Domain-Independent Instance Generation for Classical Planning

Claudia Grundke and Malte Helmert and Gabriele Röger
University of Basel
Basel, Switzerland
{claudia.grundke, malte.helmert, gabriele.roeger}@unibas.ch

Abstract

Learning-based planning systems learn domain-specific knowledge that helps them to solve unseen tasks from the same planning domain. For this purpose they require a diverse set of training instances. A recent proposal for formal specifications of planning domains allows us to exactly characterize which instances are legal for a domain. We automatically generate planning tasks from such formal specifications by means of a translation to answer set programming. We experimentally examine the scalability of the approach and the suitability for learning-based planning, following the setup of the learning track of the International Planning Competition.

1 Introduction

The Planning Domain Definition Language PDDL (McDermott et al. 1998; Fox and Long 2003) separates the specification of a planning task into a *PDDL domain* and an *instance* description. The PDDL domain defines the vocabulary and dynamics of the task, while the instance specifies the initial state and the goal. The PDDL domain file is typically shared among all tasks of an application domain.

As observed by Grundke, Röger, and Helmert (2024), PDDL domains overapproximate the set of planning tasks that a domain modeller intends to describe. For example, in the Blocksworld domain (Slaney and Thiébaux 2001), there are stacks of blocks encoded by a binary predicate *on*. With the corresponding PDDL domain, it is possible to have "cyclic" stacks of blocks (if the transitive closure of *on* is not irreflexive), which is naturally not the case in the physical world. In the same work Grundke et al. introduced a formalism for exactly characterizing planning domains, augmenting the PDDL domain with a logic program that determines whether a given initial state is legal for the domain.

Instance generation has always been important for the International Planning Competitions (IPCs) and recently gained further importance with the rise of learning-based planning systems. Most instance generators are domain-specific and create the tasks in a procedural manner, as can be seen[1] from the IPC 2023 learning track (Taitler et al. 2024). Notable exceptions use machine learning and an ad-hoc semi-declarative language for constraints (Núñez-Molina, Mesejo, and Fernández-Olivares 2024) or constraint programming (CP), augmenting the PDDL domain with additional CP constraints (Akgün et al. 2020). We build upon the formalism by Grundke, Röger, and Helmert (2024), generating instances for the domain by means of a compilation to answer set programming.

2 Background

Planning domains Due to the limited space, we do not fully introduce PDDL planning tasks but focus on the aspects relevant to this work. We go beyond the work by Grundke, Röger, and Helmert (2024) by extending their specification to *types*. Types are syntactic sugar that restricts action parameters, predicate parameters and the range of quantifiers to objects of a certain type. They are typically omitted in theoretical work but since we make a practical contribution, we support them directly. From the theoretical perspective, this means that we move from first-order logic to *order-sorted logic* (Socher-Ambrosius and Johann 1997): beyond the predicates and constant symbols there is a non-empty finite set $\mathcal{T}$ of type symbols with a partial order $\sqsubseteq$ on the types, specifying a type hierarchy. Following the practice in PDDL,[2] we assume that $\sqsubseteq$ is functional (each type has at most one immediate supertype) and has a unique root type $\text{OBJECT} \in \mathcal{T}$ that is a supertype of all types. The language restricts the parameters of all predicates and all terms to one of the types and each object o in the universe is assigned a type $\mathit{type}(o) \in \mathcal{T}$. Whenever the signature of the language requires a term of a certain type τ, an interpretation can map the term to any object o with $\mathit{type}(o) \sqsubseteq^* \tau$.

A planning task is defined over a function-free order-sorted signature $\langle \mathcal{T}, \sqsubseteq, \Sigma_{\mathcal{C},\mathcal{P}} \rangle$ with equality $=$, where $\mathcal{C}$ is a finite set of constant symbols and $\mathcal{P}$ is a finite set of predicate symbols P, each associated with a vector $\mathit{types}(P)$ of the same arity specifying the types of the arguments. A state of a planning task is a Herbrand structure for this signature and can alternatively be represented as a set of ground atoms.

The predicates are partitioned into a set $\mathcal{B}$ of *basic* predicates and a set $\mathcal{D}$ of *derived predicates*. Only the interpretation of the basic predicates may be affected by the *action* applications, whereas the interpretation of the predicates in $\mathcal{D}$ is derived by means of the *axioms* of the task:

[1] https://github.com/ipc2023-learning/benchmarks/

[2] Unfortunately the formal semantics of PDDL types are not fully defined and occasionally give rise to debate.

Definition 1 (PDDL axioms; adapted from Thiébaux, Hoffmann, and Nebel 2005)**.** *A* PDDL axiom *over order-sorted signature $\langle \mathcal{T}, \sqsubseteq, \Sigma_{\mathcal{C},\mathcal{P}} \rangle$ is a rule of the form $H(\bar{x}) \leftarrow B(\bar{x})$, where the* head $H(\bar{x})$ *is a single atom over $\langle \mathcal{T}, \sqsubseteq, \Sigma_{\mathcal{C},\mathcal{P}} \rangle$ with free variables $\bar{x}$ and the* body $B(\bar{x})$ *is a formula over $\langle \mathcal{T}, \sqsubseteq, \Sigma_{\mathcal{C},\mathcal{P}} \rangle$ with the same free variables $\bar{x}$.*

A set $\mathcal{A}$ of PDDL axioms is stratifiable *if there exists a partition (stratification) $\mathcal{P}_1, \ldots, \mathcal{P}_n$ of the predicate set $\mathcal{P}$ such that for each predicate $P_i \in \mathcal{P}_i$ and axiom $P_i(\bar{x}) \leftarrow B(\bar{x}) \in \mathcal{A}$:*

- *if $P_j \in \mathcal{P}_j$ appears in $B(\bar{x})$, then $j \leq i$, and*
- *if $P_j \in \mathcal{P}_j$ appears negated in the translation of $B(\bar{x})$ to negation normal form, then $j < i$.*

A *basic state* only interprets the basic predicates. For the computation of the *extended state* that interprets all predicates, consider the set G of ground axioms obtained by replacing each free variable in an axiom with a constant of the same type (or one of its subtypes). Let $G_{\mathcal{P}_i} \subseteq G$ be the set of all such axioms where the head predicate is from $\mathcal{P}_i$. The extension starts from the basic state, setting all ground atoms of derived predicates to false. For stratum $\mathcal{P}_1$, we compute the final interpretation of the predicates in $\mathcal{P}_1$ as a fixed point, making the head of an axiom in $G_{\mathcal{P}_1}$ true whenever the body is true under the current interpretation. Afterwards we continue analogously with $\mathcal{P}_2, \ldots, \mathcal{P}_n$.

A planning domain fixes the predicates and some constants of its instances, as well as the axioms and actions (this is already the case for a PDDL domain). For this work, it is not necessary to further introduce PDDL actions because they do not affect the legality of an instance for the domain. Grundke, Röger, and Helmert (2024) augment PDDL domains with a domain-wide first-order sentence that fixes the goal for all instances and additional axioms $\mathcal{A}_\mathrm{q}$ (with corresponding derived predicates $\mathcal{P}_\mathrm{q}$) that are used to assess whether a basic state is permitted as an initial state in an instance of the domain: it is legal iff a special predicate *legal*() is true in the extended state (using the domain axioms as well as the additional axioms).

To be able to express all properties that can be checked in polynomial time, Grundke et al. also add a linear order over the objects to the set of predicates. In their work, this linear order is expressed in terms of a successor relation. Our following definition deviates from their formalism in two aspects: it incorporates types as a built-in feature and represents the linear order directly with an ordering predicate $<$ rather than as a successor relation *succ*. The first difference is just syntactic sugar, and the second is also unobtrusive because axioms can define *succ* in terms of $<$ and vice versa.

Definition 2 (Planning domain)**.** *A* planning domain *is a tuple $\langle \mathcal{P}_\mathrm{t}, \mathcal{P}_\mathrm{q}, \mathcal{T}, \mathcal{C}, \mathcal{O}, \mathcal{A}_\mathrm{t}, \mathcal{A}_\mathrm{q}, \mathit{legal}, <, \mathcal{G} \rangle$, where*

- *$\mathcal{P}_\mathrm{t}$ is a finite set of predicate symbols that can be partitioned into a set of basic predicates $\mathcal{B}_\mathrm{t} \supseteq \{=\}$ and a set of derived predicates $\mathcal{D}_\mathrm{t}$,*
- *$\mathcal{P}_\mathrm{q}$ with $\mathcal{P}_\mathrm{q} \cap \mathcal{P}_\mathrm{t} = \emptyset$ is a finite set of predicate symbols that can be partitioned into a set of derived predicates $\mathcal{D}_\mathrm{q}$ and $\{<\}$,*
- *$\mathcal{T} \supseteq \{\text{OBJECT}\}$ is a finite set of* (PDDL) types *with partial order $\sqsubseteq$,*
- *$\mathcal{C}$ is a finite set of constants,*
- *$\mathcal{O}$ is a finite set of PDDL operators (or actions) over $\langle \mathcal{T}, \sqsubseteq, \Sigma_{\mathcal{C},\mathcal{P}_\mathrm{t}} \rangle$,*
- *$\mathcal{A}_\mathrm{t}$ and $\mathcal{A}_\mathrm{q}$ are stratifiable finite sets of PDDL axioms over $\langle \mathcal{T}, \sqsubseteq, \Sigma_{\mathcal{C},\mathcal{P}_\mathrm{t}} \rangle$ and $\langle \mathcal{T}, \sqsubseteq, \Sigma_{\mathcal{C},\mathcal{P}_\mathrm{t} \cup \mathcal{P}_\mathrm{q}} \rangle$, respectively.*
- *legal is the 0-ary query predicate with legal $\in \mathcal{D}_\mathrm{q}$,*
- *$<$ is a binary basic* ordering predicate*, and*
- *$\mathcal{G}$ is a first-order sentence over $\langle \mathcal{T}, \sqsubseteq, \Sigma_{\mathcal{C},\mathcal{P}_\mathrm{t}} \rangle$.*

Since the goal is fixed, an *instance* $\langle \mathcal{C}', \mathcal{I} \rangle$ of the domain is given by a set $\mathcal{C}'$ of additional instance-specific constants and the initial state $\mathcal{I}$, which is a basic state over $\langle \mathcal{T}, \sqsubseteq, \Sigma_{\mathcal{C} \cup \mathcal{C}', \mathcal{P}_\mathrm{t}} \rangle$. Any such instance where *legal*() is true in the extension of $\mathcal{I}$ with the axioms from $\mathcal{A}_\mathrm{t} \cup \mathcal{A}_\mathrm{q}$ is a legal instance of the domain. In this work, we generate such legal instances given a planning domain and the cardinality of $\mathcal{C}'$.

Answer Set Programming An *answer set program* (Baral 2010) is a set of logic programming rules similar to PDDL axioms.

Definition 3 (Answer set program)**.** *An* answer set program *(ASP program) for a relational first-order signature $\Sigma_{\mathcal{C},\mathcal{P}}$ is a finite set of rules of the form $H(\bar{x}) \leftarrow \exists \bar{y} B(\bar{x}, \bar{y})$, where*

- *the* head $H(\bar{x})$ *is a disjunction of atoms over $\Sigma_{\mathcal{C},\mathcal{P}}$ with $\bar{x}$ containing the free variables of all its atoms, and*
- *the* body $B(\bar{x}, \bar{y})$ *is a conjunction of literals over $\Sigma_{\mathcal{C},\mathcal{P}}$ which use variables only from $\bar{x}$ and $\bar{y}$, and where each variable of $\bar{x}$ is mentioned in at least one positive literal.*

A rule is ground *if $\bar{x}$ and $\bar{y}$ are empty, which we will write as $H \leftarrow B$.*

A rule with an empty head (and no free variables) is an *integrity constraint*. Such a rule means that the body must not evaluate to true. A rule with an empty body is a *fact* and expresses that the head must always be true.

ASP program $\mathcal{P}$ induces the ground ASP program *ground*($\mathcal{P}$) as the set of all (ground) rules that can be constructed by substituting every variable in a rule $r \in \mathcal{P}$ with a constant from $\mathcal{C}$.

A Herbrand interpretation I for $\Sigma_{\mathcal{C},\mathcal{P}}$ *satisfies* a ground rule $H \leftarrow B$ if $I \models B$ implies $I \models H$. The *reduct* $\mathcal{P}^I$ of a ground answer set program $\mathcal{P}$ for interpretation I is the ground program $\{H \leftarrow B^+ \mid I \models B^-, H \leftarrow B \in \mathcal{P}\}$, where B^+ is the conjunction of the positive literals of B and B^- is the conjunction of the negative literals of B.

Definition 4 (Answer set)**.** *An interpretation I is a* stable model *(or* answer set*) of ASP program $\mathcal{P}$ if it satisfies every rule in ground($\mathcal{P}$)^I and is subset-minimal among the models of ground($\mathcal{P}$)^I (viewed as sets of true ground atoms).*

ASP programs can also include *choice rules* of the form $\{P(\bar{x})\} \leftarrow B(\bar{x}, \bar{y})$, where P is a predicate and $B(\bar{x}, \bar{y})$ is a body as in Def. 3. If the body is true then $P(\bar{x})$ may be added to the answer set.

3 Instance Generation

Given planning domain $\langle \mathcal{P}_\mathrm{t}, \mathcal{P}_\mathrm{q}, \mathcal{T}, \mathcal{C}, \mathcal{O}, \mathcal{A}_\mathrm{t}, \mathcal{A}_\mathrm{q}, \mathit{legal}, <, \mathcal{G}\rangle$ and a number n that specifies how many objects the generated instances should have (in addition to the ones already defined in the domain), we create an ASP program $\mathcal{P}$ such that the set of answer sets corresponds to the set of legal instances for the domain with constants $\mathcal{C}_{\mathit{asp}} = \mathcal{C} \cup \{c_1, \ldots, c_n\}$.

In a preprocessing phase, we compute an equivalent stratifiable set $\mathcal{A}$ of PDDL axioms from $\mathcal{A}_\mathrm{t}$ and $\mathcal{A}_\mathrm{q}$ such that every axiom in $\mathcal{A}$ has the form $H(\bar{x}) \leftarrow \exists \bar{y} B(\bar{x}, \bar{y})$, where $B(\bar{x}, \bar{y})$ is a conjunction of literals with free variables from $\bar{x}$ and $\bar{y}$. We use the translator (Helmert 2009) of the Fast Downward planning system (Helmert 2006) for this purpose. While this transformation is not always possible in theory (Röger and Grundke 2024), this has never been a limitation in practice.

We use the ASP solver clingo (Gebser et al. 2019) to determine answer sets for $\mathcal{P}$. It treats $<$ as a built-in predicate (just as $=$), fixing some total order over all objects. Since the legality test in planning domains must be order-invariant, we can directly use this built-in predicate for the ordering predicate of the domain without any additional treatment.

For each TYPE $\in \mathcal{T}$, we use a fresh unary predicate *type* in $\mathcal{P}$. Program $\mathcal{P}$ contains the following rules:

Translated axioms Each axiom in $\mathcal{A}$ is already in the syntactic form $H(x_1, \ldots, x_r) \leftarrow \exists x_{r+1}, \ldots, x_s B(x_1, \ldots, x_s)$ but each variable x_i is associated with a type T_i that restricts the underlying domain. To enforce these types in the answer set program, we translate the axiom to

$$H(x_1, \ldots, x_r) \leftarrow \exists x_{r+1}, \ldots, x_s (B(x_1, \ldots, x_s) \wedge t_1(x_1) \wedge \cdots \wedge t_s(x_s)).$$

Type rules For each $c \in \mathcal{C}_{\mathit{asp}}$, there is a fact

$$\mathit{object}(c) \leftarrow \tag{1}$$

introducing all constants and making sure that they will be associated with type OBJECT.

For each domain constant $c \in \mathcal{C}_{dom}$ associated with a type $\mathrm{T} \neq$ OBJECTS, the answer set should preserve this type but not assign a more specific subtype, which is ensured by the following rules:

$$t(c) \leftarrow \tag{2}$$

$$\leftarrow \mathit{type}(c) \qquad \text{for all TYPE} \sqsubseteq \mathrm{T} \tag{3}$$

Type hierarchy rules To ensure that the assigned types will be propagated to all supertypes wrt. ordering relation $\sqsubseteq$, we have for each SUBTYPE $\sqsubseteq$ SUPERTYPE a rule:

$$\mathit{supertype}(x) \leftarrow \mathit{subtype}(x) \tag{4}$$

In addition, we ensure that the planning instance assigns a single type to each object (from which it then inherits all supertypes). In combination with the other rules, this is achieved by requiring the following integrity constraint for any two immediate subtypes of the same type, i.e., $\mathrm{T} \sqsubseteq \mathrm{S}$ and $\mathrm{U} \sqsubseteq \mathrm{S}$ with $\mathrm{T} \neq \mathrm{U}$:

$$\leftarrow \exists x (t(x) \wedge u(x)) \tag{5}$$

	training	easy	medium	hard
Blocksworld	93	28	0	0
Childsnack	99	29	1	0
Ferry	99	30	27	0
Floortile	19	3	0	0
Miconic	99	30	20	0
Rovers	99	30	3	0
Satellite	99	30	20	0
Spanner	99	30	0	0
Transport	99	30	26	0

Table 1: Number of instances from our instance generator that we use for each category of the IPC. The IPC used 99 training instances and 30 instances in each of the three testing categories.

Argument type rules To enforce the types for the basic predicates, we add the following integrity constraints for each $P \in \mathcal{B}_t$ with $\mathit{types}(P) = (\mathrm{T}_1, \ldots, \mathrm{T}_n)$

$$\begin{aligned} &\leftarrow \exists x_1, \ldots, x_n \, (P(x_1, \ldots, x_n) \wedge \neg t_1(x_1)) \\ &\ldots \\ &\leftarrow \exists x_1, \ldots, x_n \, (P(x_1, \ldots, x_n) \wedge \neg t_n(x_n)). \end{aligned} \tag{6}$$

Legality rule To enforce that answer sets correspond to legal instances, we require the legality predicate to be true.

$$\leftarrow \neg \mathit{legal}() \tag{7}$$

Choice rules for basic predicates and types The answer set should be able to contain any subset of ground atoms over the basic predicates (respecting the other rules, but not requiring subset minimality). Moreover, the program should be able to assign types for the additional objects. We achieve this by the following two choice rules.

$$\{P(x_1, \ldots, x_r)\} \leftarrow \mathit{object}(x_1) \wedge \cdots \wedge \mathit{object}(x_r) \tag{8}$$

for each basic predicate P (with arity r), and

$$\{\mathit{type}(x)\} \leftarrow \mathit{object}(x) \text{ for each TYPE} \in \mathcal{T} \tag{9}$$

Due to the type rules, the body is true for any substitution of the variables x_i with constants from $\mathcal{C}_{\mathit{asp}}$, so that any ground atom can be made true based on these rules.

The set of ground basic atoms in an answer set defines the initial state of the generated instance. Each of the new objects $o \in \{o_1, \ldots, o_n\}$ is associated with the most specific (wrt. $\sqsubseteq$) type TYPE such that $\mathit{type}(o)$ is in the answer set.

4 Experiments

We follow the setup of the IPC 2023 learning track (Taitler et al. 2024) to generate a benchmark set of formal PDDL instances for the instance generator and to analyse how learning-based planners perform on the generated instances (for code and data cf. Grundke, Helmert, and Röger 2025).

We augmented all but one PDDL domain from the track with legality tests and a domain goal, replicating implicit legality constraints and the goals from the domain-specific instance generators (Grundke and Röger 2025). This is impossible for Sokoban, where the IPC instance generator only creates solvable instances: deciding solvability is PSPACE-complete for Sokoban (Culberson 1997) but the formalism in Definition 1 captures only P.

Coverage	ASNets^{G}	ASNets^{C}	SMAC^{G}	SMAC^{C}	GOFAI^{G}	GOFAI^{C}	Vanir^{G}	Vanir^{C}	HUZAR^{G}	HUZAR^{C}	Muninn^{G}	Muninn^{C}
Blocksworld^{C} (90)	**31**	12	**60**	58	**63**	None	**60**	**60**	57	**61**	11	**37**
Childsnack^{C} (90)	None	**12**	29	**32**	**35**	None	**35**	**35**	35	**36**	**7**	6
Ferry^{C} (90)	19	**20**	**65**	**65**	**62**	**62**	**77**	**77**	**68**	64	23	**24**
Floortile^{C} (90)	**0**	**0**	None	**16**	**36**	35	**11**	**11**	**14**	**14**	**0**	**0**
Miconic^{C} (90)	10	**29**	**0**	**0**	**90**	**90**	**90**	**90**	**90**	**90**	**30**	**30**
Rovers^{C} (90)	4	**7**	50	**66**	**56**	**56**	**68**	**68**	**67**	66	**9**	8
Satellite^{C} (90)	**40**	**40**	84	**90**	**78**	77	**89**	**89**	66	**88**	9	**14**
Spanner^{C} (90)	**27**	12	**30**	None	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
Transport^{C} (90)	10	**31**	**67**	**67**	**71**	**71**	**66**	**66**	**66**	**66**	**12**	**12**
Sum	141	163	385	394	521	421	526	526	493	515	131	161
Blocksworld^{G} (28)	**18**	1	**28**	**28**	**28**	None	**28**	**28**	**28**	**28**	11	**25**
Childsnack^{G} (30)	None	**12**	**28**	**28**	**27**	None	**27**	**27**	27	**28**	4	**9**
Ferry^{G} (57)	**44**	39	**57**	**57**	**57**	**57**	**57**	**57**	**57**	**57**	**21**	19
Floortile^{G} (3)	**0**	**0**	None	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**1**	0
Miconic^{G} (50)	15	**44**	**0**	**0**	**50**	**50**	**50**	**50**	**50**	49	**29**	**29**
Rovers^{G} (33)	14	**16**	**23**	**23**	**24**	**24**	**33**	**33**	**33**	**33**	**20**	19
Satellite^{G} (50)	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**
Spanner^{G} (30)	**7**	5	**24**	None	**24**	**24**	**24**	**24**	**24**	**24**	**24**	**24**
Transport^{G} (56)	11	**17**	**56**	**56**	**56**	**56**	**56**	**56**	**56**	54	10	**11**
Sum	126	151	233	212	286	231	295	295	295	293	137	153

Table 2: Number of solved instances for each domain and planner variant. Bold numbers mark the highest coverage among the two variants of the same planner. "None" indicates that a planner did not learn domain knowledge for this domain and thus did not solve any tasks.

To generate instances of different sizes, the IPC instance generators use parameters specifying the number of objects of certain types. We thus support this by means of additional type rules. Additionally, one can specify cardinality constraints for the predicates, which for example is needed for the Childsnack domain where the predicate *allergic_gluten* must be true for a certain number of children.

To get diverse instances we use the *S-Greedy* approach (Böhl, Gaggl, and Rusovac 2023) for diverse answer sets.

Instance Generation Like the IPC we generate testing instances of three difficulty levels and training instances. For each run of the instance generator (given a domain and size parameters), we use a 30 minute time limit and 4 GiB memory limit. If our generator produces more instances than used in the IPC, we sample uniformly from the generated instances. Table 1 shows how many instances we generated for the subsequent planner experiment.

In most successful runs (99 training instances or 30 for a certain difficulty level) the instance generator produces many more instances than needed. Thus, of all generated instances we use only a small fraction because we aimed for the same spread of numbers of objects as the IPC instances. For example, we generated 1057 Blocksworld instances with 25 blocks, of which we only use 6 instances.

The IPC mostly scaled "difficulty" by adding more objects, e.g. up to 500 blocks in the hard Blocksworld instances. We observe that our approach does not scale well with the number of objects. For all domains but Spanner, at a certain number of objects the instance generator starts running out of time. For Blocksworld, this first happens with 27 blocks. With even more objects, we tend to run out of memory before running out of time (mostly during the grounding process of clingo). Spanner has no time-outs, but the generator runs out of memory with more than 60 objects.

Planner Results In the IPC 2023, the planners (Drexler 2023; Gzubicki, Lachowicz, and Torralba 2023; Hao et al. 2023; Seipp, Sievers, and Hutter 2023; Ståhlberg, Bonet, and Geffner 2023; Torralba and Gnad 2023) had 24 hours and 32 GiB of memory to learn domain knowledge from the training instances in each domain. Afterwards they had to solve the testing instances with a time limit of 30 minutes and a memory limit of 8 GiB per task.

We used the same setup to train each planner once on the IPC training instances (planner^{C}) and once on our generated training instances (planner^{G}). Afterwards we tested each planner on both the testing instances of the IPC (Domain^{C}) and our generated instances (Domain^{G}).

Table 2 shows the number of solved instances for each combination. We observe that the planners trained on the IPC instances tend to solve more instances than the ones that learned from our generated training set. This is more pronounced on the IPC test benchmarks but also the case for our generated test instances. If we look at the per-domain results, we see that there are nevertheless cases in which the planner^{G} variant is strictly better than planner^{C}. On the IPC instances, this is the case for 12 planner/domain combinations compared to 15 for which it is worse. On the generated instances, this changes: The planner^{G} variant is better than planner^{C} in 11 cases and worse in only 9 cases.

5 Conclusion

We demonstrated that domain-independent instance generation for planning tasks with declarative problem solvers is feasible. Our approach cannot match the scalability of existing domain-specific generators, but for a first effort in this direction using off-the-shelf diverse ASP solvers this is not surprising. We believe that the instance generator we presented can reduce the knowledge engineering efforts for future work on planning and learning and also serves as a useful challenge for future work on diverse ASP solving.

Acknowledgements

We have received funding for this work from the Swiss National Science Foundation (SNSF) as part of the project "Lifted and Generalized Representations for Classical Planning" (LGR-Plan).

References

Akgün, Ö.; Dang, N.; Espasa, J.; Miguel, I.; Salamon, A.; and Stone, C. 2020. Exploring instance generation for automated planning. In *CP 2020 Workshop on Constraint Modelling and Reformulation*.

Baral, C. 2010. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

Böhl, E.; Gaggl, S. A.; and Rusovac, D. 2023. Representative answer sets: Collecting something of everything. In Gal, K.; Nowé, A.; Nalepa, G. J.; Fairstein, R.; and Rădulescu, R., eds., *Proceedings of the 26th European Conference on Artificial Intelligence (ECAI 2023)*, 271–278. IOS Press.

Culberson, J. C. 1997. Sokoban is PSPACE-complete. Technical Report TR 97-02, Department of Computing Science, The University of Alberta, Edmonton, Alberta, Canada.

Drexler, D. 2023. Vanir: Learning and executing width-based hierarchical policies. In *Learning Track of the International Planning Competition 2023: Planner Abstracts*.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2019. Multi-shot ASP solving with clingo. *Theory and Practice of Logic Programming* 19(1):27–82.

Grundke, C., and Röger, G. 2025. Formally represented PDDL planning domains. https://doi.org/10.5281/zenodo.16875437.

Grundke, C.; Helmert, M.; and Röger, G. 2025. Code, benchmarks and experiment data for the KR 2025 paper "Domain-Independent Instance Generation for Classical Planning". https://doi.org/10.5281/zenodo.16875683.

Grundke, C.; Röger, G.; and Helmert, M. 2024. Formal representations of classical planning domains. In Bernardini, S., and Muise, C., eds., *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*, 239–248. AAAI Press.

Gzubicki, P. R.; Lachowicz, B. P.; and Torralba, Á. 2023. HUZAR: Predicting useful actions with graph neural networks. In *Learning Track of the International Planning Competition 2023: Planner Abstracts*.

Hao, M.; Toyer, S.; Wang, R.; Thiébaux, S.; and Trevizan, F. 2023. Action schema networks – IPC version. In *Learning Track of the International Planning Competition 2023: Planner Abstracts*.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence* 173:503–535.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL – The Planning Domain Definition Language – Version 1.2. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, Yale University.

Núñez-Molina, C.; Mesejo, P.; and Fernández-Olivares, J. 2024. NeSIG: A neuro-symbolic method for learning to generate planning problems. In Endriss, U., and Melo, F. S., eds., *Proceedings of the 27th European Conference on Artificial Intelligence (ECAI 2024)*, 4084–4091. IOS Press.

Röger, G., and Grundke, C. 2024. Negated occurrences of predicates in PDDL axiom bodies. In *Proceedings of the KI 2024 Workshop on Planning, Scheduling and Design (PuK 2024)*.

Seipp, J.; Sievers, S.; and Hutter, F. 2023. Fast Downward SMAC. In *Learning Track of the International Planning Competition 2023: Planner Abstracts*.

Slaney, J., and Thiébaux, S. 2001. Blocks World revisited. *Artificial Intelligence* 125(1–2):119–153.

Socher-Ambrosius, R., and Johann, P. 1997. *Deduction systems*. Graduate Texts in Computer Science. Springer.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2023. Muninn. In *Learning Track of the International Planning Competition 2023: Planner Abstracts*.

Taitler, A.; Alford, R.; Espasa, J.; Behnke, G.; Fišer, D.; Gimelfarb, M.; Pommerening, F.; Sanner, S.; Scala, E.; Schreiber, D.; Segovia-Aguas, J.; and Seipp, J. 2024. The 2023 International Planning Competition. *AI Magazine* 45(2):280–296.

Thiébaux, S.; Hoffmann, J.; and Nebel, B. 2005. In defense of PDDL axioms. *Artificial Intelligence* 168(1–2):38–69.

Torralba, Á., and Gnad, D. 2023. GOFAI. In *Learning Track of the International Planning Competition 2023: Planner Abstracts*.