Proceedings of the Thirty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2025)

Is This Plan Necessarily Redundant? On the Computational Complexity of Unobserved Domain Learning

Pascal Bachor[1], P. Maurice Dekker[2], Gregor Behnke[2]

[1]Albert-Ludwigs-Universität Freiburg
[2]ILLC, Universiteit van Amsterdam
bachorp@cs.uni-freiburg.de, {p.m.dekker, g.behnke}@uva.nl

Abstract

Domain learning is the task of inferring actions' preconditions and effects (domains) from executed sequences of actions (plans) along with a varying detail of information about the corresponding world states. Remarkably, even if the state remains completely unobserved, as in this work, we can infer the existence of certain state features if we assume that the plans we learn from are non-redundant. Moreover, plans might be redundant regardless of the underlying domain.

We study the computational complexity of deciding whether there exists a domain in which a given plan is justified in the sense that either no single action (well-justification) or no set of actions (perfect justification) can be removed without violating correctness of the plan. We allow either arbitrarily large domains or domains with a polynomial bound on the number of state variables.

We show that the problem is in P for well-justified plans and arbitrary domains, NP-complete for well-justified plans and bounded domains, in coNP for perfectly justified plans and arbitrary domains, and in Σ_2^P for perfectly justified plans and bounded domains.

Introduction

AI planning asks to find a plan before it is to be executed by some agent – it is "thinking before acting" (Haslum 2006). Typically, an AI planner receives a symbolic model of the environment and is then tasked to derive a sequence of actions that, if executed, will transform the current state of the environment into a desired goal state. This symbolic model usually describes the preconditions and effects of actions. The entirety of all action descriptions is the *domain model* or *action model*. In order to apply modern AI planning methods, one needs such a domain model. Creating domain models is unfortunately difficult, error-prone, and usually requires both expertise in the application domain as well as in planning. This difficulty can make AI planning hard to apply in practice (Wang 1995; Benson 1996; Kambhampati 2007; Gregory and Lindsay 2016).

One way to remedy this problem is to try to automatically generate the required domain model. This can be done by learning from *traces*, i.e. from plans that have actually been executed in the environment. The task of *domain learning* is thus to infer a planning domain model from given traces. The main focus of past research was on developing practical methods for domain learning (see (Jiménez et al. 2012; Arora et al. 2018) for surveys). This has lead to a variety of approaches, but not to a substantiated theoretical understanding of the algorithmic properties of domain learning.

Recently, the investigation of the theoretical foundations of domain learning gained traction. At first, this mainly focused on *domain modification* – a setting in which we already have a tentatively correct domain model, but now observe a plan that does not agree with this domain model, i.e. the plan is not executable or not successful in the sense that it does not achieve the specified goal state. The task then is to modify the domain such that the new plan is executable and successful (Lin and Bercher 2021, 2023). First practical methods for solving this type of problem already exist (Lin, Grastien, and Bercher 2023; Lin, Höller, and Bercher 2024; Lin et al. 2025). Some past domain learning approaches also allow for partially specifying the model before learning, instead of learning the model fully anew (Zhuo and Kambhampati 2013; Kučera and Barták 2018). Similarly, *model reconciliation* asks to modify a given domain model such that it agrees with another model on the optimal, i.e. shortest, plan. This problem has recently been studied from a complexity point of view as well (Sreedharan, Bercher, and Kambhampati 2022).

While these theoretical investigations are related to domain learning, they do not cover the core task of domain learning: fully inferring a domain model from traces. Bachor and Behnke (2024) formally defined a decision problem for domain learning and studied its computational complexity. In their investigation they assumed *full observability* of the traces, i.e. that the complete state before and after each action execution is known. They further argued that in order to be able to correctly infer any preconditions and some effects, one must make additional assumptions on the given traces – in particular that the plans are in some sense rational. They formalized this using the notions of well-justification and perfect justification as introduced by Fink (1992).

In this paper, we expand upon this work and study the case of domain learning under unobservability of the environment. That means we are given the executed plans, but no information about the states that were visited. The learning task is thus twofold: to infer the structure of the state and to

infer the preconditions and effects of actions.

The structure of this paper is as follows. First, we will discuss related work. Subsequently, we will introduce basic definitions and notions of planning and domain learning and elaborate on the scope of our proposed formalism. Lastly, we will present the new complexity results we have obtained.

Related Work

Some domain learning methods and algorithms assume full observability of the state (Juba, Le, and Stern 2021), most often those working in an *online* setting (Lamanna et al. 2021; Walsh and Littman 2008; Wang 1996). This assumption, however, drastically limits the scope of what can be inferred. Notably, it does not allow for discovering a previously unknown structure of the domain to be learned, as such structure will already be visible in the observed state features. To remedy this, one can assume partial observability or no observability of the state. Domain learning for partially observable environments has been studied before (Yang, Wu, and Jiang 2007; Lamanna et al. 2025; Zhuo et al. 2010), but again only from a practical point of view. Domain learning has been studied in the epistemic logic community as well (Bolander, Gierasimczuk, and Liberman 2021), but there the setting is different, as the learning agent is the one choosing the action applied in the environment.

Some methods allow learning from traces in which only the actions are observed. The most common setting is to observe only the initial state (and sometimes the goal state) and the executed actions (Zhuo and Kambhampati 2013; Kučera and Barták 2018; Aineto, Celorrio, and Onaindia 2019). Some approaches can also handle noisy traces in which only some actions are observed (Zhuo and Kambhampati 2013). Commonly, these methods utilize algorithms for NP- or PSPACE-complete problems such as maximum satisfiability (Yang, Wu, and Jiang 2007), inference in graphical models (Zhuo and Kambhampati 2013), or planning itself (Aineto, Jiménez, and Onaindia 2018; Aineto, Celorrio, and Onaindia 2019). The system LOUGA instead uses an evolutionary algorithm, but to tackle a similarly complex decision problem (Kučera and Barták 2018). These practical approaches thus align well with the result we will present in this paper, showing that the involved decision problems are computationally hard.

The setting in which no state can be observed (not even the initial state or the goal state), but where we still receive exact information about the actions being performed, has most prominently been addressed by the learning systems LOCM (Learning Object-Centred Models) and LOCM2 (Cresswell, McCluskey, and West 2009; Cresswell and Gregory 2011; Cresswell, McCluskey, and West 2013). In order to infer useful information about the structure of the state, LOCM assumes that the actions still exhibit parameters, i.e. the given actions are of the form $\mathtt{a}(x_0, \ldots, x_n)$. LOCM can then trace the occurrences of objects throughout the plan and can stipulate the existence of predicates relating to these objects.

Bonet and Geffner (2020) proposed a different kind of setting in which state features cannot be observed but instead we know when two states are equivalent. The task then is to learn a domain whose state space matches a given transition system of states.

The notion of perfect justification has also been adopted by Salerno, Fuentetaja, and Seipp (2023). And while well-justification and perfect justification are not used as often as, say, optimality, the same concept can also appear under a different name such as *action elimination* (Nakhost and Müller 2010) or *plan optimization* (Chrpa, McCluskey, and Osborne 2012; Med and Chrpa 2022).

Finally, in our analysis of the search space of candidate domains, there are notable similarities with previous work. The notions of a *safe action model* (Juba, Le, and Stern 2021) correspond to our notion of a most restrictive domain (albeit with differences regarding the observability), Aineto and Scala (2024) put forth the notion of a *sound domain* and also observe a partial order of restrictiveness for domains, and Gösgens, Jansen, and Geffner (2025) recently proposed to consider the set of all predicates (or *features*) that are consistent with a given set of plan traces.

Problem Definition

We adopt the framework introduced by Bachor and Behnke (2024), which is in turn based on the classical planning formalism STRIPS (Fikes and Nilsson 1971). The environment *state* is modeled using a set of propositional *variables* (or *state variables*) that can either be contained (true) or missing (false) in a state. Actions are characterized by their *preconditions*, variables that must be true in–, *deletions*, variables that will be removed from–, and *additions*, variables that will be added to a state in which the action is executed. A *domain* is the definition of actions' preconditions and *effects* (i.e. deletions and additions). Note that we do not allow negative preconditions.

Definition 1. *$\delta = (\mathrm{pre}, \mathrm{del}, \mathrm{add})$, where $\mathrm{pre} = \delta_{\mathbf{pre}}$, $\mathrm{del} = \delta_{\mathbf{del}}$, and $\mathrm{add} = \delta_{\mathbf{add}}$ are relations between actions and variables, is a domain if $(\mathrm{pre} \cup \mathrm{del}) \cap \mathrm{add} = \emptyset$.*

A relation rel between actions and variables is a set of pairs (a, x) such that a is an action and x is a variable. We let $\mathrm{rel}(a) \coloneqq \{\, x \mid (a, x) \in \mathrm{rel} \,\}$ and $\mathrm{rel}^{-1}(x) \coloneqq \{\, a \mid (a, x) \in \mathrm{rel} \,\}$. The condition $(\mathrm{pre} \cup \mathrm{del}) \cap \mathrm{add} = \emptyset$ rules out contradictory or redundant effects. Namely, both deleting and adding a variable and adding a variable that is necessarily already present.

We let $\mathcal{V}(\delta) \coloneqq \{\, x \mid \exists a.(a, x) \in \delta_{\mathbf{add}} \,\}$ denote the *variables of* δ. Here we disregard variables that are never added and therefore always false.

Definition 2. *A state is a set of variables. The application of the action a to the state X in the domain δ, denoted $\mathrm{app}(\delta, X, a)$, is defined as $(X \setminus \delta_{\mathbf{del}}(a)) \cup \delta_{\mathbf{add}}(a)$ if $\delta_{\mathbf{pre}}(a) \subseteq X$, else it is undefined.*

Planning

The planning problem is usually posed as follows. Given a domain, find a sequence of actions (a *plan*) that, when starting in a given *initial state*, yields a state that contains all variables in a given set of *goal variables*. Such plan is said to be *successful*.

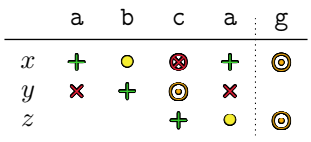


Figure 1: Example trace of $\pi = (\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{a}, \mathtt{g})$ in the domain δ with $\delta_{\mathbf{pre}} = \{ (\mathtt{c}, x), (\mathtt{c}, y), (\mathtt{g}, x), (\mathtt{g}, z) \}$, $\delta_{\mathbf{del}} = \{ (\mathtt{a}, y), (\mathtt{c}, x) \}$, and $\delta_{\mathbf{add}} = \{ (\mathtt{a}, x), (\mathtt{b}, y), (\mathtt{c}, z) \}$.

	The variable is *present*	*a precondition*	*deleted*	*added*
	yes	no	no	no
	yes	yes	no	no
	no	no	yes	no
	no	yes	yes	no
	yes	no	no	yes

Figure 2: Legend.

In this work, instead of an initial state, we assume that the state may (optionally) be initialized by a dedicated *initialization action* at the beginning of a plan; and instead of goal variables we assume there exists a dedicated *goal action* that has the appropriate variables as preconditions. We will assume that before the first (potentially initializing) action of a plan is executed, all variables are false and that the final action of a successful plan is the goal action. This technique has been used before, e.g. by Yang, Wu, and Jiang (2007). In this way we not only reduce the technical overhead but also ensure *full* unobservability in the sense that the variables in the initial and goal states are unknown as well.

Definition 3. *A plan $\pi = (\pi_i)_i$ is a sequence of actions. A trace $\tau = (\pi, \nu)$ is a plan $\pi = \tau_{\boldsymbol{\pi}}$ along with a sequence of states $\nu = \tau_{\boldsymbol{\nu}}$ such that $|\tau| := |\pi| = |\nu|$.*

Note that $(\sigma_i)_i$ is shorthand for $(\sigma_i)_{i=0}^{|\sigma|-1}$ and that we let $\sigma_{-i} := \sigma_{|\sigma|-i}$, where $|\sigma|$ denotes the length of the sequence σ. Furthermore, $\pi_{\neq i} := (\pi_0, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_{-1})$ and $\sigma^\frown\rho := (\sigma_0, \ldots, \sigma_{-1}, \rho_0, \ldots, \rho_{-1})$ denotes the concatenation of σ and ρ.

Definition 4. *The trace of the plan π in the domain δ, denoted $T(\delta, \pi)$, is the unique trace (π, ν) such that*

$$\nu_0 = \mathrm{app}(\delta, \emptyset, \pi_0)$$
$$\nu_{i+1} = \mathrm{app}(\delta, \nu_i, \pi_{i+1})$$

if all ν_i are defined, else it is undefined. The set of valid plans in the domain δ is

$$\Pi(\delta) := \{ \pi \mid T(\delta, \pi) \text{ is defined} \}.$$

An example trace is shown in Figure 1. Consult Figure 2 for an overview of the symbols we use in such diagrams throughout this paper.

We can now formally define the planning problem.

Definition 5 (Planning)**.** *Given a domain δ and a* goal action g*, find a plan $\pi \in \Pi(\delta)$ such that $\pi_{-1} = \mathtt{g}$.*

The problem is PSPACE-complete in general and NP-complete when limited to plans of length at most k, where k is polynomial in the size of the input (Bylander 1994).

Plan Justification

We are usually interested in finding a plan that is not only successful in reaching the goal but also does so quickly and without detours. We will formally define two such additional qualities, namely well-justification and perfect justification. We adopt both notions from Fink and Yang (Fink 1992; Fink and Yang 1992, 1993).

Definition 6. *$\sigma = (\sigma_i)_i$ is a subsequence of $\rho = (\rho_i)_i$ if $\sigma = (\rho_{i_j})_j$ for some $(i_j)_j$ with $i_j < i_{j+1}$. σ is a proper subsequence of ρ if σ is a subsequence of ρ and $\sigma \neq \rho$.*

For example, $(1, 1)$ and $(2, 3, 1)$ are proper subsequences of $\rho = (1, 2, 3, 2, 1, 3)$, but $(3, 1, 2)$ is not a subsequence of ρ.

Definition 7.

(i) $\pi' \prec \pi$ *if* $\pi' = \pi_{\neq i}$ *for some* $i < |\pi| - 1$.

(ii) $\pi' \lll \pi$ *if* π' *is a proper subsequence of* π *such that* $\pi'_{-1} = \pi_{-1}$.

A plan $\pi' \prec \pi$ contains all but a single action of π. A plan $\pi' \lll \pi$ contains all but arbitrarily many actions of π. In both cases the final action (goal action) π_{-1} must not be missing.

Definition 8. *A plan $\pi \in \Pi(\delta)$ is well-justified (perfectly justified) in the domain δ if there exists no $\pi' \in \Pi(\delta)$ such that $\pi' \prec \pi$ ($\pi' \lll \pi$).*

Remark. *If $\pi' \prec \pi$, then $\pi' \lll \pi$ and therefore perfect justification implies well-justification.*

For a plan π, the number of plans $\pi' \prec \pi$ is at most linear in $|\pi|$, but the number of plans $\pi' \lll \pi$ can be exponential in $|\pi|$. In fact, we can efficiently test a plan for well-justification, but perfect justification cannot be checked efficiently unless P = NP (Fink and Yang 1992).

Lemma 1. *Given a plan π and a domain δ, deciding whether π is well-justified in δ is in* P. □

Lemma 2. *Given a plan π and a domain δ, deciding whether π is perfectly justified in δ is* coNP-complete. □

Note that Fink and Yang claim NP-completeness when in fact they show coNP-completeness (note how a plan $\pi' \lll \pi$, if valid, witnesses that π is *not* perfectly justified).

Domain Learning

Domain learning is *inverse* to planning: instead of searching for a plan that is consistent with a given domain, we are searching for a domain that is consistent with a given plan. In a fully observed setting, the domain to be found must also be compatible with the states observed before and after each action and cannot use any additional unobserved variables (Bachor and Behnke 2024). In the unobserved setting, arbitrary variables can be used. However, we allow to *bound* the domains in the sense of limiting the total number of state variables. With such bounds one can ensure that learned domains stay within practical limits.

Definition 9. *The set of domains (with k variables, for $k \in \mathbb{N}$) that are consistent with a plan π is*

$$\Delta^*(\pi) := \Pi^{-1}(\pi) = \bigcup_{k \in \mathbb{N}} \Delta^k(\pi)$$
$$\Delta^k(\pi) := \{ \delta \mid \pi \in \Pi(\delta) \text{ and } |\mathcal{V}(\delta)| \leq k \}.$$

Note that finding *any* domain that is consistent with a given plan (using any number of state variables) is trivial: if there are no preconditions, any plan will be valid. However, in such a domain the given plan would not only be wholly redundant, as the goal action could be executed immediately, but it would also be devoid of information, as any other plan would be valid as well. We should therefore assume otherwise, namely that the given plan is not redundant. This entails that the given plan carries information about certain other plans being invalid. Such non-redundancy constraints are captured by the notions of well-justification and perfect justification.

Definition 10 (Unobserved Domain Learning)**.** *Given a plan π and $k \in \mathbb{N} \cup \{*\}$, find a domain $\delta \in \Delta^k(\pi)$ in which π is well-justified (perfectly justified).*

To illustrate our problem definition and see how presupposing justification allows inferring preconditions and effects, consider the following example of solving a domain learning problem (see also Figure 3).

Let $\pi = (\mathtt{i}, \mathtt{a}, \mathtt{b}, \mathtt{a}, \mathtt{g})$. Note that the actions in the given sequence are bare symbols carrying no information whatsoever about preconditions or effects. The first action i can be seen as an initialization action. It's additions fully determine the state from which the remainder of the sequence is executed. The last action g represents the plan's goal. Whenever g can be executed, which means that it's preconditions are satisfied, the goal has been reached.

Now, if π is assumed to be well-justified, it follows that at every step i some variable must be added that is a precondition at some later step $j > i$, as otherwise the action at step i would be redundant. Using as few variables as possible to fulfill this requirement, we can construct a domain that comprises two variables x and y such that (i) x is added by i and b and required by a and (ii) y is added by a and required by b and g. For the plan to be well-justified in this domain, it must also hold that b deletes y, as otherwise the second occurrence of a would be redundant (adding only a variable that is already present). Similarly, a must delete x, as otherwise b would be redundant. This suffices to ensure that π is well-justified. However, π is not yet perfectly justified, as $(\mathtt{i}, \mathtt{a}, \mathtt{g}) \lessdot \pi$ is executable and therefore constitutes a counterexample. To ensure perfect justification we may add another variable z that is added by b and required by g.

An example for a plan that is not well-justified (or perfectly justified) in any domain is $\pi = (\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{a}, \mathtt{b}, \mathtt{a}, \mathtt{c}, \mathtt{g})$. Here, the first occurrence of c is necessarily redundant. In fact, even $(\mathtt{a}, \mathtt{b}, \mathtt{a}, \mathtt{c}, \mathtt{g})$ is a valid plan in any domain that is consistent with π. We will see how such facts can be proven over the course of this paper.

More General Settings of Domain Learning

In most practical cases, domain learning presupposes a more general setting than we investigate in this paper.

First, a common assumption is that we are learning from more than one observed plan, i.e. we are given a set of plans or traces and have to find a domain that is compatible with all of them. In our case this would mean that in that domain also all of the given plans have to be justified. One notable exception to this is LOCM (Cresswell, McCluskey, and West 2013), which learns only from a single plan. All of our complexity results also hold for learning from multiple plans. Learning from a single plan is a special case of learning from a set of plans and any hardness result translates to the more general case. The membership proofs can be easily adapted to treat a set of plans instead of a single plan.

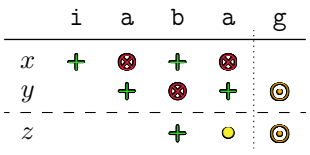


Figure 3: The plan $\pi = (\mathtt{i}, \mathtt{a}, \mathtt{b}, \mathtt{a}, \mathtt{g})$ in a domain $\delta \in \Delta^3(\pi)$ in which it is perfectly justified. If we restrict the domain to the variables x and y, the plan is still well-justified but no longer perfectly justified.

Second, it is usually assumed that the given actions are more than bare symbols (i.e. just names). Notably, domain learning systems often assume PDDL-style actions with parameters such as $\mathtt{a}(x)$ or $\mathtt{b}(y, x)$. We then have more information about the task at hand, namely the objects occurring as parameters, but we also have to solve a potentially harder problem. In this setting, if an action occurs twice with different parameters, then the preconditions and effects of these two actions must be *congruent*. For example, if $\mathtt{b}(y, x)$ has the precondition $P(y)$, then $\mathtt{b}(z, v)$ must have the precondition $P(z)$ — whereas, in our setting, we would regard these two actions as wholly different and could choose distinct preconditions and effects.

In this paper, we restrict ourselves to studying the more simple STRIPS-style setting, as (1) any hardness result obtained here carries over to the PDDL-style case and (2) we can analyze characteristics of domain learning problems more concisely in this setting, which can form the basis for a later analysis of the more complex PDDL-style setting.

Unbounded Domains

We will proceed with the base case in which there are no restrictions on the domain to be found.

While domains can be regarded as a collection of otherwise independent actions each of which requires, deletes, and adds some variables, the fact that the variables are unknown invites us to assume a more *variable-centric* perspective: A domain can be regarded as a collection of otherwise independent variables each of which is required, deleted, and added by some actions. The invalidity of a plan will always be *witnessed* by some variable, a precondition that is not satisfied at some point. Conversely, for any variable there are some plans that the variable witnesses to be invalid. A variable-centric perspective has also been assumed, for example, in the work of Shaik and van de Pol (2022), where a QBF encoding a planning problem will be valid if and only if no variable is missing at some point at which it is a precondition.

Now, because the number of variables is unbounded, the domain to be found can include any variable that is consis-

tent with the given plan π. This means that we are able to disallow *any* plan π' for which there exists some state variable that witnesses that π' is invalid and does not witness that π is invalid. Therefore, for any plan π there exist *most restrictive* domains in the sense that they invalidate any plan that can potentially be invalidated while simultaneously being consistent with π.

Definition 11. *Let δ and δ' be domains. We say δ is more restrictive than δ', denoted $\delta \leq \delta'$, if $\Pi(\delta) \subseteq \Pi(\delta')$.*

Note that $\leq$ is a coarsening of the order defined in (Bachor and Behnke 2024, Definition 11). We let $\min \Delta$ denote the set of most restrictive domains of Δ, i.e. the set of domains that are $\leq$-minimal in Δ.

Lemma 3. *If π is well-justified (perfectly justified) in some domain δ, then π is well-justified (perfectly justified) in any domain $\delta' \in \Delta^*(\pi)$ with $\delta' \leq \delta$ and in particular in any $\delta' \in \min \Delta^*(\pi)$.*

Proof. $\Pi(\delta') \subseteq \Pi(\delta)$ and therefore any counterexample to the well-justification (perfect justification) of π in δ' is a counterexample to the well-justification (perfect justification) of π in δ. □

To find a domain in which a given plan π is justified, we can therefore search for a most restrictive domain and check whether π is justified in that domain. To construct a most restrictive domain we can simply enumerate all variables that are consistent with π. However, there might be exponentially many such variables and not all of them are required for the domain to be most restrictive. So can we construct a most restrictive domain using only polynomially many state variables? We answer this question negatively, showing that there exist plans for which the size of any most restrictive domain is exponential in the length of the plan.

Lemma 4. *There exists a sequence of plans $(\pi^{(n)})_{n\in\mathbb{N}}$ such that $|\pi^{(n)}| \in \mathcal{O}(n^2)$, whereas $|\mathcal{V}(\delta^{(n)})| \in \Omega(2^n)$ for any $\delta^{(n)} \in \min \Delta^*(\pi^{(n)})$.*

Proof. We use the actions $\mathtt{r}$ and $\mathtt{p}_i, \mathtt{q}_i$ for each $i < n$ and let $\Upsilon := \bigcup_{i<n} \{ \mathtt{p}_i, \mathtt{q}_i \}$. Let $\pi^{(n)} = \alpha^\frown\beta^\frown\gamma$, where

$$\varepsilon = \left(\bigcirc_{a\in\Upsilon}(a)\right)^\frown(\mathtt{r})$$
$$\alpha = \bigcirc_{a\in\Upsilon}\varepsilon^\frown(a)$$
$$\beta = \bigcirc_{a,b\in\Upsilon}(a,b)$$
$$\gamma = \bigcirc_{i<n}\varepsilon^\frown(\mathtt{p}_i,\mathtt{q}_i,\mathtt{r})^\frown\varepsilon^\frown(\mathtt{q}_i,\mathtt{p}_i,\mathtt{r}).$$

Let $\delta^{(n)} \in \min \Delta^*(\pi^{(n)})$. W.l.o.g. we assume that no two variables are required, deleted, and added by the same actions.

Claim: $\varepsilon^\frown\sigma \in \Pi(\delta^{(n)})$ for any $\sigma = (\sigma_j)_{j=0}^l$ with $\sigma_j \in \Upsilon$.

$\underline{l = 0}$: ε is a prefix of $\pi^{(n)}$ and therefore executable. Because ε contains all actions, all plans with suffix ε have the same final state. In this state we can execute any $\sigma_0 \in \Upsilon$ as seen in α.

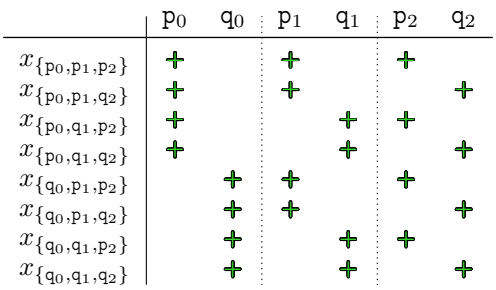


Figure 4: Variables x_ϱ for $n = 3$ as in the proof of Lemma 4. We require both $\mathtt{p}_i$ and $\mathtt{q}_i$ for some i to add all the x_ϱ.

$\underline{l = k+1}$: Applying the claim for $l = k$, we know that both $\varepsilon^\frown(\sigma_0,\dots,\sigma_k)$ and $\varepsilon^\frown(\sigma_0,\dots,\sigma_{k-1},\sigma_{k+1})$ are executable. Therefore, all preconditions of σ_{k+1} are satisfied after executing $\varepsilon^\frown(\sigma_0,\dots,\sigma_{k-1})$. Furthermore, because σ_{k+1} occurs directly after σ_k in β, we know that σ_k does not delete any precondition of σ_{k+1}. It follows that $\varepsilon^\frown(\sigma_0,\dots,\sigma_{k+1})$ is executable.

Claim: For each $\varrho \subseteq \Upsilon$ with $\mathtt{p}_i \in \varrho \iff \mathtt{q}_i \notin \varrho$ the variable x_ϱ with $\mathrm{pre}^{-1}(x_\varrho) = \mathrm{del}^{-1}(x_\varrho) = \{\mathtt{r}\}$ and $\mathrm{add}^{-1}(x_\varrho) = \varrho$ is a variable of $\delta^{(n)}$.

See Figure 4 for an example of the variables x_ϱ.

Let $\overline{\varrho} := \Upsilon \setminus \varrho$ and $\vartheta := \varepsilon^\frown\bigcirc_{a\in\overline{\varrho}}(a)$. Because $\delta^{(n)}$ is most restrictive w.r.t. $\pi^{(n)}$, $\pi^{(n)}$ is consistent with x_ϱ, and x_ϱ ensures that $\vartheta^\frown(\mathtt{r})$ is not executable, it holds $\vartheta^\frown(\mathtt{r}) \notin \Pi(\delta^{(n)})$.

We know that $\vartheta \in \Pi(\delta^{(n)})$ and that no precondition of $\mathtt{r}$ can be deleted by a $\mathtt{p}_i$ or $\mathtt{q}_i$ because $\mathtt{r}$ occurs directly after all of these actions in γ. This means that there must be some precondition x_0 of $\mathtt{r}$ that is deleted by $\mathtt{r}$ and not added by any of the $\overline{\varrho}$. Furthermore, x_0 must be added by either $\mathtt{p}_i$ or $\mathtt{q}_i$ for all i to allow $\varepsilon^\frown(\mathtt{p}_i,\mathtt{q}_i,\mathtt{r}) \in \Pi(\delta^{(n)})$ as seen in γ. It follows $x_0 = x_\varrho \in \mathcal{V}(\delta^{(n)})$.

$$|\pi^{(n)}| = 16n^2 + 12n.$$
$$|\mathcal{V}(\delta^{(n)})| \geq |\{\, \varrho \subseteq \Upsilon \mid \mathtt{p}_i \in \varrho \iff \mathtt{q}_i \notin \varrho \,\}| = 2^n. \quad \square$$

Although constructing a domain that allows a given π and disallows as many $\pi' \neq \pi$ as possible might require superpolynomial time and space, we show that a domain that allows a given π and disallows a single given π' can be constructed efficiently, assuming such a domain exists. This has consequences for the domain learning problem insofar as we are able to decide whether a plan π' is a counterexample to the justification of another plan π in polynomial time.

Lemma 5. *Given plans π and π', deciding whether there exists a domain δ such that $\pi \in \Pi(\delta)$ and $\pi' \notin \Pi(\delta)$ is in* P.

Proof. If π' is not executable, there must exist $i, j \in \mathbb{Z}$ with $-1 \leq i < j < |\pi'|$ and a state variable $x_{i,j}$ such that

$$x_{i,j} \in \mathrm{del}(\pi'_i) \text{ if } i \neq -1 \tag{1}$$
$$x_{i,j} \notin \mathrm{add}(\pi'_k) \text{ for } i \leq k \leq j \tag{2}$$
$$x_{i,j} \in \mathrm{pre}(\pi'_j). \tag{3}$$

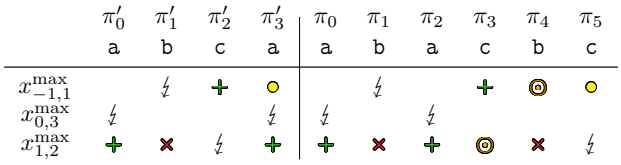


Figure 5: Three example $x^{\max}_{i,j}$ as in the proof of Lemma 5 for $\pi = (\mathtt{a}, \mathtt{b}, \mathtt{a}, \mathtt{c}, \mathtt{b}, \mathtt{c})$ and $\pi' = (\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{a})$. $\lightning$ denotes that a precondition is not satisfied. There does not exist a domain in which π is a valid plan and π' is not.

$x_{i,j}$ will be missing at point i of π' (possibly from the very beginning if $i = -1$) and not added until point j of π' at which it will be a precondition. Thus, $x_{i,j}$ witnesses that π' is not executable. Conversely, any witness of the invalidity of π' must satisfy propositions (1) to (3) for some i, j.

$x_{i,j}$ is only partially constrained and there can be exponentially many such witnesses, some (and possibly all) of which will also make π invalid. However, for given i, j there exists a unique variable $x^{\max}_{i,j}$ that is *least restrictive* among the $x_{i,j}$ in the sense that π is consistent with $x^{\max}_{i,j}$ if and only if it is consistent with any $x_{i,j}$. This is achieved by minimizing preconditions and deletions and maximizing additions as follows. Let

$$x^{\max}_{i,j} \in \mathrm{del}(a) \iff a = \pi'_i \text{ and } i \neq -1 \tag{4}$$

$$x^{\max}_{i,j} \notin \mathrm{add}(a) \iff a = \pi'_k \text{ for some } i \leq k \leq j \tag{5}$$

$$x^{\max}_{i,j} \in \mathrm{pre}(a) \iff a = \pi'_j \tag{6}$$

for all actions a. See Figure 5 for examples.

Therefore, to decide whether there exists a domain in which π is valid and π' is invalid, it suffices to check whether this is fulfilled by any of the $x^{\max}_{i,j}$. The number of $x^{\max}_{i,j}$ is quadratic in the length of π'. □

Well-Justified Plans

The following results promptly follow from Lemma 5.

Theorem 1. *Given a plan π, deciding whether there exists a domain $\delta \in \Delta^*(\pi)$ in which π is well-justified is in* P.

Proof. $\pi' \prec \pi$ is a counterexample to the well-justification of π if and only if there exists no domain in which π is valid and π' is not. The existence of such a domain can be checked in polynomial time using Lemma 5. The number of $\pi' \prec \pi$ is linear in $|\pi|$. □

Because a single variable suffices to ensure that a plan is not executable and there are at most $|\pi| - 1$ plans $\pi' \prec \pi$, we obtain the following corollary.

Corollary 1. *If there exists a domain $\delta \in \Delta^*(\pi)$ in which π is well-justified, then there exists a domain $\delta' \in \Delta^{|\pi|-1}(\pi)$ in which π is well-justified.*

Perfectly Justified Plans

We conjecture that in general an exponential number of state variables is required not only to make a domain most restrictive but also to ensure that a plan is perfectly justified. However, thanks to Lemma 5, it suffices to consider polynomially many variables for any given $\pi' \lessdot \pi$.

Theorem 2. *Given a plan π, deciding whether there exists a domain $\delta \in \Delta^*(\pi)$ in which π is perfectly justified is in* coNP.

Proof. Guess a $\pi' \lessdot \pi$. π' is a counterexample to the perfect justification of π if and only if there exists no domain in which π is valid and π' is not. This can be checked in polynomial time using Lemma 5. □

Bounded Domains

Having an upper bound on the number of state variables allows to constrain domains to be of size polynomial in the length of the given plan. Interestingly, this increases the complexity of domain learning. While in the unbounded case we can simply assume that any state variable that is consistent with the given plan is included in the domain, in the bounded case all of a domain's restrictions on potential plans must be expressed using a limited number of variables.

Well-Justified Plans

The following result can be regarded as an alternative definition of well-justification and will be important in the proof of the subsequent Theorem 3. It demonstrates the meaning of well-justification: at every step $i < |\pi| - 1$ some variable x_i must be added such that x_i would otherwise not be present at some later point $p_i > i$ at which x_i is a precondition. See Figure 6 for an illustration.

Lemma 6. *A plan π is well-justified in δ if and only if for every $i < |\pi| - 1$ there exist $x_i \in \mathcal{V}(\delta)$ and $d_i, p_i \in \mathbb{Z}$ with $-1 \leq d_i < i < p_i < |\pi|$ such that*

$$x_i \in \mathrm{del}(\pi_{d_i}) \text{ if } d_i \neq -1 \tag{1}$$

$$x_i \notin \mathrm{del}(\pi_j) \cup \mathrm{add}(\pi_j) \text{ for } d_i < j < i \tag{2}$$

$$x_i \in \mathrm{add}(\pi_i) \tag{3}$$

$$x_i \notin \mathrm{del}(\pi_j) \cup \mathrm{add}(\pi_j) \text{ for } i < j < p_i \tag{4}$$

$$x_i \in \mathrm{pre}(\pi_{p_i}). \tag{5}$$

The edge case $d_i = -1$ occurs if x_i is not modified at all before point i.

Fink and Yang use similar but less explicit constraints in their definition of well-justification. Their notion of *establishment* is weaker and captures only propositions (3) to (5).

Proof. $\Longrightarrow$ If π is well-justified, it holds $\pi \in \Pi(\delta)$ and $\pi_{\neq i} \notin \Pi(\delta)$ for all $i < |\pi| - 1$. This means that for all such i some precondition x_i that occurs after point i, i.e. $x_i \in \mathrm{pre}(\pi_{p_i})$ for some $p_i > i$, is violated in $\pi_{\neq i}$ — (5). This precondition would not be violated if the action number i were not missing, so we know that $x_i \in \mathrm{add}(\pi_i)$ and that no action between i and p_i modifies x_i — (3) and (4). Furthermore, x_i must not be present when π_i is executed, so

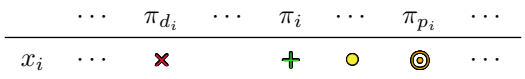


Figure 6: Trace of a variable x_i as in Lemma 6. x_i is deleted by π_{d_i}, added by π_i and required by π_{p_i}. Between d_i and p_i no action other than π_i modifies x_i.

it is either never modified, and in particular never added, before i (case $d_i = -1$) or modified, and in particular deleted, at some point $d_i < i$ for the last time before i — corresponding to propositions (1) and (2).

$\Longleftarrow$ Propositions (1) to (5) imply $\pi_{\neq i} \notin \Pi(\delta)$. □

Theorem 3. *Given a plan π and a number k that is polynomial in the size of the input, deciding whether there exists a domain $\delta \in \Delta^k(\pi)$ in which π is well-justified is* NP-complete.

Proof. Membership: Guess a domain with k variables and check whether π is well-justified (Lemma 1).

Hardness: Given an undirected graph $G = (V, E)$, we construct a plan π such that G is 3-colorable if and only if there exists a domain $\delta \in \Delta^3(\pi)$ in which π is well-justified. We use the actions $\mathtt{x}_v, \mathtt{a}_v, \mathtt{b}_v, \mathtt{c}_v$ for each $v \in V$, and $\mathtt{p}_e, \mathtt{q}_e, \mathtt{r}_e$ for each $e \in E$.

Let

$$\pi = \frown_{v \in V} (\mathtt{x}_v, \mathtt{a}_v, \mathtt{c}_v, \mathtt{a}_v, \mathtt{b}_v, \mathtt{c}_v, \mathtt{b}_v) \frown \frown_{(u,v) \in E} (\mathtt{p}_{(u,v)}, \mathtt{q}_{(u,v)}, \mathtt{a}_u, \mathtt{b}_u, \mathtt{r}_{(u,v)}, \mathtt{c}_v).$$

π is designed such that for each vertex v the preconditions and effects of all actions in $(\mathtt{x}_v, \mathtt{a}_v, \mathtt{c}_v, \mathtt{a}_v, \mathtt{b}_v, \mathtt{c}_v, \mathtt{b}_v)$ are fully determined up to the permutation of the three state variables. We will use these permutations to encode a vertex coloring. In particular, for each vertex v, the color $\mathfrak{c}(v)$ of v will be the (single) state variable that is added by $\mathtt{c}_v$. It will then be shown that $(\mathtt{a}_u, \mathtt{b}_u, \mathtt{r}_{(u,v)}, \mathtt{c}_v)$, i.e. the plan component corresponding to the edge (u, v), is non-redundant only if $\mathfrak{c}(u) \neq \mathfrak{c}(v)$.

Claim: If π is well-justified in $\delta \in \Delta^3(\pi)$, then for all $v \in V$ there exists a variable c_v such that

$$\mathrm{add}(\mathtt{a}_v) \cup \mathrm{add}(\mathtt{b}_v) = \mathcal{V}(\delta) \setminus \{ c_v \} \quad (1)$$

$$\mathrm{del}(\mathtt{a}_v) \cup \mathrm{del}(\mathtt{b}_v) = \{ c_v \} \quad (2)$$

$$\mathrm{pre}(\mathtt{c}_v) = \mathcal{V}(\delta) \setminus \{ c_v \} \quad (3)$$

$$\mathrm{add}(\mathtt{c}_v) = \{ c_v \}. \quad (4)$$

$\mathtt{c}_v$ must add some variable c_v. Any action must add some variable that is neither added by the action just before nor by the action just behind. Therefore, if we consider the segment $(\mathtt{c}_v, \mathtt{a}_v, \mathtt{b}_v, \mathtt{c}_v)$ of π, we can infer that there must exist some variable $a_v \in \mathrm{add}(\mathtt{a}_v)$ ($b_v \in \mathrm{add}(\mathtt{b}_v)$) that is added by neither $\mathtt{c}_v$ nor $\mathtt{b}_v$ ($\mathtt{a}_v$). Hence, $\mathcal{V}(\delta) = \{ a_v, b_v, c_v \}$ and no further additions are possible. It follow (1) and (4).

The segment $(\mathtt{a}_v, \mathtt{c}_v, \mathtt{a}_v)$ witnesses $a_v \in \mathrm{pre}(\mathtt{c}_v)$, as otherwise the first occurrence of $\mathtt{a}_v$ would be redundant. Similarly, $(\mathtt{b}_v, \mathtt{c}_v, \mathtt{b}_v)$ witnesses $b_v \in \mathrm{pre}(\mathtt{c}_v)$. It follows (3).

It holds $c_v \in \mathrm{del}(\mathtt{a}_v) \cup \mathrm{del}(\mathtt{b}_v)$, as otherwise the second occurrence of $\mathtt{c}_v$ in the segment $(\mathtt{c}_v, \mathtt{a}_v, \mathtt{b}_v, \mathtt{c}_v)$ would be redundant. $b_v \notin \mathrm{del}(\mathtt{a}_v)$ ($a_v \notin \mathrm{del}(\mathtt{b}_v)$) follows from the fact that $b_v \in \mathrm{pre}(\mathtt{c}_v)$ ($a_v \in \mathrm{pre}(\mathtt{c}_v)$) and $\mathtt{c}_v$ is executed just after $\mathtt{a}_v$ ($\mathtt{b}_v$). It follows (2).

Claim: If there exists a domain $\delta \in \Delta^3(\pi)$ in which π is well-justified, then there exists a 3-coloring $\mathfrak{c} : V \to \mathcal{V}(\delta)$ of G.

Using the previous result, we let $\mathfrak{c}$ be such that $v \mapsto c_v$. Suppose $\mathfrak{c}$ is not a valid coloring and π is well-justified. Then there exists an edge $(u, v) \in E$ with $c_u = c_v$ and $(\mathtt{a}_u, \mathtt{b}_u, \mathtt{r}_{(u,v)}, \mathtt{c}_v)$ is an infix of π. We know that executing $\mathtt{a}_u$ followed by $\mathtt{b}_u$ yields the state $\mathcal{V}(\delta) \setminus \{ c_u \}$ due to propositions (1) and (2). Furthermore, we know from propositions (3) and (4) that the state to which $\mathtt{c}_v$ is applied must be $\mathcal{V}(\delta) \setminus \{ c_v \}$ in order for $\mathtt{c}_v$ to be executable and non-redundant. Because $c_u = c_v$, these two states are equivalent such that $\mathtt{r}_{(u,v)}$ cannot have any effect and is therefore redundant. Then π is not well-justified. *(Contradiction.)*

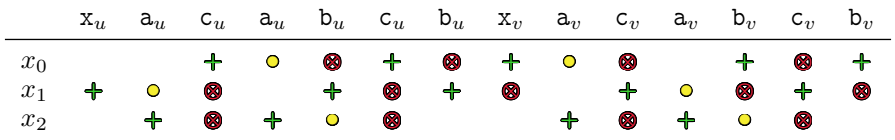
Claim: If there exists a 3-coloring $\mathfrak{c} : V \to \{ 0, 1, 2 \}$ of G, then there exists a domain $\delta \in \Delta^3(\pi)$ in which π is well-justified.

An example trace of π in δ is shown in Figure 7.

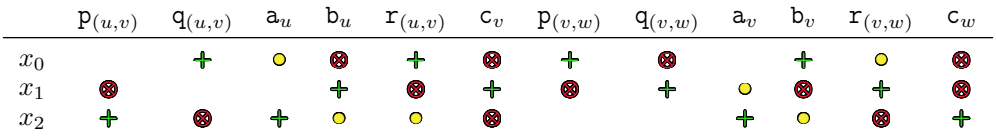
Let $V = \{ v_0, v_1, \dots \}$ and $E = \{ e_0, e_1, \dots \}$ with the order used in the construction of π. We construct δ such that every action adds exactly one variable. Furthermore, a variable will be deleted by an action if and only if the variable is also a precondition of this action, i.e. $\mathrm{pre} = \mathrm{del}$. If an action requires and deletes a variable, we say the action *consumes* the variable. Every action will consume exactly one variable except for the actions $\mathtt{x}_{v_0}$ and $\mathtt{a}_{v_i}$, which will not consume a variable and the actions $\mathtt{c}_{v_i}$, which will consume two variables. The domain will be constructed in such a way that the variable that is added at each step $i < |\pi| - 1$ will be subsequently consumed without having been modified by another action in between. This ensures that π is well-justified (Lemma 6).

We let $\mathcal{V}(\delta) = \{ x_0, x_1, x_2 \}$ and let $X^{\complement}$ denote the complement of X w.r.t. $\mathcal{V}(\delta)$, i.e. $X^{\complement} := \mathcal{V}(\delta) \setminus X$. Let $c_{v_i} = x_{\mathfrak{c}(v_i)}$ for all $v_i \in V$. Let $b_0 \in \{ c_{v_0} \}^{\complement}$ and let $b_{i+1} \in \{ b_i, c_{v_{i+1}} \}^{\complement}$. Let $a_i \in \{ b_i, c_{v_i} \}^{\complement}$. We let $\mathtt{x}_{v_i}$ add b_i. We let $\mathtt{x}_{v_{i+1}}$ consume b_i. We let $\mathtt{a}_{v_i}$ add a_i. We let $\mathtt{c}_{v_i}$ add c_i and consume a_i and b_i. We let $\mathtt{b}_{v_i}$ add b_i and consume c_i. Let $p_{(u,v)} \in \{ b_{-1}, c_u \}^{\complement}$ for $(u, v) = e_0$ and let $p_{(u,v)} \in \{ c_{v'}, c_u \}^{\complement}$ for $(u, v) = e_{i+1}$ and $e_i = (u', v')$. We let $\mathtt{p}_{e_i}$ add p_i. We let $\mathtt{p}_{(u,v)}$ consume b_{-1} for $(u, v) = e_0$ and let $\mathtt{p}_{(u,v)}$ consume $c_{v'}$ for $(u, v) = e_{i+1}$ and $e_i = (u', v')$. We let $\mathtt{q}_{(u,v)}$ add c_u and consume $p_{(u,v)}$. We let $\mathtt{r}_{(u,v)}$ add c_u and consume c_v — note that this is well-defined only because $\mathfrak{c}$ is a valid coloring such that $c_u \neq c_v$. □

The bound $k = 3$ as used in above proof is the smallest bound for which the problem is NP-hard. For $k = 1$ any plan π with $|\pi| > 2$ is necessarily redundant because π_0 and π_1 will be unable to add distinct variables. For $k = 2$ the actions (except for π_{-1}) must alternate between adding x and adding y for distinct variables x and y. Consequently,



(a) Subsequence corresponding to nodes u and v.



(b) Subsequence corresponding to edges (u, w) and (v, w).

Figure 7: Example trace of π as in the proof of Theorem 3 for a 3-clique $\{ u, v, w \}$ with coloring $\mathfrak{c} = \{ (u, 0), (v, 1), (w, 2) \}$.

π is well-justified in some domain δ with $\mathcal{V}(\delta) = \{ x, y \}$ if and only if any action occurs at either only even or only odd positions in the plan, which can be checked in polynomial time.

Our proof can be extended to work for any constant $k \geq 3$ and from Theorem 1 and Corollary 1 we know that the problem is in P for bounds $k \geq |\pi| - 1$. The complexity for bounds such as $\log |\pi|$ remains unknown.

Perfectly Justified Plans

For the case of perfectly justified plans, we obtain the following membership result.

Theorem 4. *Given a plan π and a number k that is polynomial in the size of the input, deciding whether there exists a domain $\delta \in \Delta^k(\pi)$ in which π is perfectly justified is in Σ_2^{P}.*

Proof. Guess a domain with k variables and check whether π is perfectly justified (Lemma 2). □

To show hardness, however, we will have to come up with a construction that is substantially more involved than the one we used in the proof of Theorem 3, as the following result indicates.

Lemma 7. *Unless* P = NP*, for any polynomial reduction from an* NP*-hard problem to deciding whether there exists a domain $\delta \in \Delta^k(\pi)$ in which a given plan π is perfectly justified it holds $k \notin \mathcal{O}(1)$.*

Proof. Contraposition: Let f be such a reduction with k bounded by some constant $c \in \mathbb{N}$. In a domain with at most c variables, there are at most 2^c distinct states and therefore any plan π with $|\pi| > 2^c$ is not perfectly justified. Furthermore, there exist finitely many plans of length at most 2^c up to isomorphism (equivalence up to renaming actions).

This means that solving this domain learning problem can be done in constant time (trivial for $|\pi| > 2^c$, finitely many distinct runtimes for $|\pi| \leq 2^c$). It follows that the problem that f reduces is solvable in polynomial time. □

This means that for any polynomial reduction showing NP-hardness (and in particular Σ_2^{P}-hardness) of the problem k must grow with the size of the input.

Conclusion

Domain learning is a central task in the field of AI planning that is especially relevant for planning to be used by non-experts. While several domain learning methods assume full observability of the state, in practice the set of relevant state variables may be unknown and has to be inferred. In this paper, we have started a theoretical investigation into domain learning in the fully unobserved setting. We formalized the respective decision problem and investigated its computational complexity. As sensible assumptions on the given action sequences, we put forward well-justification and perfect justification.

We showed that for well-justified plans domain learning is solvable in polynomial time. If we limit the number of state variables of the learned domain, the problem becomes NP-complete for bounds of at least three. Regarding perfect justification, we showed that in the unbounded case domain learning is in coNP, while it is in Σ_2^{P} for the bounded case. Tight lower bounds still need to be established.

We believe that this work lays a strong foundation for analyzing domain learning problems also within planning frameworks that are more expressive than STRIPS. Future research might transfer our findings to the widely used PDDL formalism, where our hardness results are immediately applicable as a special case. For membership, we can likely use logical formulae to constrain the learned model (see Bachor and Behnke (2024)), ensuring it corresponds to a lifted PDDL model.

Ultimately, our proofs and formalizations reveal properties of the planning problem, such as circumstances under which plans are potentially or necessarily redundant, that can be insightful beyond the specific context of domain learning.

References

Aineto, D.; Celorrio, S. J.; and Onaindia, E. 2019. Learning action models with minimal observability. *Artificial Intelligence*, 275: 104–137.

Aineto, D.; Jiménez, S.; and Onaindia, E. 2018. Learning STRIPS Action Models with Classical Planning. In *28th International Conference on Automated Planning and Scheduling (ICAPS 2018)*, 299–407.

Aineto, D.; and Scala, E. 2024. Action Model Learning with Guarantees. In *21st International Conference on Principles of Knowledge Representation and Reasoning (KR 2024)*, 801–811.

Arora, A.; Fiorino, H.; Pellier, D.; Métivier, M.; and Pesty, S. 2018. A Review of Learning Planning Action Models. *The Knowledge Engineering Review*, 33: e20.

Bachor, P.; and Behnke, G. 2024. Learning Planning Domains from Non-Redundant Fully-Observed Traces: Theoretical Foundations and Complexity Analysis. In *38th AAAI Conference on Artificial Intelligence (AAAI 2024)*, 20028–20035.

Benson, S. 1996. *Learning Action Models for Reactive Autonomous Agents*. Ph.D. thesis, Stanford University.

Bolander, T.; Gierasimczuk, N.; and Liberman, A. O. 2021. Learning to Act and Observe in Partially Observable Domains. *CoRR*, abs/2109.06076.

Bonet, B.; and Geffner, H. 2020. Learning First-Order Symbolic Representations for Planning from the Structure of the State Space. In *24th European Conference on AI (ECAI 2020)*, 2322–2329.

Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. *Artificial Intelligence*, 69(1-2): 165–204.

Chrpa, L.; McCluskey, T.; and Osborne, H. 2012. Optimizing Plans through Analysis of Action Dependencies and Independencies. In *22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 338–342.

Cresswell, S.; and Gregory, P. 2011. Generalised Domain Model Acquisition from Action Traces. In *21st International Conference on Automated Planning and Scheduling (ICAPS 2011)*, 42–49.

Cresswell, S.; McCluskey, T.; and West, M. 2009. Acquisition of Object-Centred Domain Models from Planning Examples. In *19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 338–341.

Cresswell, S. N.; McCluskey, T. L.; and West, M. M. 2013. Acquiring planning domain models using LOCM. *The Knowledge Engineering Review*, 28(2): 195–213.

Fikes, R.; and Nilsson, N. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3–4): 189–208.

Fink, E. 1992. *Justified Plans and Ordered Hierarchies*. Master's thesis, University of Waterloo.

Fink, E.; and Yang, Q. 1992. Formalizing Plan Justifications. In *9th Canadian Conference on Artificial Intelligence*, 9–14.

Fink, E.; and Yang, Q. 1993. A Spectrum of Plan Justifications. In *AAAI 1993 Spring Symposium*, 23–33.

Gregory, P.; and Lindsay, A. 2016. Domain Model Acquisition in Domains with Action Costs. In *26th International Conference on Automated Planning and Scheduling (ICAPS 2016)*, 149–157.

Gösgens, J.; Jansen, N.; and Geffner, H. 2025. Learning Lifted STRIPS Models from Action Traces Alone: A Simple, General, and Scalable Solution. In *35th International Conference on Automated Planning and Scheduling (ICAPS 2025)*.

Haslum, P. 2006. *Admissible Heuristics for Automated Planning*. Phd dissertation, Linköpings Universitet.

Jiménez, S.; De La Rosa, T.; Fernández, S.; Fernández, F.; and Borrajo, D. 2012. A Review of Machine Learning for Automated Planning. *The Knowledge Engineering Review*, 27(4): 433–467.

Juba, B.; Le, H. S.; and Stern, R. 2021. Safe Learning of Lifted Action Models. In *18th International Conference on Principles of Knowledge Representation and Reasoning (KR 2021)*, 379–389.

Kambhampati, S. 2007. Model-lite Planning for the Web Age Masses: The Challenges of Planning with Incomplete and Evolving Domain Models. In *22nd AAAI Conference on Artificial Intelligence (AAAI 2007)*, 1601–1604.

Kučera, J.; and Barták, R. 2018. LOUGA: learning planning operators using genetic algorithms. In *15th Pacific Rim Knowledge Acquisition Workshop (PKAW 2018)*, 124–138.

Lamanna, L.; Saetti, A.; Serafini, L.; Gerevini, A.; and Traverso, P. 2021. Online Learning of Action Models for PDDL Planning. In *30th International Joint Conference on Artificial Intelligence (IJCAI 2021)*, 4112–4118.

Lamanna, L.; Serafini, L.; Saetti, A.; Gerevini, A. E.; and Traverso, P. 2025. Lifted action models learning from partial traces. *Artificial Intelligence*, 339: 104256.

Lin, S.; and Bercher, P. 2021. Change the World - How Hard Can that Be? On the Computational Complexity of Fixing Planning Models. In *30th International Joint Conference on Artificial Intelligence (IJCAI 2021)*, 4152–4159.

Lin, S.; and Bercher, P. 2023. Was Fixing this Really That Hard? On the Complexity of Correcting HTN Domains. In *37th AAAI Conference on Artificial Intelligence (AAAI 2023)*, 12032–12040.

Lin, S.; Grastien, A.; and Bercher, P. 2023. Towards Automated Modeling Assistance: An Efficient Approach for Repairing Flawed Planning Domains. In *37th AAAI Conference on Artificial Intelligence (AAAI 2023)*, 12022–12031.

Lin, S.; Grastien, A.; Shome, R.; and Bercher, P. 2025. Told You That Will Not Work: Optimal Corrections to Planning Domains Using Counter-Example Plans. In *39th AAAI Conference on Artificial Intelligence (AAAI 2025)*, 26596–26604.

Lin, S.; Höller, D.; and Bercher, P. 2024. Modeling Assistance for Hierarchical Planning: An Approach for Correcting Hierarchical Domains with Missing Actions. In *17th International Symposium on Combinatorial Search (SoCS 2024)*.

Med, J.; and Chrpa, L. 2022. On Speeding Up Methods for Identifying Redundant Actions in Plans. In *32nd International Conference on Automated Planning and Scheduling (ICAPS 2022)*, 252–260.

Nakhost, H.; and Müller, M. 2010. Action elimination and plan neighborhood graph search: Two algorithms for plan improvement. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 121–128.

Salerno, M.; Fuentetaja, R.; and Seipp, J. 2023. Eliminating Redundant Actions from Plans using Classical Planning. In *20th International Conference on Principles of Knowledge Representation and Reasoning (KR 2023)*, 774–778.

Shaik, I.; and van de Pol, J. 2022. Classical planning as QBF without grounding. In *32nd International Conference on Automated Planning and Scheduling (ICAPS 2022)*, 329–337.

Sreedharan, S.; Bercher, P.; and Kambhampati, S. 2022. On the Computational Complexity of Model Reconciliations. In *31st International Joint Conference on Artificial Intelligence (IJCAI 2022)*, 4657–4664.

Walsh, T. J.; and Littman, M. L. 2008. Efficient learning of action schemas and web-service descriptions. In *23rd AAAI Conference on Artificial Intelligence (AAAI 2008)*, 714–719.

Wang, X. 1995. Learning by Observation and Practice: An Incremental Approach for Planning Operator Acquisition. In *12th International Conference on Machine Learning (ICML 1995)*, 549–557.

Wang, X. 1996. *Learning planning operators by observation and practice*. Ph.D. thesis, Carnegie Mellon University.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence*, 171(2-3): 107–143.

Zhuo, H. H.; and Kambhampati, S. 2013. Action-model acquisition from noisy plan traces. In *23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 2444–2450.

Zhuo, H. H.; Yang, Q.; Hu, D. H.; and Li, L. 2010. Learning complex action models with quantifiers and logical implications. *Artificial Intelligence*, 174(18): 1540–1569.