MIP and CP models are from the literature though we present new models in the appendix when they achieve superior performance to the literature. We use CAASDy as a solver for all the DP models. We use Gurobi Optimizer 9.5.1 for MIP and CP Optimizer from CPLEX Optimization Studio 22.1.0 for CP. As CAASDy is not an anytime solver, i.e., the first found solution is the optimal solution, we evaluate the number of instances solved to optimality within time and memory limits.

We implement the YAML-DyPDL parser and CAASDy in Rust 1.62.1.[2] Problem instances are transformed from their benchmark format into YAML-DyPDL by a Python 3.10.4 script and passed to CAASDy. We also use Python 3.10.4 to implement MIP and CP models. We run all experiments on a machine running Ubuntu 22.04 with an Intel Core i7 11700 processor using GNU Parallel (Tange 2011). For each instance, we use a single thread with a 30-minute time and 8 GB memory limit. We show the results in Table 2. In the last row, we present the ratio of optimally solved instances in each problem class averaged over all problem classes. DP solves the largest ratio of instances on average. Overall problems, we observe that if DP fails to solve an instance, it is due to the memory limit.

TSPTW We use four benchmark sets, Dumas (Dumas et al. 1995), GDE (Gendreau et al. 1998), OT (Ohlmann and Thomas 2007), and AFG (Ascheuer 1995). In the DP model, as the travel time satisfies the triangle inequality, we replace c^*_{ij} with c_{ij}. For MIP, we use Formulation (1) proposed by Hungerländer and Truden (2018). When there are zero-cost edges, flow-based subtour elimination constraints (Gavish and Graves 1978) are added. We adapt a CP model for a single machine scheduling problem with time windows (Booth et al. 2016) to TSPTW, where an interval variable represents the time to visit a customer. We change the objective to the sum of travel costs and add a First constraint ensuring that the depot is visited first. DP solves more instances than MIP and CP benefiting from pruning based on time windows.

CVRP We use A, B, E, F, and P instances from CVRPLIB (Uchoa et al. 2017) because they have at most 100 customers. The travel time is symmetric in these instances. We use a MIP model proposed by Gadegaard and Lysgaard (2021) and a CP model proposed by Rabbouch, Saâdaoui, and Mraihi (2019). MIP solves more instances than DP. Since there is no efficient pruning method unlike TSPTW, CAASDy suffers from an increasing branching factor with the number of customers and quickly runs out of memory. CP does not solve any instances to optimality.

SALBP-1 We use the benchmark set proposed by Morrison, Sewell, and Jacobson (2014). For MIP, we use the NF4 formulation (Ritt and Costa 2018). We use a CP model proposed by Bukchin and Raviv (2018) but implement it using the global constraint Pack in CP Optimizer as it performs better than the original model (see the appendix). 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. DP is better than MIP and CP, especially in large instances.

Bin Packing We use instances in BPPLIB (Delorme, Iori, and Martello 2018) and the MIP model (Martello and Toth 1990) extended with inequalities ensuring that bins are used in order of index and item j is packed in the j-th bin or before as described in Delorme, Iori, and Martello (2016). 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 the appendix. CP solves more instances than MIP and DP except

TSPTW	MIP	CP	DP
Dumas (135)	121	36	**135**
GDE (130)	71	4	**77**
OT (25)	0	0	0
AFG (50)	33	7	**45**
Total (340)	225	47	**257**
CVRP	MIP	CP	DP
A, B, E, F, P (90)	**26**	0	4
SALBP-1	MIP	CP	DP
Small (525)	**525**	**525**	**525**
Medium (525)	**518**	501	509
Large (525)	318	404	**414**
Very large (525)	0	155	**204**
Total (2100)	1360	1585	**1652**
Bin Packing	MIP	CP	DP
Falkenauer U (80)	25	**36**	33
Falkenauer T (80)	37	**56**	27
Scholl 1 (720)	**605**	533	517
Scholl 2 (480)	354	**445**	335
Scholl 3 (10)	0	**1**	0
Wäscher (17)	2	**10**	**10**
Schwerin 1 (100)	80	**96**	0
Schwerin 2 (100)	54	**61**	0
Hard 28 (28)	**0**	**0**	**0**
Total (1615)	1157	**1238**	922
MOSP	MIP	CP	DP
Constraint Modelling Challenge (46)	41	**44**	**44**
SCOOP Project (24)	8	**23**	16
Faggioli and Bentivoglio (300)	130	**300**	298
Chu and Stuckey (200)	44	70	**125**
Total (570)	223	437	**483**
Graph-Clear	MIP	CP	DP
Planar (60)	16	1	**45**
Random (75)	8	3	**31**
Total (135)	24	4	**76**
Average ratio	0.48	0.41	**0.59**

Table 2: Number of instances solved to optimality. 'Average ratio' is the ratio of optimally solved instances in each problem class averaged over all problem classes.

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

for Scholl 1. Similar to CVRP, without the precedence constraints of SALBP-1, CAASDy suffers from a large branching factor and quickly runs out of memory.

MOSP We use instances in Constraint Modelling Challenge (Smith and Gent 2005), SCOOP Project, Faggioli and Bentivoglio (1998), and Chu and Stuckey (2009). The MIP and CP models are proposed by Martin, Yanasse, and Pinto (2021). From their two MIP models, we select MOSP-ILP-I as it solves more instances optimally in their paper. DP solves more instances than MIP and CP in the Chu and Stuckey problem set, which results in higher coverage in total.

Graph-Clear We generate instances using planar and random graphs in the same way as Morin et al. (2018), where the number of nodes in a graph is 20, 30, or 40. We use MIP and CP models proposed by Morin et al. (2018). From the two proposed CP models, we select CPN as it performs better in our setting. DP solves more instances than MIP and CP. While MIP and CP only solve instances with 20 nodes, DP solves all planar instances with 20 and 30 nodes, all random instances with 20 nodes, 5 out of 20 planar instances with 40 nodes, and 6 out of 25 random instances with 30 nodes.

Discussion

First, we compare DIDP with existing approaches to clarify its novelty. Then, we summarize the significance of DIDP.

Model-Based DP

Little work has considered DP as a domain-independent model-based approach. DP2PN2Solver (Lew and Mauch 2006) is a C++/Java style modeling language with an associated DP solver that explicitly enumerates all reachable states. Algebraic dynamic programming (ADP) (Giegerich and Meyer 2002) is a framework to formulate a DP model using context-free grammar that was originally designed for bioinformatics and limited to problems on strings. Although ADP has been extended to describe diverse DP models (zu Siederdissen, Prohaska, and Stadler 2015), it is focused on bioinformatics applications.

AI Planning

Except for MOSP and graph-clear, the DP models above can be formulated as numeric planning problems, but resource variables, forced transitions, and a dual bound cannot be modeled. We evaluated the PDDL models with numeric planners using A* with admissible heuristics (Kuroiwa, Shleyfman, and Beck 2022), but they did not show competitive performance as they are not designed for such problems. For MOSP, a classical planning model based on a different formulation was used in the International Planning Competition.[3] A state-of-the-art planner, SymBA* (Torralba, Linares López, and Borrajo 2016) outperforms MIP but not CP or DP. Picat (Zhou, Kjellerstrand, and Fruhman 2015) is a logic-based programming language providing a DP solver and an associated AI planning module. However, the solution method is restricted to backtracking, and it cannot model resource variables.

[3]https://ipc06.icaps-conference.org/deterministic/

Decision Diagram Solver

The existing work most similar to ours is ddo, a decision diagrams (DD) solver that uses DP as a modeling interface (Gillard, Schaus, and Coppé 2020). Ddo is not a generic DP solver as it requires a problem-specific merge operator for DD nodes in addition to a DP model. The merge operator is necessary for relaxed DDs and, consequently, dual bounds. Defining a dual bound is optional in DyPDL and is done in the language of the model, not of the solver (i.e., a merge operator is only relevant to a DD-based solver). Developing a domain-independent merge operator for ddo is an interesting direction for future work. Since ddo was previously used in TSPTW, we evaluate it in our setting. While CAASDy solves 257 instances, ddo solves 179 instances (see the appendix for details). We do not evaluate ddo in other problems as the merge operators are not defined by previous work.

Hadook (Gentzel, Michel, and van Hoeve 2020) is a modeling language for decision diagrams developed for constraint propagation in CP. Hadook is similar to DyPDL in that its formalism is based on a state transition system.

Significance and Impact

In summary, DIDP is novel in the following points: it considers DP as a model-based approach, separating modeling and solving, for combinatorial optimization; its modeling formalism, DyPDL, is explicitly designed to allow a user to incorporate implications of the problem definition, i.e., resource variables, forced transitions, state constraints, and a dual bound, in a DP model, following the standard in OR. Our prototype solver shows state-the-of-art performance in multiple problem classes, as shown in the experimental result, which supports the significance of DIDP. DIDP also bridges the gap between AI and OR communities: DIDP enables researchers in AI planning and heuristic search to apply their methods to OR problems.

Conclusion

We proposed Domain-Independent Dynamic Programming (DIDP), a new model-based paradigm for combinatorial optimization. We developed Dynamic Programming Description Language (DyPDL), a modeling language for DP, and Cost-Algebraic A* Solver for DyPDL (CAASDy), a prototype DyPDL solver. Our solver outperforms MIP and CP in multiple combinatorial optimization problems.

While we formulated diverse DP models with DyPDL, there is significant room for extensions. For example, dominance relationships and symmetry breaking based on other criteria may be desired for efficient DP models. Also, there is a significant opportunity to improve our solver using state space search methods that have been developed in AI planning and heuristic search over the past two decades. For example, while CAASDy uses a dual bound defined in a DP model as a heuristic function, most AI planners automatically compute heuristic functions. Adapting AI planning methods to obtain a dual bound is one of our future plans.

Acknowledgments

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

References

Ascheuer, N. 1995. *Hamiltonian Path Problems in the On-Line Optimization of Flexible Manufacturing Systems*. Ph.D. thesis, Technische Universität Berlin.

Booth, K. E.; Tran, T. T.; Nejat, G.; and Beck, J. C. 2016. Mixed-Integer and Constraint Programming Techniques for Mobile Robot Task Planning. *IEEE Robot. and Autom. Lett.*, 1(1): 500–507.

Bukchin, Y.; and Raviv, T. 2018. Constraint Programming for Solving Various Assembly Line Balancing Problems. *Omega*, 78: 57–68.

Chu, G.; and Stuckey, P. J. 2009. Minimizing the Maximum Number of Open Stacks by Customer Search. In *Proc. CP*, 242–257.

Delorme, M.; Iori, M.; and Martello, S. 2016. Bin Packing and Cutting Stock Problems: Mathematical Models and Exact Algorithms. *Eur. J. Oper. Res.*, 255(1): 1–20.

Delorme, M.; Iori, M.; and Martello, S. 2018. BPPLIB: a Library for Bin Packing and Cutting Stock Problems. *Optim. Lett.*, 12(2): 235–250.

Dumas, Y.; Desrosiers, J.; Gelinas, E.; and Solomon, M. M. 1995. An Optimal Algorithm for the Traveling Salesman Problem with Time Windows. *Oper. Res.*, 43(2): 367–371.

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

Faggioli, E.; and Bentivoglio, C. A. 1998. Heuristic and Exact Methods for the Cutting Sequencing Problem. *Eur. J. Oper. Res.*, 110: 564–575.

Fikes, R.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. In *Proc. IJCAI*, 608–620.

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

Gadegaard, S. L.; and Lysgaard, J. 2021. A Symmetry-Free Polynomial Formulation of the Capacitated Vehicle Routing Problem. *Discret. Appl. Math.*, 296: 179–192.

Gavish, B.; and Graves, S. C. 1978. The Travelling Salesman Problem and Related Problems. Operations Research Center Working Paper.

Gendreau, M.; Hertz, A.; Laporte, G.; and Stan, M. 1998. A Generalized Insertion Heuristic for the Traveling Salesman Problem with Time Windows. *Oper. Res.*, 46(3): 330–346.

Gentzel, R.; Michel, L.; and van Hoeve, W.-J. 2020. HADDOCK: A Language and Architecture for Decision Diagram Compilation. In *Proc. CP*, 531–547.

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

Giegerich, R.; and Meyer, C. 2002. Algebraic Dynamic Programming. In *Proc. AMAST*, 349–364.

Gillard, X.; Schaus, P.; and Coppé, V. 2020. Ddo, a Generic and Efficient Framework for MDD-Based Optimization. In *Proc. IJCAI*, 5243–5245.

Gromicho, J.; Hoorn, J. J. V.; Kok, A. L.; and Schutten, J. M. 2012. Restricted Dynamic Programming: A Flexible Framework for Solving Realistic VRPs. *Comput. Oper. Res.*, 39(5): 902–909.

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.

Hungerländer, P.; and Truden, C. 2018. Efficient and Easy-to-Implement Mixed-Integer Linear Programs for the Traveling Salesperson Problem with Time Windows. *Transp. Res. Proc.*, 30: 157–166.

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. 2023. Appendix for Domain-Independent Dynamic Programming: Generic State Space Search for Combinatorial Optimization. https://tidel.mie.utoronto.ca/pubs/Appendix_CAASDy_ICAPS23.pdf. Accessed: 2023-02-24.

Kuroiwa, R.; Shleyfman, A.; and Beck, J. C. 2022. LM-Cut Heuristics for Optimal Linear Numeric Planning. In *Proc. ICAPS*, 203–212.

Lew, A.; and Mauch, H. 2006. *Dynamic Programming: A Computational Tool*. Springer Berlin.

Martello, S.; and Toth, P. 1990. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc.

Martin, M.; Yanasse, H. H.; and Pinto, M. J. 2021. Mathematical Models for the Minimization of Open Stacks Problem. *Int. Trans. Oper. Res.*

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.

Ohlmann, J. W.; and Thomas, B. W. 2007. A Compressed-Annealing Heuristic for the Traveling Salesman Problem with Time Windows. *INFORMS J. Comput.*, 19(1): 80–90.

Rabbouch, B.; Saâdaoui, F.; and Mraihi, R. 2019. Constraint Programming Based Algorithm for Solving Large-Scale Vehicle Routing Problems. In *Proc. HAIS*, 526–539.

Ritt, M.; and Costa, A. M. 2018. Improved Integer Programming Models for Simple Assembly Line Balancing and Related Problems. *Int. Trans. Oper. Res.*, 25(4): 1345–1359.

Smith, B.; and Gent, I. 2005. Constraint modelling challenge report 2005. https://ipg.host.cs.st-andrews.ac.uk/challenge/. Accessed: 2023-02-24.

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

Torralba, A.; Linares López, C.; and Borrajo, D. 2016. Abstraction Heuristics for Symbolic Bidirectional Search. In *Proc. IJCAI*, 3272–3278.

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.

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.

Zhou, N.-F.; Kjellerstrand, H.; and Fruhman, J. 2015. *Constraint Solving and Planning with Picat*. Springer Cham.

zu Siederdissen, C. H.; Prohaska, S. J.; and Stadler, P. F. 2015. Algebraic Dynamic Programming over General Data Structures. *BMC Bioinform.*, 16(19).