Arguments in Favor of Allowing a Modeler to Constrain Action Repetitions

Pascal Lauer

School of Computing, The Australian National University, Canberra, Australia
& Saarland Informatics Campus, Saarland University, Saarbrücken, Germany
lauer@cs.uni-saarland.de

Abstract

It is increasingly recognized that adding additional constraints to a planning domain can significantly enhance expressiveness and solvability. In this paper, we advocate for bounding action repetitions, i.e., the number of times an action occurs in a plan. We show that bounds on action repetitions occur naturally in many domains and, when enforced, reduce computational complexity. In particular, we show that bounding all actions reduces the complexity of plan existence to NP. We also identify planning tasks where plan existence remains NP-complete when bounding only some actions.

1 Introduction

Research in automated planning focuses on developing one algorithm (planner) that solves arbitrary problems from a user-provided description. To describe these problems, many different planning formalisms exist. A commonly considered formalism is classical planning where action descriptions are limited to simple logical constraints. The main selling point of classical planning is that planners achieve a lot better performance than in more expressive formalisms. To this day, research about solving classical planning tasks mostly focuses on heuristic search[1], which remains to be most successful in general (Taitler et al. 2024). But, other solvers may outperform heuristic search in specific settings, especially when tailoring problem models in the right way. E.g., if one can provide a close estimate on the plan length, this allows for great performance for planning via SAT (Rintanen 2014; Höller and Behnke 2022). The problem is that providing this bound is in general as hard as planning itself.

In this work, we argue that while helpful bounds are hard to infer in general, meaningful bounds can often be established by the modeler in practice. In particular, by establishing a bound on how many times single actions may be repeated in a plan. To illustrate our argument, we show that the bounds are easy to establish in logistics-like tasks, which have been a central focus in the planning community. To test whether these bounds are too restrictive, we analyze the complexity of plan existence with bounded action repetitions. The complexity drops from PSPACE-complete in the unbounded case (Erol, Nau, and Subrahmanian 1995) to NP-complete, matching complexity of unary-bounded plan existence (Bäckström and Jonsson 2011) and SAT solving. This is exactly what we want: We lower complexity, to make solving easier, without limiting us to too simple problems.

We extend this result by showing that bounding only some action repetitions is often enough to reduce the complexity of plan existence to NP. Our results suggest that modelers can significantly reduce planning time by providing very limited information that is easy to obtain.

[1]Examples are the works by: Hoffmann et al. 2001; 2004; 2005; Helmert et al. 2007; 2014; Pommerening et al. 2013; 2014; Seipp et al. 2015; 2018; Corrêa et al. 2020; 2021; 2022; Lauer et al. 2020; 2021; 2025a; Fišer et al. 2022a; 2022b; 2024; Chen et al. 2024

2 Background

Classical Planning Tasks A *classical planning task* is a tuple $\Pi = \langle F, A, s_0, G\rangle$. F is a finite set of propositional facts. We associate the set of *literals* L with F, which add fact negations, i.e., $L := F \cup \{\neg f \mid f \in F\}$. An *action* $a = \langle pre(a), \mathit{eff}(a)\rangle$ consists of the *precondition* $pre(a) \subseteq L$ and the *effect* $\mathit{eff}(a) \subseteq L$, which are both sets of literals. A is the finite set of actions in the task. A *state* $s \subseteq F$ represents the facts that are true in that state. The set of all states is denoted by $\mathcal{S} := 2^F$. The *literal extension* $s^L := s \cup \{\neg f \mid f \in \mathcal{S} \setminus s\}$ of state s contains all positive facts of s and the negation of all others. $s_0 \subseteq F$ is a state, called *initial state*. $G \subseteq L$ is the *goal condition*.

For action $a \in A$: $add(a) := \{f \in F \mid f \in \mathit{eff}(a)\}$ are the positive effects, and $del(a) := \{f \in F \mid \neg f \in \mathit{eff}(a)\}$ the negative effects. A fact f is called *non-deletable* iff there is no action $a \in A$ with $f \in del(a)$. An action a is applicable in a state $s \in \mathcal{S}$ iff $pre(a) \subseteq s^L$. Applying a to s results in the successor state $progr(s, a) := (s \setminus del(a)) \cup add(a)$. $progr(s, a)$ is undefined if a is not applicable in s.

A state s_n is reachable by a sequence of actions $\pi = \langle a_1, \ldots, a_n\rangle$ if each a_i is applicable in the state resulting from applying $a_1, \ldots, a_{i-1}$, starting from s_0. That is: $s_n = progr(...progr(progr(s_0, a_1), a_2), ..., a_n)$. The sequence π is a *plan* iff s_n satisfies the goal G, i.e., $G \subseteq s_n^L$.

Running Example: Logistics To illustrate our results, we use a simplified Logistics domain as running example. It captures typical features of Logistics domains while remaining concise enough for presentation within this paper. The domain consists of trucks moving on a directed graph, picking up and delivering packages to their target locations.

A logistics task $\Pi_L = \langle F, A, s_0, G\rangle$ is defined based on a

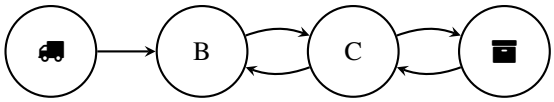

Figure 1: Logistic task with $Trucks = \{T_1\}$, $Packages = \{p_1\}$, $Roads = \{(A,B),(B,C),(C,B),(C,D),(D,C)\}$, and $Locations = \{A,B,C,D\}$. The initial state is $s_0 = \{at(t_1,A), at(p_1,D)\}$. A goal condition that could be fulfilled by a plan would be $G_1 = \{at(p_1,D)\}$. The goal condition $G_2 = \{at(p_1,A)\}$ could never be reached.

directed graph $RoadSystem = (Locations, Roads)$, a set of packages $Packages$, and a set of trucks $Trucks$.

The fact set F includes atoms $at(t,l)$ for each truck $t \in Trucks$ and location $l \in Locations$, $at(p,l)$ for each package $p \in Packages$ and location $l \in Locations$, and $in(p,t)$ for each package $p \in Packages$ and truck $t \in Trucks$. There are two actions:

$$\begin{aligned} Load(p,t,l): &\quad \text{pre: } at(p,l),\ at(t,l) \\ &\quad \text{eff: } \neg at(p,l),\ in(p,t) \end{aligned}$$

$$\begin{aligned} Unload(p,t,l): &\quad \text{pre: } in(p,t),\ at(t,l) \\ &\quad \text{eff: } \neg in(p,t),\ at(p,l) \end{aligned}$$

for every package p, truck t, and location l. And also:

$$\begin{aligned} Drive(t,from,to): &\quad \text{pre: } at(t,from) \\ &\quad \text{eff: } \neg at(t,from),\ at(t,to) \end{aligned}$$

for each $(from,to) \in Roads$ and truck t. The initial state s_0 and goal G are specified per task instance. A concrete instantiation is shown in Figure 1, containing a single truck t_1, package p_1, and four locations: A, B, C, D.

Mutexes Facts $f_1,\ldots,f_n \in F$ are *mutex* if no reachable state contains more than one of them, i.e., $|\{f_1,\ldots,f_n\} \cap s| \leq 1$ for all reachable states s. Deciding whether two facts are mutex is, in general, as hard as planning itself[2], i.e., PSPACE-complete (Erol, Nau, and Subrahmanian 1995). To avoid this complexity, there exist different polynomial-time algorithms for detecting mutexes (Helmert 2009; Fišer and Komenda 2018). We assume a fixed finite set of polynomial-time algorithms to identify mutexes. We call facts $f_1,\ldots,f_n \in F$ *poly-mutex* if they are mutex and detected by this set of polynomial-time algorithms. E.g., it is easy to identify that in the Logistics example $at(t,l_1)$, $at(t,l_2)$ for $l_1 \neq l_2$ is mutex by simply realizing that there is one initial truck location and each action adds and deletes exactly one fact designating the truck location.

Petri Nets We will use Petri Nets to prove our claims in Section 5. We adopt the notation of Esparza (1995), adapting the terminology to better align with classical planning tasks, to help readers more familiar with planning.

[2]This can be shown by adding a new fact f to the initial state and replacing the goal with a new fact g, which is achieved by an action whose precondition encodes the original goal. Then, f and g are mutex iff there is no plan.

A *Petri net* is a tuple $\mathcal{N} = \langle P,T,W\rangle$, where P is a finite set of *places* (similar to facts), T is a finite set of *transitions* (similar to actions), and $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ is a *weight function* that defines directed arcs between places and transitions. These arcs determine how transitions consume tokens from input places and produce tokens in output places. For a transition $t \in T$, its *pre-set* is $\mathrm{pre}(t) := \{p \in P \mid W(p,t) > 0\}$. A *marking* is a function $M\colon P \to \mathbb{N}$, similar to a state, assigns a non-negative number of tokens to each place. A *Petri net system* is a pair (N, M_0) where M_0 is the *initial marking*. A transition $t \in T$ is *enabled* (applicable) in marking M if for all $p \in \mathrm{pre}(t)$ it holds that $M(p) \geq W(p,t)$. If t is enabled, it can *fire* (be applied), producing a new marking $\mathrm{trans}(M,t) := M'$ defined by $M'(p) := M(p) - W(p,t) + W(t,p)$ for all $p \in P$. If t is not enabled, then $\mathrm{trans}(M,t)$ is undefined.

The *reachability problem* asks: given a Petri net system (N, M_0) and a marking M, does there exist a sequence of transitions $\pi = \langle t_1,\ldots,t_n\rangle$ such that applying them successively leads to M, i.e., $M = \mathrm{trans}(...\mathrm{trans}(\mathrm{trans}(M_0,t_1),t_2),...,t_n)$? This problem is Ackermann-complete in general (Leroux and Schmitz 2019), therefore we consider a constrained class:

A Petri net $N = \langle P,T,W\rangle$ is called *communication-free* if each transition consumes at most one token from at most one place. I.e., for or every $t \in T$ it holds that $|\mathrm{pre}(t)| \leq 1$ and $W(p,t) \leq 1$ for all $p \in P$. A Petri net system (N, M_0) is *communication-free* iff N is. For this class of nets, the reachability problem is in NP-complete (Esparza 1995).

3 Action Repetition Bounds

We now describe a way to formally capture bounds on action repetitions. To this end we introduce the notion of a *bound map* which maps actions to a bound on their repetitions.

Definition 1 (Bound Map)**.** *Let $\Pi = (F,A,s_0,G)$ be a classical planning task. A* bound map *is a function $b : A \to \mathbb{N} \cup \{\infty\}$ indicating the maximum number of times $b(a)$ an action $a \in A$ may be applied. A plan π* respects *a bound map b iff for every action $a \in A$ the number of occurrences of a in π does not exceed $b(a)$.*

In some planning tasks, limiting the number of times actions may be applied does not affect solvability. This motivates a semantic notion of when a bound is *non-restrictive*.

Definition 2 (Non-Restrictive Bound Map)**.** *Abound map b is* non-restrictive *for a planning task Π iff: There exists a plan that respects b or there is no plan for Π.*

We now observe that non-zero bounds on $Load$ and $Unload$ actions are non-restrictive for Logistics tasks.

Example 1. *Let Π_L and let b be a bound map such that $b(a) = 1$ for all actions a of the form $Load(p,t,l)$ or $Unload(p,t,l)$, and $b(a) = \infty$ for all other (Drive) actions.*

In any valid plan for $\mathcal{P}_L$, one can remove all `Load` *actions for a package p after its first load at the source, and all* `Unload` *actions before its final unload at the destination. Preconditions remain satisfied, so these repetitions are unnecessary. The resulting plan respects b, showing that $\mathcal{P}_L$ is solvable if and only if a b-respecting plan exists.*

4 Planning with Constantly Bounded Action Repetitions is NP-Complete

Reconsider Figure 1, a simple logistics task with only one package to deliver. If there is only one package, each road (edge) needs to be used at most once. So a constant bound of 1 on drive actions suffices. A similar situation arises in the well-known Gripper domain, where a robot moves balls from one room to another: The robot only needs to move to a room at most twice, once to collect and once to deliver.

These examples show that there are planning tasks, that are commonly used in our community, where all actions can be bounded by a small constant. We now formalize this insight and show that plan existence becomes NP-complete when all action repetitions are bounded by a fixed constant.

Theorem 1. *Deciding whether there is a plan for a classical planning task $\Pi = (F, A, s_0, G)$ that respect bound b, where each $b(a)$ is at most some constant $k \in \mathbb{N}^+$ is NP-hard.*

Proof. Membership: Since each action $a \in A$ can occur at most $b(a) \leq k$ times, any plan has length at most $|A| \cdot k$, which is polynomial in the size of the input. A non-deterministic polynomial-time algorithm can guess the a plan and reject if it does not match b's bounds.

Hardness: We reduce from the NP-complete problem 3SAT. Let φ be a 3-CNF formula over variables $x_1, \ldots, x_n$ with clauses $C_1, \ldots, C_m$. We define a planning task $\mathcal{P}_\varphi = \langle F, A, s_0, G \rangle$ solving the 3SAT formula as follows, akin to Bylander (1994, Thm. 3.5). Facts F include facts $true_{x_i}$, $false_{x_i}$, and $unassigned_{x_i}$ for each variable x_i; and also a fact sat_{C_j} for each clause C_j. The initial state $s_0 = \{unassigned_{x_i} \mid i = 1, \ldots, n\}$ marks all variables unassigned. And the goal $G = \{sat_{C_j} \mid j = 1, \ldots, m\}$ requires all conjunctions to be fulfilled. The actions are:

- Two assignment actions for each variable x_i:

$$a^T_{x_i} = (\{unassigned_{x_i}\}, \{\neg unassigned_{x_i}, true_{x_i}\})$$
$$a^F_{x_i} = (\{unassigned_{x_i}\}, \{\neg unassigned_{x_i}, false_{x_i}\})$$

- For each clause C_j, positive occurrence x_i in C_j:

$$a^{x_i}_{C_j} = (\{true_{x_i}\}, \{sat_{C_j}\})$$

- For each clause C_j, negative occurrence $\neg x_i$ in C_j:

$$a^{\neg x_i}_{C_j} = (\{false_{x_i}\}, \{sat_{C_j}\})$$

Each variable assignment action is applicable once, since $unassigned_{x_i}$ is deleted. Thus, exactly one of $a^T_{x_i}$ or $a^F_{x_i}$ can be applied for each x_i, encoding a truth assignment. Each clause action a_{C_j} is applicable only if at least one of its literals is satisfied by the chosen assignment. Hence, there exists a valid plan iff φ is satisfiable. Since each action is applied at most once, a bound map b with $b(a) = 1$ for all $a \in A$ suffices, completing the construction. □

5 Planning with Simple Actions is NP-Complete

In the previous section, we assumed the modeler being able to identify constant bounds for *Drive* actions. But, this may not always be obvious or possible. To show that sometimes, the bounds are not needed, we now show that if the action structure of a planning task is simple enough, the complexity of plan existence still decreases to NP. In the next section, we combine both ideas by bounding only non-simple actions.

We identify a notion of simple actions which aligns with our running example and other IPC benchmarks.

Definition 3. *An action $a \in A$ is called* simple *if it has exactly one fact $f \in F$ in its precondition and delete list, i.e., $\{f\} = pre(a) = del(a)$, and every add effect $f' \in add(a)$ is poly-mutex with f or non-deletable. Actions that are not simple are called* non-simple*.*

E.g., all *Drive* actions in the running example are simple, as truck locations are trivially mutex. Another example is the VisitAll domain from the IPC benchmarks. It contains only *Move* actions, which are similar to *Drive* actions but additionally mark positions as visited using non-deletable facts, which make them simple. We will now prove that this restriction decreases the complexity of plan existence.

Theorem 2. *Deciding whether there is a plan for a planning task $\Pi = (F, A, s_0, G)$ where all actions $a \in A$ are simple is NP-complete. (Even without bounding action repetitions.)*

Proof. Membership: We follow the construction by Hickmott et al. (2007) to encode Π into a communication-free Petri net $\mathcal{N} = (P, T, W)$, tailored to simple actions:

- For each fact $f \in F$, include a place p_f in S.
- For each action $a \in A$, include a transition t_a in T with:
  - $W(p_f, t_a) = 1$ for the unique fact f with $\{f\} = pre(a) = del(a)$.
  - $W(t_a, p_f) = 1$ for every $f \in add(a)$.

All unspecified weights are zero. A token in p_f means fact f is true. The transition construction preserves the semantics of action application: Places f_p for non-deletable facts f can accumulate multiple token, which is no problem as they are never deleted. For all other facts never accumulate more than one token, since added facts are mutex with the precondition and so always 0 when a token is placed there. This means, the subtraction of one in firing transitions removes all tokens in the place, like the action would negate the fact. The initial marking M_0 is constructed by placing a token on places corresponding to the facts in the state. That is: $M_0 := \{p_f \mapsto bool(f \in s_0) \mid f \in F\}$ where *bool* maps true statements to 1 and false statements to 0.

We will now construct the mapping M, representing the goal state, for which we want to check Petri-Net reachability. By Esparza (1995), communication-free Petri net reachability can be encoded as an ILP with columns and rows of linear size, where some variables represent the final marking values. Thus, by Papadimitriou (1981), if the ILP is solvable, then it has a solution with values bounded by an exponential constant u. Hence, if a marking is reachable, there exists one with the same values except for those exceeding u, which can be reduced to at value less or equal u. This allows us to non-deterministically guess the marking M as follows: For each non-deletable fact $f \in F$, guess $M(p_f) \in \{1, ..., u\}$, if $f \in G$ if $\neg f \in G$, set

$M(p_f) = 0$; otherwise guess $M(p_f) \in \{0, ..., u\}$. For all other facts $f \in F$, set $M(p_f) = 1$ if $f \in G$, $M(p_f) = 0$ if $\neg f \in G$, and guess $M(p_f) \in \{0, 1\}$ otherwise. Then apply the non-deterministic reachability check of Esparza (1995). The marking is reachable iff a plan exists.

Hardness: We reduce from 3SAT as in Theorem 1, where each variable assignment action satisfies the simple action condition since variable values are trivially mutex. Clause satisfaction actions add non-deletable facts and have no delete effects, hence are also simple. Thus, the reduction fits our setting, proving NP-hardness. □

6 Planning with Bounded Action Repetitions or Simple Actions remains NP-Complete

We conclude that bounding only a subset of action repetitions is sufficient when the remaining actions are simple. This applies to tasks such as Logistics, where determining a bound for $Drive$ actions may be difficult, but $Load$ and $Unload$ actions can naturally be bounded. Since $Drive$ actions are simple, bounding only $Load$ and $Unload$ suffices to use an algorithm with reduced complexity.

Theorem 3. *Let $k \in \mathbb{N}$ be a fixed constant. Given a classical planning task $\Pi = (F, A, s_0, G)$, and a bound map b for Π, such that every action $a \in A$ with $b(a) > k$ is simple:*

Deciding whether there exists a plan for Π that respects b is NP-complete.

Proof. Membership: We use a guess-and-check algorithm. There are at most $|A| \cdot k$ non-simple actions in every plan that respects $|b|$, so we guess:

1. An integer $0 \leq m \leq |A| \cdot k$
2. A sequence of m non-simple actions $a_1, \ldots, a_m$.
3. A sequence of states $s_1, \ldots, s_m$, where each s_i is the state in which a_i can be applied.

$a_1, \ldots, a_m$ is the subsequence of actions in the plan we are guessing for. To ensure that simple actions can be inserted between them, we verify, for each $i = 0, \ldots, m$, that state s_{i+1} is reachable from s_i by constructing a planning task restricted to the set of simple actions, with initial state s_i and goal s_{i+1}. We then apply the plan verification algorithm for task with simple actions, as established in Theorem 2. Finally, we guess whether the goal is reached directly by the last non-simple action a_m, in which case we verify that applying a_m in s_m achieves G. Otherwise, we check that G is reachable from s_m using only simple actions, again by applying the verification algorithm for the simple action fragment.

Hardness: We reduce from the special case of the bounded planning problem where all actions are non-simple, which is shown to be NP-complete in Theorem 2.

We drop the possibility of simple actions by adjusting all actions to become non-simple. This is done by adding two static preconditions, $useless_1$ and $useless_2$, to every action. These facts are included in the initial state and are not affected by any action. This ensures that all actions violate the simplicity condition, while preserving their original semantics. □

7 Related & Future Work

Model Constraints Prior work has formalized ways of adding constraints and preferences to planning tasks (Gerevini and Long 2005; Edelkamp 2006; Lauer et al. 2025b). However, the constraint frameworks are very general. Offering a broad variety of constraints is not necessarily an advantage. It leaves modelers uncertain about which constraints are useful to include in the task. Our work shows that bounds on action repetitions are one type of constraint that modelers should include, as they are often easy to encode and seem likely to improve solver performance.

Bounding Plan Length Bounding action repetitions is closely related to bounding plans directly, a topic thoroughly studied in prior work (Bäckström and Jonsson 2011; Lin et al. 2024; Lauer, Lin, and Bercher 2025). This is similar in theory, but quite different in practice: For a human it is far easier to observe a bound on an individual action repetitions than the total plan length. E.g., in our running example any delivery driver knows that they $Load$ and $Unload$ every package once. An estimate on the total plan length, requires additional reasoning to add up several bounds and is more error-prone. Moreover, a bound on the plan length always bounds *all* repetitions, instead of possibly just some. I.e., bounds on specific operators are more fine-grained, making them easier to encode and exploit in practice.

Simple Actions There are alternatives for defining simple actions to still capture our $Drive$ actions. E.g. one could use SAS-US (Bäckström and Klein 1991), where plan existence is also NP-complete. The difference between SAS-US and our definition is that SAS-US allows multiple preconditions while we allow multiple effects. To the best our knowledge our definition was not studied before. We chose the definition, as it matches multiple (sub)tasks from past International Planning Competitions. Since the Petri Nets constructed for tasks with simple actions can be encoded into Integer Linear Programs (Esparza 1995), this points at interesting future work.

Other Planning Formalisms We argue that constraint solving approaches, especially SAT, could become more competitive in solving classical planning task when providing additional constraints. The main reason is the drop in plan existence complexity from PSPACE to NP. As there are known PSPACE fragments of other formalism, e.g., numerical planning (Shleyfman, Gnad, and Jonsson 2023; Lauer et al. 2025b), it is reasonable to assume that this effect could transfer to those settings as well.

8 Conclusion

We have shown that bounding action repetitions is an effective tool for reducing the complexity of plan existence. In particular, it lowers the complexity from PSPACE-complete to NP-complete. We have also that this remains the case when bounding only some actions in many naturally occurring planning problems, e.g., Logistics-like domains. This highlights that bounding action repetitions is a powerful tool that should be get more focus from both a modeler and planner perspective.

Acknowledgments

This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – ProjectID 232722074 – SFB 1102.

References

Bäckström, C.; and Jonsson, P. 2011. All PSPACE-Complete Planning Problems Are Equal but Some Are More Equal than Others. In *4th SOCS*, 10–17.

Bäckström, C.; and Klein, I. 1991. Planning in polynomial time: the SAS-PUBS class. *Comput. Intell.*, 7: 181–197.

Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. *AIJ*, 69: 165–204.

Chen, D. Z.; Thiébaux, S.; and Trevizan, F. 2024. Learning domain-independent heuristics for grounded and lifted planning. In *38th AAAI*, 20078–20086.

Chen, D. Z.; Trevizan, F.; and Thiébaux, S. 2024. Return to tradition: Learning reliable heuristics with classical machine learning. In *34th ICAPS*, 68–76.

Corrêa, A. B.; Francès, G.; Pommerening, F.; and Helmert, M. 2021. Delete-relaxation heuristics for lifted classical planning. In *31st ICAPS*, 94–102.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Frances, G. 2020. Lifted successor generation using query optimization techniques. In *30th ICAPS*, 80–89.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2022. The FF Heuristic for Lifted Classical Planning. In *36th AAAI*, 9716–9723.

Edelkamp, S. 2006. On the Compilation of Plan Constraints and Preferences. In *16th ICAPS*, 374–377.

Erol, K.; Nau, D. S.; and Subrahmanian, V. S. 1995. Complexity, Decidability and Undecidability Results for Domain-Independent Planning. *AIJ*, 76: 75–88.

Esparza, J. 1995. Petri Nets, Commutative Context-Free Grammars, and Basic Parallel Processes. In *10th FCT*, 221–232.

Fišer, D.; and Komenda, A. 2018. Fact-Alternating Mutex Groups for Classical Planning. *JAIR*, 61: 475–521.

Fišer, D.; and Steinmetz, M. 2024. Towards Feasible Higher-Dimensional Potential Heuristics. In *34th ICAPS*, 210–2020.

Fišer, D.; Torralba, A.; and Hoffmann, J. 2022a. Operator-Potential Heuristics for Symbolic Search. In *36th AAAI*, 9750–9757.

Fišer, D.; Torralba, A.; and Hoffmann, J. 2022b. Operator-potentials in symbolic search: From forward to bi-directional search. In *32nd ICAPS*, 80–89.

Gerevini, A.; and Long, D. 2005. Plan constraints and preferences in PDDL3. Technical report, Department of Electronics for Automation, University of Brescia, Italy.

Helmert, M. 2009. Concise Finite-Domain Representations for PDDL Planning Tasks. *AIJ*, 173: 503–535.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible Abstraction Heuristics for Optimal Sequential Planning. In *17th ICAPS*, 176–183.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge & Shrink Abstraction: A Method for Generating Lower Bounds in Factored State Spaces. *JACM*, 61: 16:1–16:63.

Hickmott, S. L.; Rintanen, J.; Thiébaux, S.; and White, L. B. 2007. Planning via Petri Net Unfolding. In *20th IJCAI*, 1904–1911.

Hoffmann, J.; and Edelkamp, S. 2005. The Deterministic Part of IPC-4: An Overview. *JAIR*, 24: 519–579.

Hoffmann, J.; and Nebel, B. 2001. The FF planning system: fast plan generation through heuristic search. *JAIR*, 14: 253–302.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered Landmarks in Planning. *JAIR*, 22: 215–278.

Höller, D.; and Behnke, G. 2022. Encoding Lifted Classical Planning in Propositional Logic. In *32nd ICAPS*, 134–144.

Lauer, P.; and Fickert, M. 2020. Beating LM-cut with LM-cut: Quick Cutting and Practical Tie Breaking for the Precondition Choice Function. In *12th HSDIP at ICAPS*.

Lauer, P.; and Fišer, D. 2025. Potential Heuristics: Weakening Consistency Constraints. In *35th ICAPS*.

Lauer, P.; Lin, S.; and Bercher, P. 2025. Tight Bounds for Lifted HTN Plan Verification and Bounded Plan Existence. In *35th ICAPS*.

Lauer, P.; Torralba, Á.; Fišer, D.; Höller, D.; Wichlacz, J.; and Hoffmann, J. 2021. Polynomial-Time in PDDL Input Size: Making the Delete Relaxation Feasible for Lifted Planning. In *30th IJCAI*, 4119–4126.

Lauer, P.; Torralba, Á.; Höller, D.; and Hoffmann, J. 2025a. Continuing the Quest for Polynomial Time Heuristics in PDDL Input Size: Tractable Cases for Lifted hAdd. In *35th ICAPS*.

Lauer, P.; Zhang, Y.; Haslum, P.; and Bercher, P. 2025b. PSPACE Planning With Expressivity Beyond STRIPS: Plan Constraints via Unordered HTNs, ILPs, Numerical Goals, and More. In *8th HPLAN at ICAPS*.

Leroux, J.; and Schmitz, S. 2019. Reachability in Vector Addition Systems is Primitive-Recursive in Fixed Dimension. In *34th LICS*, 1–13.

Lin, S.; Olz, C.; Helmert, M.; and Bercher, P. 2024. On the Computational Complexity of Plan Verification, (Bounded) Plan-Optimality Verification, and Bounded Plan Existence. In *38th AAAI*, 20203–20211.

Papadimitriou, C. H. 1981. On the complexity of integer programming. *JACM*, 28: 765–768.

Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the Most Out of Pattern Databases for Classical Planning. In *23rd IJCAI*, 2357–2364.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-based heuristics for cost-optimal planning. In *24th ICAPS*, 226–234.

Rintanen, J. 2014. Madagascar: Scalable planning with SAT. In *IPC at ICAPS*, 1–5.

Seipp, J.; and Helmert, M. 2018. Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning. *JAIR*, 62: 535–577.

Seipp, J.; Pommerening, F.; and Helmert, M. 2015. New Optimization Functions for Potential Heuristics. In *25th ICAPS*, 193–201.

Shleyfman, A.; Gnad, D.; and Jonsson, P. 2023. Structurally Restricted Fragments of Numeric Planning - a Complexity Analysis. In *37th AAAI*, 12112–12119.

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