


STRIPS2DyPDL: A Translator from Automated Planning Problems into Domain-Independent Dynamic Programming Problems

Dillon Z. Chen ✉ 

LAAS-CNRS, University of Toulouse, France

Abstract

Domain-Independent Dynamic Programming (DIDP) is a recently introduced model-based paradigm for solving combinatorial optimisation problems and is inspired by automated planning models. Automated planning refers to the problem of and methodologies associated with sequential decision making under formally specified models. The most commonly used and simplest model for specifying automated planning models is STRIPS. In this paper, we implement the tool **STRIPS2DyPDL**, a translation of STRIPS planning problems into Dynamic Programming Description Language (DyPDL) problems, the de facto formalism for defining DIDP models. Our aim with **STRIPS2DyPDL** is to provide additional benchmarks to spur the progress of DIDP solvers, and to experimentally evaluate the search performance of DIDP solvers with respect to classical planners. The tool is available at <https://tinyurl.com/4ubvm44r>

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Combinatorial Optimisation, Automated Planning, Domain-Independent Dynamic Programming, Planning Domain Definition Language

■ Introduction

Domain-Independent Dynamic Programming (DIDP) [14] is a recently introduced model-based paradigm for solving combinatorial optimisation (CO) problems whose models and solvers drew great inspiration from automated planning (AP) models and solvers. AP refers to the field of problem of and algorithms associated with sequential decision making under formally specified models. Indeed, at the core of a DIDP solver is heuristic search, a sound and complete algorithm for finite state spaces which has also been the core of most state-of-the-art AP solvers [2, 13, 12, 19, 21]. DIDP solvers have shown (cf. [14]) to be highly competitive with combinatorial optimisation approaches and have shown to outperform industry grade mixed integer programming and constraint programming solvers. Furthermore, experiments performed in [17] show that DIDP solvers also outperform state-of-the-art planners in CO problems modelled in the Planning Domain Definition Language (PDDL), the de facto formalism used for modelling AP problems [18, 9].

In this paper, we conversely study the performance of DIDP solvers for classical AP problems modelled in the Dynamic Programming Description Language (DyPDL), the de facto formalism used for modelling DIDP problems. We implement a straightforward translation of STRIPS, the most widely used fragment of PDDL, into PyPDL, via the pipeline identified in Figure 1. Namely, we leverage an existing translation from STRIPS into SAS+ [1], an alternative AP modelling language, and implement the translation from SAS+ into PyPDL described in [17, Thm. 1]. We conduct experiments to evaluate the performance of DyPDL solvers in their current state compared to PDDL planners on the standard International Planning Competition (IPC) benchmark suite. We identify which DIDP solvers perform best with respect to various planning metrics and identify strengths and room for improvement relative to more mature PDDL planners.



■ **Figure 1** The pipeline for translating STRIPS models for planning into DyPDL models.

■ Background

We begin by defining a classical planning problem as a state transition model and representations used for modelling such problems: STRIPS, SAS+, and DyPDL. Table 1 illustrates the analogies between model components. All introduced sets are finite unless otherwise specified. Let 2^X denote the powerset of a set X . Let \mathbb{N} denote the set of natural numbers including zero and $\mathbb{R}_{\geq 0}$ the set of nonnegative real numbers.

Classical Planning Problem We follow the notation of a classical planning problem by Geffner and Bonet [6], with the exclusion of action costs¹. A classical planning problem is a deterministic, fully-observable state transition model with a specified initial state and a set of goal states. A solution for a classical planning problem is a plan, a finite sequence of applicable actions that progresses the initial state to a goal state.

► **Definition 1** (Classical Planning Problem). *A classical planning problem is a tuple $\mathcal{P} = \langle \mathcal{S}, \mathcal{A}, f, s_0, \mathcal{G} \rangle$ where the components are defined as follows.*

- \mathcal{S} is the state space, a set of states.
- \mathcal{A} is a set of actions.
- $f : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S} \cup \{\perp\}$ is a transition function where $f(s, a) = s' \neq \perp$ denotes that the action a is applicable in the state s where s' represents the successor of a in s , and otherwise $f(s, a) = \perp$ denotes that a is not applicable in s .
- $s_0 \in \mathcal{S}$ is the initial state.
- $\mathcal{G} \subseteq \mathcal{S}$ is a non-empty set of goal states.

► **Definition 2** (Plan). *A plan \vec{a} for a classical planning problem is a finite sequence of actions a_0, \dots, a_n such that $s_{i+1} = f(s_i, a_i) \neq \perp$ for $i = 0, \dots, n$ and $s_{n+1} \in \mathcal{G}$. The length of a plan is equal to its number of actions. A plan is optimal if it has the minimal length among all plans for a problem.*

In the three classical planning problem models that follow, we require defining the state space, action applicability, state successors, and goal states.

STRIPS STRIPS [4] is the simplest (ground) fragment of PDDL where states, goals and action preconditions are represented as single facts or conjunctions of facts. The representation of states as powersets of facts leads to compact planning representations that make planning over STRIPS models PSPACE-complete [3].

► **Definition 3** (STRIPS model). *A STRIPS model is a tuple $\langle F, O, I, G \rangle$ where the components are defined as follows.*

¹ The reasoning for excluding action costs is because STRIPS, the language we use to model planning problems, has uniform action costs, i.e. each action has a cost 1. We also exclude the presentation of action costs of all subsequent models for ease of presentation. However, our implementation and experiments support ground action costs.

- F is a set of propositions. A state s in STRIPS is a subset of F where under the closed world assumption, a proposition not in a state is presumed false. Thus, the state space of the model is defined by $\mathcal{S} = 2^F$.
- O is a set of actions of the form $a = \langle \text{pre}(a), \text{add}(a), \text{del}(a) \rangle$ where $\text{pre}(a)$, $\text{add}(a)$, $\text{del}(a) \subseteq F$ denote the preconditions, add effects, and delete effects of the action, respectively.
 - An action a is applicable in a state s if $s \supseteq \text{pre}(a)$.
 - The successor of an action a applicable in a state s is given by $s' = (s \setminus \text{del}(a)) \cup \text{add}(a)$.
- $I \subseteq F$ is the initial state.
- $G \subseteq F$ is the goal condition. A state s is a goal state if it contains the goal condition, i.e. $s \supseteq G$.

SAS+ SAS+ [1] can be viewed as an extension of STRIPS where states are represented no longer via propositional atoms but with multi-valued state variables. The original SAS+ definition does not fully specify the initial state but we will do so here following the classical planning problem formalism. SAS+ is in turn a special case of the Finite Domain Representation (FDR) [10, 11] which extends SAS+ with conditional effects and state axioms.

► **Definition 4** (SAS+ model). A SAS+ model is a tuple $\langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ where the components are defined as follows.

- $\mathcal{V} = \{v_1, \dots, v_n\}$ is a set of state variables where each $v_i \in \mathcal{V}$ has a domain \mathcal{D}_v consisting of a set of elements. Let $\mathcal{D}_v^+ = \mathcal{D}_v \cup \{\perp\}$ where \perp represents an undefined value. Then we can define the state space $\mathcal{S} = \times_{i=1}^n \mathcal{D}_{v_i}$ and partial state space $\mathcal{S}^+ = \times_{i=1}^n \mathcal{D}_{v_i}^+$, noting that $\mathcal{S} \subset \mathcal{S}^+$. Given a (partial) state $s \in \mathcal{S}^+$, denote $s[v]$ the value associated with v in s .
- \mathcal{O} is a set of operators (i.e. actions) of the form $o = \langle \text{pre}(o), \text{eff}(o) \rangle$ where $\text{pre}(o), \text{eff}(o) \in \mathcal{S}^+$ denote the precondition and effect of the operator, respectively.
 - An operator o is applicable in a state s if $s[v] = \text{pre}(o)[v]$ for all $v \in \mathcal{V}$ where $\text{pre}(o)[v] \neq \perp$.
 - The successor of an operator o applicable in a state s is the state s' where $s'[v] = s[v]$ if $\text{eff}(o)[v] = \perp$ and $s'[v] = \text{eff}(o)[v]$ otherwise.
- $s_0 \in \mathcal{S}$ is the initial state.
- $s_* \in \mathcal{S}^+$ is the goal condition. A state s is a goal state if $s[v] = s_*[v]$ for all $v \in \mathcal{V}$ where $s_*[v] \neq \perp$.

DyPDL DyPDL [14, 17] is the formalism used to encode DIDP models. It is inspired by the factoring of variables in STRIPS and SAS+ for encoding state transition models, and is presented in a way that reflects the recursive nature of dynamic programming and the Bellman backup equations. Specifically, DyPDL names what is commonly referred to as the initial state in PDDL planning as the ‘target state’, and the goal condition as the ‘base case’. We omit any definitions referring to costs due to the aforementioned reason that the theoretical translation from STRIPS does not require handling action costs.

► **Definition 5** (DyPDL model). A DyPDL model is a tuple $\langle \mathcal{V}, \mathcal{T}, S^0, \mathcal{B} \rangle$ where the components are defined as follows.

- $\mathcal{V} = \{v_1, \dots, v_n\}$ is a set of state variables that consists of element, set, and numeric variables. An element variable has domain \mathbb{N} , a set variable has domain $2^{\mathbb{N}}$, and a numeric variable has domain \mathbb{Q} (rational numbers). Let \mathcal{D}_v denote the domain of a state

■ **Table 1** STRIPS, SAS+ and DyPDL model components and the analogies between them.

STRIPS		SAS+		DyPDL	
propositions	F	state variables	\mathcal{V}	state variables	\mathcal{V}
actions	O	operators	\mathcal{O}	transitions	\mathcal{T}
initial state	I	inital state	s_0	target state	S^0
goal condition	G	goal condition	s_*	base case	\mathcal{B}

variables and let $\mathcal{D}_v^+ = \mathcal{D}_v \cup \{\perp\}$ where \perp represents an undefined value. We define the state space by $\mathcal{S} = \times_{i=1}^n \mathcal{D}_{v_i}$ and partial state space by $\mathcal{S}^+ = \times_{i=1}^n \mathcal{D}_{v_i}^+$. Given a (partial) state $s \in \mathcal{S}^+$, denote $s[v]$ the value associated with v in s .

- \mathcal{T} is a set of transitions (i.e. actions) of the form $\tau = \langle \text{eff}_\tau, \text{pre}_\tau \rangle$ where $\text{eff}_\tau \in \mathcal{S}^+$ and pre_τ is a set of conditions which are functions of the form $c : \mathcal{S} \rightarrow \{\top, \perp\}$.
 - A transition τ is applicable in a state s if $c(s) = \top$ for all $c \in \text{pre}_\tau$.
 - The successor of a transition τ in a state s is the state s' where $s'[v] = s[v]$ if $\text{eff}_\tau[v] = \perp$ and $s'[v] = \text{eff}_\tau[v]$ otherwise.
- $S^0 \in \mathcal{S}$ is the target state.
- \mathcal{B} is the base case which is a set of conditions. A state s is a base state (i.e. goal state) if $c(s) = \top$ for all $c \in \mathcal{B}$.

■ STRIPS2DyPDL

STRIPS2DyPDL involves a 2 step process of translating classical planning problems represented in STRIPS in DyPDL models as illustrated in Figure 1. The first step involves translating STRIPS (Definition 3) into SAS+ (Definition 4), while the second involves translating SAS+ into DyPDL (Definition 5). Both steps are indeed already described in the literature with the translation from STRIPS to SAS+² described in [11], and the latter SAS+ into DyPDL translation described implicitly in [17, Thm. 1].

Translating from STRIPS to SAS+ We note that there exists a naive translation from STRIPS to SAS+ by simply converting each proposition in F into a binary state variable and similarly for action preconditions, effects, and goals. However, this translation does not leverage the power of mutexes induced by SAS+ representations, whereas the translation by Helmert [11] computes invariants in order to encode mutexes into the multi-valued SAS+ state variables alongside other optimisations such as pruning of irrelevant propositions and actions for reducing the size of the model. Notably, Helmert’s translation has an existing and stable implementation that is used by many modern automated planners.

Translating from SAS+ to DyPDL The translation of SAS+ into DyPDL is straightforward as DyPDL states extend SAS+ states via the introduction of numeric and set variables in \mathcal{V} , as opposed to just finite-valued variables. Similarly, DyPDL action preconditions, goal conditions, and action effects generalise the same notions which constitute value assignments

² More specifically, we mean the *lifted* STRIPS fragment of PDDL, where propositions have additional attributes such as predicate and object type information. For ease of presentation, we presented propositional STRIPS where each proposition has no additional attributes.

to variables in SAS+. Indeed, Theorem 1 in [17] highlights this fact where numeric planning (SAS+ extended with numeric variables) can be translated into DyPDL. This work implements this direct translation.

■ Experiments

Benchmarks We collect STRIPS planning problems from the International Planning Competitions (IPC) 1998–2023 and translate them into SAS+ [11] with the translator in Fast Downward 24.06 [10]. We keep the problems that could be translated within 30 minutes and 8GB of memory on a cluster with Intel Xeon 3.2 GHz CPU cores, and with an output file size of at most 128kB for a total of 660 SAS+ problems. The SAS+ files are used as input for all planning approaches.

Approaches We evaluate the performance of DIDP solvers on classical STRIPS planning problems translated into DyPDL models described in the previous section compared to STRIPS baselines and planners. The DIDP solvers we experiment with are **acps**, **apps**, **caasdy**, **cabs** [16] and **lnbs** [15]. The experimented STRIPS planners are blind A* search (**blind**) and anytime LAMA (**lama**) [19] implemented in Fast Downward 24.06.1³, the agile configuration of decoupled state space search (**decstar**⁴) [7, 8], A* search with saturated cost partitioning heuristics (**scorpion**⁵) [21, 20], and optimal planning with symbolic, bidirectional search (**symk**⁶) [23, 22].

The following planners return provably optimal solutions, i.e. they terminate once an optimal plan is returned and proven to be optimal: **acps**, **apps**, **caasdy**, **cabs**, **lnbs**, **blind**, **scorpion**, and **symk**; and the following planners are anytime planners, i.e. they continue to return plans of increasing quality over time: **acps**, **apps**, **cabs**, **lnbs**, and **lama**.

Evaluation All planning experiments are run on the same cluster and resource limits as the translation process. Tables 2–4 summarise various planning performance metrics where **bold cells** indicate the best score among DIDP solvers for each row, while underlined cells indicate the best score among all approaches for each row. The metrics are described as follows conditioned on an approach:

- **IPC satisficing track score (Table 2):** the sum of the function which returns for each problem \mathcal{P} C^*/C where C^* is the lowest plan cost over all approaches, and C the plan cost of the planner if \mathcal{P} is solved, and 0 otherwise.
For example, if **acps** returned a plan with cost 3 for a problem **apps** a plan with cost 4, **lama** a plan with cost 4, **decstar** a plan with cost 5, and all other approaches failed to find a plan, then **acps** gets a score of 1, **apps** and **lama** a score of $\frac{3}{4}$, **decstar** a score of $\frac{3}{5}$, and all other approaches a score of 0 for the problem. Results for **decstar** are omitted as the configuration we experiment with is not focused on solution quality.
- **IPC agile track score (Table 3):** the sum of the function which returns for each problem \mathcal{P} 1 if \mathcal{P} is solved in less than a second, $1 - \frac{\log(t)}{\log(300)}$ if \mathcal{P} is solved within 300s and 8GB mem, and 0 otherwise. Results for **scorpion** and **symk** are omitted as these planners are not designed for agile planning.

³ Available at <https://github.com/aibasel/downward/releases/tag/release-24.06.1>

⁴ Available at <https://github.com/ipc2023-classical/planner15>

⁵ Available at <https://github.com/jendrikseipp/scorpion>

⁶ Available at <https://github.com/speckdavid/symk>

■ **Table 2** IPC satisficing track scores. Evaluates *quality* of generated solutions.

Domain	acps	apps	caasdy	cabs	lnbs	blind-fd	decstar	lama	scorpion	symk
blocks	18.0	15.1	18.0	18.0	15.0	21.0	–	<u>34.6</u>	28.0	31.0
depot	3.0	3.0	4.0	3.0	2.1	6.0	–	<u>9.0</u>	<u>9.0</u>	6.0
driverlog	4.0	4.0	6.0	5.0	3.0	8.0	–	<u>14.0</u>	<u>14.0</u>	<u>14.0</u>
elevators	0.0	0.0	1.0	1.0	0.0	2.0	–	<u>9.6</u>	3.0	3.0
floortile	0.0	0.0	0.0	0.9	1.0	0.0	–	8.0	22.0	<u>27.0</u>
freecell	2.0	2.0	2.0	2.0	2.0	2.0	–	<u>2.0</u>	<u>2.0</u>	<u>2.0</u>
gripper	16.7	16.9	7.0	17.5	17.3	8.0	–	<u>20.0</u>	7.0	19.0
hiking	2.0	2.0	2.0	2.0	2.0	2.0	–	<u>2.0</u>	<u>2.0</u>	<u>2.0</u>
logistics	12.0	11.9	12.0	12.0	7.6	13.0	–	<u>35.2</u>	35.0	26.0
maintenance	0.0	0.0	0.0	0.0	0.0	0.0	–	<u>0.0</u>	<u>0.0</u>	<u>0.0</u>
miconic	75.2	86.1	45.0	72.3	66.3	55.0	–	<u>114.7</u>	114.0	111.0
movie	30.0	30.0	30.0	30.0	30.0	<u>30.0</u>	–	<u>30.0</u>	29.0	<u>30.0</u>
mprime	3.8	3.8	4.0	4.0	4.0	4.0	–	<u>4.0</u>	<u>4.0</u>	<u>4.0</u>
mystery	5.0	5.0	5.0	5.0	5.0	5.0	–	<u>5.0</u>	<u>5.0</u>	<u>5.0</u>
nomystery	3.0	3.0	3.0	3.0	2.8	3.0	–	<u>3.0</u>	<u>3.0</u>	<u>3.0</u>
pegsol	48.0	47.8	44.0	48.0	48.0	46.0	–	49.5	<u>50.0</u>	49.0
pipesworld	14.6	13.3	16.0	14.0	9.7	<u>18.0</u>	–	<u>18.0</u>	<u>18.0</u>	<u>18.0</u>
rovers	10.1	7.4	5.0	7.7	8.0	6.0	–	<u>16.7</u>	13.0	14.0
satellite	4.0	4.0	4.0	4.6	4.0	6.0	–	<u>9.7</u>	9.0	9.0
scanalyzer	6.0	6.2	9.0	8.3	6.0	<u>9.0</u>	–	<u>9.0</u>	<u>9.0</u>	<u>9.0</u>
schedule	15.3	14.4	12.0	19.8	29.0	15.0	–	<u>107.4</u>	40.0	38.0
sokoban	14.0	15.0	12.0	15.8	16.9	25.0	–	<u>42.7</u>	41.0	17.0
storage	12.2	12.0	13.0	12.2	11.0	<u>15.0</u>	–	<u>15.0</u>	<u>15.0</u>	<u>15.0</u>
termes	0.0	0.0	0.0	0.0	0.0	0.0	–	<u>10.8</u>	0.0	3.0
thoughtful	1.3	1.4	1.0	1.0	0.9	<u>2.0</u>	–	<u>2.0</u>	<u>2.0</u>	<u>2.0</u>
tpp	10.1	8.5	5.0	8.6	9.2	6.0	–	11.0	<u>12.0</u>	8.0
transport	5.6	5.2	6.0	6.0	3.8	6.0	–	<u>6.0</u>	<u>6.0</u>	<u>6.0</u>
visitall	1.7	1.1	0.0	0.3	1.3	0.0	–	<u>2.9</u>	1.0	0.0
woodworking	5.0	5.0	5.0	5.2	4.3	<u>7.0</u>	–	<u>7.0</u>	<u>7.0</u>	<u>7.0</u>
zenotravel	7.0	5.1	7.0	7.0	4.2	8.0	–	10.8	<u>11.0</u>	<u>11.0</u>
Total	329.6	329.3	278.0	333.9	314.5	328.0	–	<u>609.5</u>	511.0	489.0
Best in Domain (DIDP)	14	9	18	18	11	–	–	–	–	–
Best in Domain (Overall)	6	6	9	8	6	13	–	<u>27</u>	17	16

- **IPC optimal track score (Table 4):** the sum of the function which returns for each problem \mathcal{P} 1 if the problem is solved optimally and 0 otherwise. Results for **decstar** and **lama** are omitted as they are not provably optimal solvers.

Results

We present our results by answering the following questions with regards to various planning metrics for solution quality, planning speed, and provably optimal planning performance.

(1) Which DIDP solver performs best for *quality-focused* planning, and how does it compare to PDDL planners? We refer the reader to Table 2 for IPC satisficing score scores to answer this question. In terms of total score, complete anytime beam search (**cabs**) performs best out of all DIDP solvers, which reflects the observation that **cabs** does best in CO problems out of existing DIDP solvers. However, the only DIDP solver that is not anytime, **caasdy**, despite having the worse total score is the tied best DIDP solver on the most domains. This is because **caasdy** gets no score unless it solves a problem (optimally) in which case it gets 1. Thus, in domains where **caasdy** is able to solve problems it generally achieves higher scores compared to other DIDP solvers which find solutions faster but prove optimality slower. Furthermore, **cabs** achieves a greater score than **blind**, a blind, forward search planner. This suggests that DIDP implementations are quite efficient despite supporting different modelling features and being very recently introduced. However, they

■ **Table 3** IPC agile track scores. Evaluates *speed* of generating solutions.

Domain	acps	apps	caasdy	cabs	lnbs	blind-fd	decstar	lama	scorpion	symk
blocks	14.9	14.2	15.3	14.6	15.8	16.8	<u>35.0</u>	34.9	—	—
depot	3.0	3.0	2.4	3.3	3.5	3.0	8.9	<u>9.0</u>	—	—
driverlog	3.9	3.8	4.2	4.0	4.3	5.4	<u>14.0</u>	<u>14.0</u>	—	—
elevators	0.7	0.7	0.6	0.6	0.6	1.1	<u>10.0</u>	<u>10.0</u>	—	—
floortile	0.0	0.0	0.0	0.1	0.5	0.0	<u>6.9</u>	3.3	—	—
freecell	2.0	2.0	1.7	2.0	2.0	2.0	<u>2.0</u>	<u>2.0</u>	—	—
gripper	20.0	20.0	5.7	20.0	20.0	6.4	<u>20.0</u>	<u>20.0</u>	—	—
hiking	1.4	1.4	1.6	1.3	1.2	2.0	<u>2.0</u>	<u>2.0</u>	—	—
logistics	11.9	11.8	11.0	11.1	11.5	12.0	<u>36.0</u>	<u>36.0</u>	—	—
maintenance	0.0	0.0	0.0	0.0	0.0	0.0	<u>4.7</u>	0.0	—	—
miconic	111.8	113.4	39.3	82.2	86.0	43.2	<u>115.0</u>	<u>115.0</u>	—	—
movie	30.0	30.0	30.0	30.0	30.0	<u>30.0</u>	<u>30.0</u>	<u>30.0</u>	—	—
mprime	<u>4.0</u>	<u>4.0</u>	3.9	3.5	3.6	<u>4.0</u>	<u>4.0</u>	<u>4.0</u>	—	—
mystery	5.0	5.0	5.0	4.8	5.0	<u>5.0</u>	<u>5.0</u>	<u>5.0</u>	—	—
nomystery	1.0	1.0	0.6	0.4	0.5	2.1	<u>3.0</u>	<u>3.0</u>	—	—
pegsol	47.8	47.7	37.0	46.8	47.6	41.3	47.2	<u>49.6</u>	—	—
pipesworld	17.5	17.4	13.3	16.9	17.0	15.9	<u>18.0</u>	<u>18.0</u>	—	—
rovers	16.0	16.0	4.2	15.6	16.0	4.5	<u>17.0</u>	<u>17.0</u>	—	—
satellite	5.0	4.8	3.7	5.6	5.9	4.1	<u>10.0</u>	<u>10.0</u>	—	—
scanalyzer	7.4	7.8	6.0	7.3	7.3	6.8	<u>9.0</u>	<u>9.0</u>	—	—
schedule	26.7	21.6	9.9	22.5	31.0	11.9	107.0	<u>108.0</u>	—	—
sokoban	9.4	10.6	8.8	9.7	13.4	10.5	<u>33.1</u>	31.3	—	—
storage	14.3	14.6	11.7	12.8	12.9	13.2	<u>15.0</u>	<u>15.0</u>	—	—
termes	0.0	0.0	0.0	0.0	0.0	0.0	6.1	<u>7.3</u>	—	—
thoughtful	0.5	1.2	0.1	0.0	1.0	0.9	<u>2.0</u>	<u>2.0</u>	—	—
tpp	12.0	11.7	5.0	8.6	10.6	5.3	<u>12.0</u>	<u>12.0</u>	—	—
transport	5.3	5.0	4.4	5.4	5.7	5.5	<u>6.0</u>	<u>6.0</u>	—	—
visitall	2.9	2.9	0.0	2.4	3.0	0.0	2.7	<u>3.0</u>	—	—
woodworking	5.6	6.6	4.3	5.6	5.7	4.7	<u>7.0</u>	<u>7.0</u>	—	—
zenotravel	7.0	6.9	5.9	6.9	6.9	6.9	<u>11.0</u>	<u>11.0</u>	—	—
Total	386.9	385.1	235.6	343.9	368.6	264.5	<u>599.7</u>	594.4	—	—
Best in Domain (DIDP)	16	15	5	5	16	—	—	—	—	—
Best in Domain (Overall)	6	5	2	3	5	5	<u>26</u>	<u>26</u>	—	—

are outperformed by advanced planners which have access to strong domain-independent heuristics (*lama* and *scorpion*) and state representations (*symk*).

(2) Which DIDP solver performs best for *speed*-focused planning, and how does it compare to PDDL planners? We refer the reader to Table 3 for IPC agile scores to answer this question. In terms of number of domains a DIDP solver does best in, *acps* and *lnbs* are tied, while *acps* very marginally outperforms *lnbs* in total IPC agile score. Furthermore, the anytime DIDP solvers outperform *blind*. However, we note that solution quality is not a factor in the IPC agile score, such that anytime solvers which begin by trying to find any solution quickly unsurprisingly outperform non-anytime optimal solvers that have to prove optimality of their solutions (*caasdy* and *blind*). Again, advanced planners achieve higher scores as they have access to domain-independent heuristics (*lama*) and problem reformulations (*decstar*) neither of which DIDP solvers in their current state support.

(3) Which DIDP solver performs best for *provably optimal* planning, and how does it compare to PDDL planners? We refer the reader to Table 4 for IPC optimal scores to answer this question. Conversely to previous notes in (1) and (2), non-anytime optimal DIDP planners take longer to return a solution but generally prove optimality of solutions faster. This is reflected in the observation that the non-anytime optimal *caasdy* solver performs best in both total score and number of best placements per domain out of all DIDP solvers. However, we now notice a larger margin between DIDP solvers and the blind search

■ **Table 4** IPC optimal track scores. Evaluates speed of generating provably *optimal* solutions.

Domain	acps	apps	caasdy	cabs	lnbs	blind-fd	decstar	lana	scorpion	symk
blocks	18	15	18	18	15	21	—	—	28	31
depot	3	3	3	3	2	6	—	—	9	6
driverlog	4	4	5	5	3	8	—	—	14	14
elevators	0	0	0	0	0	2	—	—	3	3
floortile	0	0	0	0	0	0	—	—	22	27
freecell	2	2	2	2	2	2	—	—	2	2
gripper	7	6	7	7	5	8	—	—	7	19
hiking	2	2	2	2	2	2	—	—	2	2
logistics	12	11	12	12	6	13	—	—	35	26
maintenance	0	0	0	0	0	0	—	—	0	0
miconic	45	45	45	45	45	55	—	—	114	111
movie	30	30	30	30	30	30	—	—	29	30
mprime	1	1	1	1	1	4	—	—	4	4
mystery	2	2	2	2	2	5	—	—	5	5
nomystery	3	3	3	3	1	3	—	—	3	3
pegsol	42	42	42	42	42	46	—	—	50	49
pipesworld	14	12	14	14	6	18	—	—	18	18
rovers	4	4	4	4	4	6	—	—	13	14
satellite	4	4	4	4	4	6	—	—	9	9
scanalyzer	6	6	9	6	6	9	—	—	9	9
schedule	3	3	3	3	3	15	—	—	40	38
sokoban	10	10	10	10	9	25	—	—	41	17
storage	12	12	12	12	10	15	—	—	15	15
termes	0	0	0	0	0	0	—	—	0	3
thoughtful	0	0	0	0	0	2	—	—	2	2
tpp	5	5	5	5	5	6	—	—	12	8
transport	4	4	6	6	3	6	—	—	6	6
visitall	0	0	0	0	0	0	—	—	1	0
woodworking	5	4	5	5	4	7	—	—	7	7
zenotravel	7	5	7	7	4	8	—	—	11	11
Total	245	235	251	248	214	328	—	—	511	489
Best in Domain (DIDP)	27	21	31	29	17	—	—	—	—	—
Best in Domain (Overall)	5	5	7	6	4	13	—	—	25	22

planner **blind** as optimal planning most highlights the optimisation benefit that automated planners gain from solving their intended problems. This is because blind optimal planning requires generating all states with optimal cost from the initial state lower bounding the optimal plan cost as quickly as possible.

■ Conclusion

We introduce the **STRIPS2DyPDL** tool for translating automated planning problems modelled in STRIPS into Domain-Independent Dynamic Programming (DIDP) problems modelled in the Dynamic Programming Description Language (DyPDL). We conduct experiments to evaluate the best performing DIDP models under different automated planning metrics on a large suite of automated planning benchmarks. We find that current DIDP models, which use blind search without heuristic guidance, perform favourably compared to blind automated planners. This is despite the fact that DIDP supports an orthogonal set of modelling features, and that automated planners are several decades more mature and hence are more optimised for planning. We hope that benchmarks induced by **STRIPS2DyPDL** from planning benchmarks will spur the progress of DIDP solvers. Future work could include extending **STRIPS2DyPDL** to handle more expressive planning model features present in PDDL such as state axioms [24] via *state functions* introduced in DyPDL 0.9.0, and numeric state representations [5] via existing DyPDL features.

References

- 1 Christer Bäckström and Bernhard Nebel. Complexity results for SAS+ planning. *Comput. Intell.*, 11:625–656, 1995.
- 2 Blai Bonet and Hector Geffner. Planning as heuristic search. *Artif. Intell.*, 129(1-2):5–33, 2001.
- 3 Tom Bylander. The computational complexity of propositional STRIPS planning. *Artif. Intell.*, 69:165–204, 1994.
- 4 Richard Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, 2(3/4):189–208, 1971.
- 5 Maria Fox and Derek Long. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res.*, 20:61–124, 2003.
- 6 Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers, 2013.
- 7 Daniel Gnad and Jörg Hoffmann. Star-topology decoupled state space search. *Artif. Intell.*, 257:24–60, 2018.
- 8 Daniel Gnad, Álvaro Torralba, and Alexander Shleyfman. Decstar. International Planning Competition 2023 Classical Track Planner Abstracts, 2023.
- 9 Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise. *An Introduction to the Planning Domain Definition Language*. Morgan & Claypool Publishers, 2019.
- 10 Malte Helmert. The fast downward planning system. *J. Artif. Intell. Res.*, pages 191–246, 2006.
- 11 Malte Helmert. Concise finite-domain representations for PDDL planning tasks. *Artif. Intell.*, 173(5-6):503–535, 2009.
- 12 Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What’s the difference anyway? In *ICAPS*, 2009.
- 13 Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res.*, 14:253–302, 2001.
- 14 Ryo Kuroiwa and J. Christopher Beck. Domain-independent dynamic programming: Generic state space search for combinatorial optimization. In *ICAPS*, 2023.
- 15 Ryo Kuroiwa and J. Christopher Beck. Large neighborhood beam search for domain-independent dynamic programming. In *CP*, 2023.
- 16 Ryo Kuroiwa and J. Christopher Beck. Solving domain-independent dynamic programming problems with anytime heuristic search. In *ICAPS*, 2023.
- 17 Ryo Kuroiwa and J. Christopher Beck. Domain-independent dynamic programming. *CoRR*, abs/2401.13883, 2024.
- 18 Drew McDermott, Malik Ghallab, Adele E. Howe, Craig A. Knoblock, Ashwin Ram, Manuela M. Veloso, Daniel S. Weld, and David E. Wilkins. Pddl-the planning domain definition language. Technical report, 1998.
- 19 Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *J. Artif. Intell. Res.*, 39:127–177, 2010.
- 20 Jendrik Seipp. Scorpion 2023. International Planning Competition 2023 Classical Track Planner Abstracts, 2023.
- 21 Jendrik Seipp, Thomas Keller, and Malte Helmert. Saturated cost partitioning for optimal classical planning. *J. Artif. Intell. Res.*, 67:129–167, 2020.
- 22 David Speck. Symk – a versatile symbolic search planner. International Planning Competition 2023 Classical Track Planner Abstracts, 2023.
- 23 David Speck, Jendrik Seipp, and Álvaro Torralba. Symbolic search for cost-optimal planning with expressive model extensions. *J. Artif. Intell. Res.*, 82, 2025.
- 24 Sylvie Thiébaux, Jörg Hoffmann, and Bernhard Nebel. In defense of PDDL axioms. *Artif. Intell.*, 168(1-2):38–69, 2005.