Predicate Invention for Bilevel Planning

Tom Silver[*1], **Rohan Chitnis**[*2], **Nishanth Kumar**[1], **Willie McClinton**[1], **Tomás Lozano-Pérez**[1], **Leslie Kaelbling**[1], **Joshua Tenenbaum**[1]

[1]MIT Computer Science and Artificial Intelligence Laboratory
[2]Meta AI
{tslvr, njk, wbm3, tlp, lpk, jbt}@mit.edu, ronuchit@meta.com

Abstract

Efficient planning in continuous state and action spaces is fundamentally hard, even when the transition model is deterministic and known. One way to alleviate this challenge is to perform bilevel planning with abstractions, where a high-level search for abstract plans is used to guide planning in the original transition space. Previous work has shown that when state abstractions in the form of symbolic predicates are hand-designed, operators and samplers for bilevel planning can be learned from demonstrations. In this work, we propose an algorithm for learning predicates from demonstrations, eliminating the need for manually specified state abstractions. Our key idea is to learn predicates by optimizing a surrogate objective that is tractable but faithful to our real efficient-planning objective. We use this surrogate objective in a hill-climbing search over predicate sets drawn from a grammar. Experimentally, we show across four robotic planning environments that our learned abstractions are able to quickly solve held-out tasks, outperforming six baselines.

1 Introduction

Hierarchical planning is a powerful approach for decision-making in environments with continuous states, continuous actions, and long horizons. A crucial bottleneck in scaling hierarchical planning is the reliance on human engineers to manually program domain-specific abstractions. For example, in bilevel sample-based task and motion planning (Srivastava et al. 2014; Garrett et al. 2021), an engineer must design (1) symbolic predicates; (2) symbolic operators; and (3) samplers that propose different refinements of the symbolic operators into continuous actions. However, recent work has shown that when predicates are *given*, operators and samplers can be learned from a modest number (50–200) of demonstrations (Silver et al. 2021; Chitnis et al. 2022). Our objective in this work is to *learn* predicates that can then be used to learn operators and samplers.

Predicates in bilevel planning represent a discrete state abstraction of the underlying continuous state space (Li, Walsh, and Littman 2006; Abel, Hershkowitz, and Littman 2017). For example, `On(block1, block2)` is an abstraction that discards the exact continuous poses of `block1` and `block2`. State abstraction alone is useful for decision-making, but predicates go further: together with operators, predicates enable the use of highly-optimized domain-independent AI planners (Helmert 2006).

We consider a problem setting where a small set of *goal predicates* are available and sufficient for describing task goals, but practically insufficient for bilevel planning. For example, in a block stacking domain (Figure 1), we start with `On` and `OnTable`, but have no predicates for describing whether a block is currently held or graspable. Our aim is to invent new predicates to enrich the state abstraction beyond what can be expressed with the goal predicates alone, leading to stronger reasoning at the abstract level.

What objective should we optimize to learn predicates for bilevel planning? First, consider our real objective: we want a predicate set such that bilevel planning is fast and successful, in expectation over a task distribution, when we use those predicates to learn operators and samplers for planning. Unfortunately, this real objective is far too expensive to use directly, since even a single evaluation requires neural network sampler training and bilevel planning.

In this work, we propose a novel surrogate objective that is deeply connected to our real bilevel-planning objective, but is tractable for predicate learning. Our main insight is that demonstrations can be used to analytically approximate bilevel planning time. To leverage this objective for predicate learning, we take inspiration from the program synthesis literature (Menon et al. 2013; Ellis et al. 2020), and learn predicates via a hill-climbing search through a grammar, with the search guided by the objective. After predicate learning, we use the predicates to learn operators and samplers. All three components can then be used for efficient bilevel planning on new tasks.

In experiments across four robotic planning environments, we find predicates, operators, and samplers learned from 50–200 demonstrations enable efficient bilevel planning on held-out tasks that involve different numbers of objects, longer horizons, and larger goal expressions than seen in the demonstrations. Furthermore, predicates learned with our proposed surrogate objective substantially outperform those learned with objectives inspired by previous work, which are based on prediction error (Pasula, Zettlemoyer, and Kaelbling 2007; Jetchev, Lang, and Toussaint 2013), bisimulation (Konidaris, Kaelbling, and Lozano-Perez 2018;

[*]These authors contributed equally.

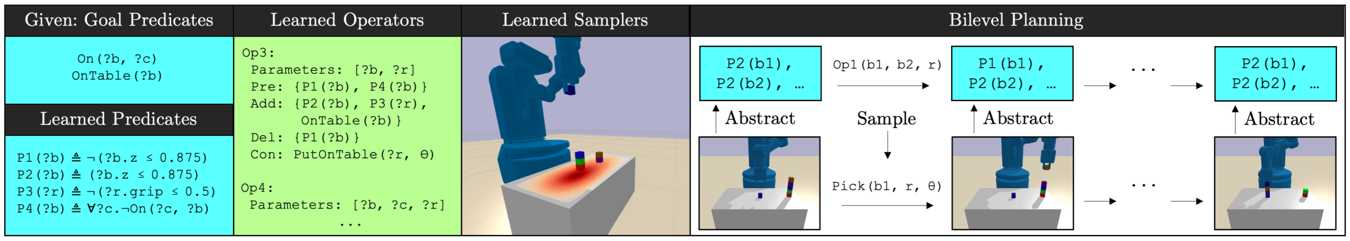


Figure 1: Overview of our framework. Given a small set of goal predicates (first panel, top), we use demonstration data to learn new predicates (first panel, bottom). In this Blocks example, the learned predicates `P1 – P4` intuitively represent `Holding`, `NotHolding`, `HandEmpty`, and `NothingAbove` respectively. Collectively, the predicates define a state abstraction that maps continuous states x in the environment to abstract states s. Object types are omitted for clarity. After predicate invention, we learn abstractions of the continuous action space and transition model via planning operators (second panel). For each operator, we learn a sampler (third panel), a neural network that maps continuous object features in a given state to continuous action parameters for controllers which can be executed in the environment. In this example, the sampler proposes different placements on the table for the held block. With these learned representations, we perform bilevel planning (fourth panel), with search in the abstract spaces guiding planning in the continuous spaces.

Curtis et al. 2021), and inverse planning (Baker, Saxe, and Tenenbaum 2009; Ramírez and Geffner 2010; Zhi-Xuan et al. 2020). We compare against several other baselines and ablations of our system to further validate our results.

2 Problem Setting

We consider learning from demonstrations in deterministic planning problems. These problems are goal-based and object-centric, with continuous states and hybrid discrete-continuous actions. Formally, an *environment* is a tuple $\langle \Lambda, d, \mathcal{C}, f, \Psi_G \rangle$, and is associated with a distribution $\mathcal{T}$ over *tasks*, where each task $T \in \mathcal{T}$ is a tuple $\langle \mathcal{O}, x_0, g \rangle$.

Λ is a finite set of object *types*, and the map $d : \Lambda \to \mathbb{N}$ defines the dimensionality of the real-valued feature vector for each type. Within a task, $\mathcal{O}$ is an *object set*, where each object has a type drawn from Λ; this $\mathcal{O}$ can (and typically will) vary between tasks. $\mathcal{O}$ induces a state space $\mathcal{X}_\mathcal{O}$ (going forward, we simply write $\mathcal{X}$ when clear from context). A *state* $x \in \mathcal{X}$ in a task is a mapping from each $o \in \mathcal{O}$ to a feature vector in $\mathbb{R}^{d(\text{type}(o))}$; x_0 is the initial state of the task.

$\mathcal{C}$ is a finite set of *controllers*. A controller $C((\lambda_1, \ldots, \lambda_v), \Theta) \in \mathcal{C}$ can have both discrete typed parameters $(\lambda_1, \ldots, \lambda_v)$ and a continuous real-valued vector of parameters Θ. For instance, a controller `Pick` for picking up a block might have one discrete parameter of type `block` and a Θ that is a placeholder for a specific grasp pose. The controller set $\mathcal{C}$ and object set $\mathcal{O}$ induce an action space $\mathcal{A}_\mathcal{O}$ (going forward, we write $\mathcal{A}$ when clear). An *action* $a \in \mathcal{A}$ in a task is a controller $C \in \mathcal{C}$ with both discrete and continuous arguments: $a = C((o_1, \ldots o_v), \theta)$, where the objects $(o_1, \ldots o_v)$ are drawn from the object set $\mathcal{O}$ and must have types matching the controller's discrete parameters $(\lambda_1, \ldots, \lambda_v)$. Transitions through states and actions are governed by $f : \mathcal{X} \times \mathcal{A} \to \mathcal{X}$, a known, deterministic transition model that is shared across tasks.

A *predicate* ψ is characterized by an ordered list of types $(\lambda_1, \ldots, \lambda_m)$ and a lifted binary state classifier $c_\psi : \mathcal{X} \times \mathcal{O}^m \to \{\text{true}, \text{false}\}$, where $c_\psi(x, (o_1, \ldots, o_m))$ is defined only when each object o_i has type λ_i. For instance, the predicate `Holding` may, given a state and two objects, robot and block, describe whether the block is held by the robot in this state. A *lifted atom* is a predicate with typed variables (e.g., `Holding(?robot, ?block)`). A *ground atom* $\underline{\psi}$ consists of a predicate ψ and objects $(o_1, \ldots, o_m)$, again with all type$(o_i) = \lambda_i$ (e.g., `Holding(robby, block7)`). Note that a ground atom induces a binary state classifier $c_{\underline{\psi}} : \mathcal{X} \to \{\text{true}, \text{false}\}$, where $c_{\underline{\psi}}(x) \triangleq c_\psi(x, (o_1, \ldots, o_m))$.

Ψ_G is a small set of *goal predicates* that we assume are given and sufficient for representing task goals, but insufficient practically as standalone state abstractions. Specifically, the goal g of a task is a set of ground atoms over predicates in Ψ_G and objects in $\mathcal{O}$. A goal g is said to *hold* in a state x if for all ground atoms $\underline{\psi} \in g$, the classifier $c_{\underline{\psi}}(x)$ returns true. A solution to a task is a *plan* $\pi = (a_1, \ldots, a_n)$, a sequence of actions $a \in \mathcal{A}$ such that successive application of the transition model $x_i = f(x_{i-1}, a_i)$ on each $a_i \in \pi$, starting from x_0, results in a final state x_n where g holds.

The agent is provided with a set of *training tasks* from $\mathcal{T}$ and a set of demonstrations $\mathcal{D}$, with *one demonstration per task*. We assume action costs are unitary and demonstrations are near-optimal. Each demonstration consists of a training task $\langle \mathcal{O}, x_0, g \rangle$ and a plan π^* that solves the task. Note that for each π^*, we can recover the associated state sequence starting at x_0, since f is known and deterministic. The agent's objective is to *efficiently* solve held-out tasks from $\mathcal{T}$ using anything it chooses to learn from $\mathcal{D}$.

3 Predicates, Operators, and Samplers

Since the agent has access to the transition model f, one approach for optimizing the objective described in Section 2 is to forgo learning entirely, and solve any held-out task by running a planner over the state state $\mathcal{X}$ and action space $\mathcal{A}$. However, searching for a solution directly in these large spaces is highly infeasible. Instead, we propose to *learn abstractions* using the provided demonstrations. In this section, we will describe representations that allow for fast bilevel planning with abstractions (Section 4). In Section 5, we then describe how to learn these abstractions.

We adopt a very general definition of an abstraction (Konidaris and Barto 2009): mappings from $\mathcal{X}$ and $\mathcal{A}$

$$\begin{tikzcd}
 & \text{Abstraction} & \\
 & F & \\
\mathcal{S}_\Psi \arrow[rr] & \underline{\Omega} \arrow[d, "\Sigma"] & \mathcal{S}_\Psi \\
\mathcal{X} \arrow[u, "\Psi"] \arrow[rr] & \mathcal{A} & \mathcal{X} \arrow[u, "\Psi"] \\
 & f & \\
 & \text{Task} &
\end{tikzcd}$$

Figure 2: Summary of abstractions.

to alternative state and action spaces. We first characterize an abstract state space $\mathcal{S}_\Psi$ and a transformation from states in $\mathcal{X}$ to abstract states. Next, we describe an abstract action space $\underline{\Omega}$ and an abstract transition model $F : \mathcal{S}_\Psi \times \underline{\Omega} \to \mathcal{S}_\Psi$ that can be used to plan in the abstract space. Finally, we define samplers Σ for refining abstract actions back into $\mathcal{A}$, i.e., actions that can be executed. See Figure 2.

(1) An abstract state space. We use a set of predicates Ψ (as defined in Section 2) to induce an abstract state space $\mathcal{S}_\Psi$. Recalling that a ground atom $\underline{\psi}$ induces a classifier $c_{\underline{\psi}}$ over states $x \in \mathcal{X}$, we have:

Definition 1 (Abstract state)**.** *An* abstract state *s is the set of ground atoms under Ψ that hold true in x:*

$$s = \textsc{Abstract}(x, \Psi) \triangleq \{\underline{\psi} : c_{\underline{\psi}}(x) = \text{true}, \forall \psi \in \Psi\}.$$

The (discrete) abstract state space induced by Ψ is denoted $\mathcal{S}_\Psi$. Throughout this work, we use predicate sets Ψ that are supersets of the given goal predicates Ψ_G. However, only the goal predicates are given, and they alone are typically very limited; in Section 5, we will discuss how the agent can use data to *invent predicates* that will make up the rest of Ψ. See Figure 1 (first panel) for an example.

(2) An abstract action space and abstract transition model. We address both by having the agent learn *operators*:

Definition 2 (Operator)**.** *An* operator *is a tuple $\omega = \langle \textsc{Par}, \textsc{Pre}, \textsc{Eff}^+, \textsc{Eff}^-, \textsc{Con} \rangle$ where:*

- $\textsc{Par}$ *is an ordered list of* parameters*: variables with types drawn from the type set Λ.*
- $\textsc{Pre}, \textsc{Eff}^+, \textsc{Eff}^-$ *are* preconditions*,* add effects*, and* delete effects*, each a set of lifted atoms over Ψ and* $\textsc{Par}$*.*
- $\textsc{Con}$ *is a tuple $\langle C, \textsc{Par}_{\textsc{Con}} \rangle$ where $C((\lambda_1, \dots, \lambda_v), \Theta)$ is a controller and $\textsc{Par}_{\textsc{Con}}$ is an ordered list of* controller arguments*, each a variable from* $\textsc{Par}$*. Furthermore, $|\textsc{Par}_{\textsc{Con}}| = v$, and each argument i must be of the respective type λ_i.*

We denote the set of operators as Ω. See Figure 1 (second panel) for an example. Unlike in STRIPS (Fikes and Nilsson 1971), our operators are augmented with controllers and controller arguments, which will allow us to connect to the task actions in **(3)** below. Now, given a task with object set $\mathcal{O}$, the set of all *ground operators* defines our (discrete) abstract action space for a task:

Definition 3 (Ground operator / abstract action)**.** *A* ground operator *$\underline{\omega} = \langle \omega, \delta \rangle$ is an operator ω and a substitution $\delta : \textsc{Par} \to \mathcal{O}$ mapping parameters to objects. We use $\underline{\textsc{Pre}}, \underline{\textsc{Eff}^+}, \underline{\textsc{Eff}^-}$, and $\underline{\textsc{Par}_{\textsc{Con}}}$ to denote the ground preconditions, ground add effects, ground delete effects, and ground controller arguments of $\underline{\omega}$, where variables in* $\textsc{Par}$ *are substituted with objects under δ.*

We denote the set of ground operators (the abstract action space) as $\underline{\Omega}$. Together with the abstract state space $\mathcal{S}_\Psi$, the preconditions and effects of the operators induce an abstract transition model for a task:

Definition 4 (Abstract transition model)**.** *The* abstract transition model *induced by predicates Ψ and operators Ω is a partial function $F : \mathcal{S}_\Psi \times \underline{\Omega} \to \mathcal{S}_\Psi$. $F(s, \underline{\omega})$ is only defined if $\underline{\omega}$ is* applicable *in s: $\underline{\textsc{Pre}} \subseteq s$. If defined, $F(s, \underline{\omega}) \triangleq (s - \underline{\textsc{Eff}^-}) \cup \underline{\textsc{Eff}^+}$.*

(3) A mechanism for refining abstract actions into task actions. A ground operator $\underline{\omega}$ induces a partially specified controller, $C((o_1, \dots o_v), \Theta)$ with $(o_1, \dots o_v) = \underline{\textsc{Par}_{\textsc{Con}}}$, where object arguments have been selected but continuous parameters Θ have not. To *refine* this abstract action $\underline{\omega}$ into a task-level action $a = C((o_1, \dots o_v), \theta)$, we use *samplers*:

Definition 5 (Sampler)**.** *Each operator $\omega \in \Omega$ is associated with a* sampler *$\sigma : \mathcal{X} \times \mathcal{O}^{|\textsc{Par}|} \to \Delta(\Theta)$, where $\Delta(\Theta)$ is the space of distributions over Θ, the continuous parameters of the operator's controller.*

Definition 6 (Ground sampler)**.** *For each ground operator $\underline{\omega} \in \underline{\Omega}$, if $\underline{\omega} = \langle \omega, \delta \rangle$ and σ is the sampler associated with ω, then the* ground sampler *associated with $\underline{\omega}$ is a state-conditioned distribution $\underline{\sigma} : \mathcal{X} \to \Delta(\Theta)$, where $\underline{\sigma}(x) \triangleq \sigma(x, \delta(\textsc{Par}))$.*

We denote the set of samplers as Σ. See Figure 1 (third panel) for an example.

What connects the transition model f, abstract transition model F, and samplers Σ? While previous works enforce the downward refinability property (Marthi, Russell, and Wolfe 2007; Pasula, Zettlemoyer, and Kaelbling 2007; Jetchev, Lang, and Toussaint 2013; Konidaris, Kaelbling, and Lozano-Perez 2018), it is important in robotics to be robust to violations of this property, since learned abstractions will typically lose critical geometric information. Therefore, we only require our learned abstractions to satisfy the following *weak semantics*: for every ground operator $\underline{\omega}$ with partially specified controller $C((o_1, \dots, o_v), \Theta)$ and associated ground sampler $\underline{\sigma}$, there exists some $x \in \mathcal{X}$ and some θ in the support of $\underline{\sigma}(x)$ such that $F(s, \underline{\omega})$ is defined and equals s', where $s = \textsc{Abstract}(x, \Psi)$, $a = C((o_1, \dots, o_v), \theta)$, and $s' = \textsc{Abstract}(f(x, a), \Psi)$. Note that downward refinability (Marthi, Russell, and Wolfe 2007) makes a much stronger assumption: that this statement holds for *every* $x \in \mathcal{X}$ where $F(s, \underline{\omega})$ is defined.

4 Bilevel Planning

To use the components of an abstraction — predicates Ψ, operators Ω, and samplers Σ — for efficient planning, we build on *bilevel* planning techniques (Srivastava et al. 2014; Garrett et al. 2021). We conduct an outer search over *abstract plans* using the predicates and operators, and an inner search over refinements of an abstract plan into a task solution π using the predicates and samplers.

Algorithm 1 Bilevel Planning

```
Plan($x_0$, $g$, $\Psi$, $\Omega$, $\Sigma$ ):
    $s_0 \leftarrow$ ABSTRACT($x_0$, $\Psi$)
    for $\hat{\pi}$ in GENABSTRACTPLAN($s_0$, $g$, $\Omega$, $n_{\text{abstract}}$) do
        if $\pi \sim$ REFINE($\hat{\pi}$, $x_0$, $\Psi$, $\Sigma$, $n_{\text{samples}}$) then
            return $\pi$
        end if
    end for
```

Definition 7 (Abstract plan)**.** *An* abstract plan $\hat{\pi}$ *for a task* $\langle \mathcal{O}, x_0, g\rangle$ *is a sequence of ground operators* $(\underline{\omega}_1, \ldots, \underline{\omega}_n)$ *such that applying the abstract transition model* $s_i = F(s_{i-1}, \underline{\omega}_i)$ *successively starting from* $s_0 =$ ABSTRACT(x_0, Ψ) *results in a sequence of abstract states* $(s_0, \ldots, s_n)$ *that achieves the goal, i.e.,* $g \subseteq s_n$*. This* $(s_0, \ldots, s_n)$ *is called the* expected abstract state sequence.

Because downward refinability does not hold in our setting, an abstract plan $\hat{\pi}$ is *not* guaranteed to be refinable into a solution π for the task, which necessitates bilevel planning. We now describe the planning algorithm in detail.

The overall structure of the planner is outlined in Algorithm 1. For the outer search that finds abstract plans $\hat{\pi}$, denoted GENABSTRACTPLAN (Alg. 1, Line 2), we leverage the STRIPS-style operators and predicates (Fikes and Nilsson 1971) to automatically derive a domain-independent heuristic popularized by the AI planning community, such as LMCut (Helmert and Domshlak 2009). We use this heuristic to run an A* search over the abstract state space $\mathcal{S}_\Psi$ and abstract action space $\underline{\Omega}$. This A* search is used as a generator (hence the name GENABSTRACTPLAN) of abstract plans $\hat{\pi}$, outputting one at a time[1]. Parameter n_{abstract} governs the maximum number of abstract plans that can be generated before the planner terminates with failure.

For each abstract plan $\hat{\pi}$, we conduct an inner search that attempts to REFINE (Alg. 1, Line 3) it into a solution π (a plan that achieves the goal under the transition model f). While various implementations of REFINE are possible (Chitnis et al. 2016), we follow Srivastava et al. (2014) and perform a backtracking search over the abstract actions $\underline{\omega}_i \in \hat{\pi}$. Recall that each $\underline{\omega}_i$ induces a partially specified controller $C_i((o_1, \ldots, o_v)_i, \Theta_i)$ and has an associated ground sampler $\underline{\sigma}_i$. To begin the search, we initialize an indexing variable i to 1. On each step of search, we sample continuous parameters $\theta_i \sim \underline{\sigma}_i(x_{i-1})$, which fully specify an action $a_i = C_i((o_1, \ldots, o_v)_i, \theta_i)$. We then check whether $x_i = f(x_{i-1}, a_i)$ obeys the expected abstract state sequence, i.e., whether $s_i =$ ABSTRACT(x_i, Ψ). If so, we continue on to $i \leftarrow i+1$. Otherwise, we repeat this step, sampling a new $\theta_i \sim \underline{\sigma}_i(x_{i-1})$. Parameter n_{samples} governs the maximum number of times we invoke the sampler for a single value of i before backtracking to $i \leftarrow i-1$. REFINE succeeds if the goal g holds when $i = |\hat{\pi}|$, and fails when i backtracks to 0.

[1]This usage of A* search as a generator is related to top-k planning (Katz et al. 2018; Ren, Chalvatzaki, and Peters 2021). We experimented with off-the-shelf top-k planners, but chose A* because it was faster in our domains. Note that the abstract plan generator is used heavily in learning (Section 5).

Algorithm 2 Estimate Total Planning Time

```
ETPT( $x_0$, $g$, $\Psi$, $\Omega$, $\pi^*$):
    $s_0 \leftarrow$ ABSTRACT($x_0$, $\Psi$)
    $p_{\text{terminate}} \leftarrow 0.0$
    $t_{\text{expected}} \leftarrow 0.0$
    for $\hat{\pi}$ in GENABSTRACTPLAN($s_0$, $g$, $\Omega$, $n_{\text{abstract}}$) do
        $p_{\text{refined}} \leftarrow$ ESTIMATEREFINEPROB($\hat{\pi}$, $\pi^*$)
        $p_{\text{terminate}} \leftarrow (1 - p_{\text{terminate}}) \cdot p_{\text{refined}}$
        $t_{\text{iter}} \leftarrow$ ESTIMATETIME($\hat{\pi}$, $x_0$, $\Psi$, $\Omega$)
        $t_{\text{expected}} \leftarrow t_{\text{expected}} + p_{\text{terminate}} \cdot t_{\text{iter}}$
        $t_{\text{expected}} \leftarrow t_{\text{expected}} + (1 - p_{\text{terminate}}) \cdot t_{\text{upper}}$
    end for
    return $t_{\text{expected}}$
```

If REFINE succeeds given a candidate $\hat{\pi}$, the planner terminates with success (Alg. 1, Line 4) and returns the plan $\pi = (a_1, \ldots, a_{|\hat{\pi}|})$. Crucially, if REFINE fails, we continue with GENABSTRACTPLAN to generate the next candidate $\hat{\pi}$. In the taxonomy of task and motion planners (TAMP), this approach is in the "search-then-sample" category (Srivastava et al. 2014; Dantam et al. 2016; Garrett et al. 2021). As we have described it, this planner is *not* probabilistically complete, because abstract plans are not revisited. Extensions to ensure completeness are straightforward (Chitnis et al. 2016), but are not our focus in this work.

5 Learning from Demonstrations

To use bilevel planning at evaluation time, we must learn predicates, operators, and samplers at training time. We use the methods of Chitnis et al. (2022) for operator learning and sampler learning; see Section A.1 and Section A.3 for descriptions. For what follows, it is important to understand that operator learning is fast ($O(|\mathcal{D}|)$), but sampler learning is slow, and both require a given set of predicates. Our main contribution is a method for predicate invention that precedes operator and sampler learning in the training pipeline.

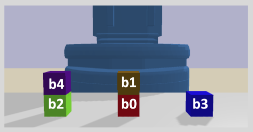
Inspired by prior work (Bonet and Geffner 2019; Loula et al. 2019; Curtis et al. 2021), we approach the predicate invention problem from a program synthesis perspective (Stahl 1993; Lavrac and Dzeroski 1994; Cropper and Muggleton 2016; Ellis et al. 2020). First, we define a compact representation of an infinite space of predicates in the form of a *grammar*. We then enumerate a large pool of *candidate predicates* from this grammar, with simpler candidates enumerated first. Next, we perform a *local search* over subsets of candidates, with the aim of identifying a good final subset to use as Ψ. The crucial question in this step is: what *objective function* should we use to guide the search over candidate predicate sets?

5.1 Scoring a Candidate Predicate Set

Ultimately, we want to find a set of predicates Ψ that will lead to efficient planning, after we use the predicates to learn operators Ω and samplers Σ. I.e., our real objective is:

$$J_{\text{real}}(\Psi) \triangleq \mathbb{E}_{(\mathcal{O}, x_0, g)\sim\mathcal{T}}[\text{TIME}(\text{PLAN}(x_0, g, \Psi, \Omega, \Sigma))],$$

Initial state:



Goal:

```
OnTable(b3)
On(b2, b3)
On(b1, b2)
```

Search Iteration	Predicate Set	J_{surr} (lower is better)	Success Rate on Evaluation Tasks
0	`On(?b, ?c)` `OnTable(?b)`	$1.3 \cdot 10^7$	0%
1	`On(?b, ?c)` `OnTable(?b)` `¬(?b.z ≤ 0.875)`	$1.0 \cdot 10^7$	12%
2	`On(?b, ?c)` `OnTable(?b)` `¬(?b.z ≤ 0.875)` `∀?c.¬On(?c, ?b)`	$2.0 \cdot 10^6$	14%
3	`On(?b, ?c)` `OnTable(?b)` `¬(?b.z ≤ 0.875)` `∀?c.¬On(?c, ?b)` `¬(?r.grip ≤ 0.5)`	9351	100%

Abstract plans:	Refinable?
`[Stack b2 on b3, Stack b1 on b2]`	✗
`...`	
`[Pick b2, Stack b2 on b3, Pick b1, Stack b1 on b2]`	✗
`...`	
`[Pick b4, Pick b2, Stack b2 on b3, Pick b1, Stack b1 on b2]`	✗
`...`	
`[Pick b4, PutOnTable b4, Pick b2, Stack b2 on b3, Pick b1, Stack b1 on b2]`	✓

Figure 3: Predicate invention via hill climbing. (Left) An example task in Blocks. (Middle) Hill climbing over predicate sets, starting with the goal predicates Ψ_G. On each iteration, the single predicate that improves J_{surr} the most is added to the set. The rightmost table column shows success rates on held-out evaluation tasks. Each iteration of hill climbing adds a predicate that causes all abstract plans above the dotted line to be pruned from consideration. At iteration 0, the robot believes it can achieve the goal by simply stacking `b2` on `b3` and `b1` on `b2`, even though it hasn't picked up either block. The first step of this abstract plan (shown in red) is thus unrefinable. At iteration 1, a predicate with the intuitive meaning `Holding` is added, which makes the A* only consider abstract plans that pick up blocks before stacking them. Still, the abstract plan shown is unrefinable on the first step because `b4` is obstructing `b2` in the initial state. At iteration 2, a predicate with the intuitive meaning `NothingAbove` is added, which allows the agent to realize that it must move `b4` out of the way if it wants to pick up `b2`. This plan is still unrefinable, though: the second step fails, because the abstraction still does not recognize that the robot cannot be holding two blocks simultaneously. Finally, at iteration 3, a predicate with the intuitive meaning `HandEmpty` is added, and planning succeeds.

where Ω and Σ are learned using Ψ as we described in Sections A.1 and A.3, PLAN is the algorithm described in Section 4, and TIME$(\cdot)$ measures the time that PLAN takes to find a solution[2]. However, we need an objective that can be used to guide a *search* over candidate predicate sets, meaning the objective must be evaluated many times. J_{real} is far too expensive for this, due to two speed bottlenecks: sampler learning, which involves training several neural networks; and the repeated calls to REFINE from within PLAN, which each perform backtracking search to refine an abstract plan. To overcome this intractability, we will use a *surrogate objective* J_{surr} that is cheaper to evaluate than J_{real}, but that approximately preserves the ordering over predicate sets, i.e., $J_{\text{surr}}(\Psi) < J_{\text{surr}}(\Psi') \iff J_{\text{real}}(\Psi) < J_{\text{real}}(\Psi')$.

We propose a surrogate objective that uses the demonstrations $\mathcal{D}$ to *estimate* the time it would take to solve the training tasks under the abstraction induced by a candidate predicate set Ψ, without using samplers or doing refinement. Recalling that $\mathcal{D}$ has one demonstration π^* for each training task $\langle \mathcal{O}, x_0, g \rangle$, the objective is defined as follows:

$$J_{\text{surr}}(\Psi) \triangleq \frac{1}{|\mathcal{D}|} \sum_{(\mathcal{O}, x_0, g, \pi^*) \in \mathcal{D}} [\text{ETPT}(x_0, g, \Psi, \Omega, \pi^*)],$$

where ETPT abbreviates Estimate Total Planning Time (Algorithm 2). ETPT uses the candidate predicates and induced operators to perform the first part of bilevel planning: A* search over abstract plans. However, for each generated abstract plan, rather than learning samplers and calling REFINE, we use the available demonstrations to estimate the probability that refinement *would* succeed if we *were* to learn samplers and call REFINE. Since bilevel planning terminates upon the successful refinement of an abstract plan, we can use these probabilities to approximate the total expected planning time. We now describe these steps in detail.

Estimating Refinement Probability ETPT maintains a probability $p_{\text{terminate}}$, initialized to 0 (Line 2), that planning would terminate after each generated abstract plan. To update $p_{\text{terminate}}$ (Lines 5-6), we must estimate both whether PLAN would have terminated *before* this step, and whether PLAN would terminate *on* this step. For the former, we can use $(1 - p_{\text{terminate}})$. For the latter, since PLAN terminates only if REFINE succeeds, we use a helper function ESTIMATEREFINEPROB to approximate the probability of successfully refining the given abstract plan, if we were to learn samplers Σ and then call REFINE. We use the following implementation:

$$\text{ESTIMATEREFINEPROB}(\hat{\pi}, \pi^*) \triangleq (1 - \epsilon)\epsilon^{|\text{COST}(\hat{\pi}) - \text{COST}(\pi^*)|}.$$

Here, $\epsilon > 0$ is a small constant (10^{-5} in our experiments), and COST$(\cdot)$ is in our case simply the number of actions in the plan, due to unitary costs. The intuition for this geometric distribution is as follows. Since the demonstration π^* is assumed to be near-optimal, an abstract plan $\hat{\pi}$ that is cheaper than π^* should look suspicious; if such a $\hat{\pi}$ were refinable, then the demonstrator would have likely used it to produce a better demonstration. If $\hat{\pi}$ is more expensive than π^*, then even though this abstraction would eventually produce a refinable abstract plan, it may take a long time for the outer loop of the planner, GENABSTRACTPLAN, to get to it (Section 4). We note that this scheme for estimating refinability is surprisingly minimal, in that it needs only the cost of each demonstration rather than its contents.

Estimating Time To approximate the total planning time, ETPT estimates the time required for each generated abstract plan, conditioned on its successful refinement, and then uses the refinement probabilities to compute the total expectation. The time estimate is maintained in t_{expected}, initialized to 0 (Line 3). To update t_{expected} on each abstract

[2]If no plan can be found (e.g., a task is infeasible under the abstraction), TIME would return a large constant representing a timeout.

plan (Lines 7-8), we use a helper function ESTIMATETIME, which sums together estimates of the *abstract search time* and of the *refinement time*. Since we are running abstract search, we could exactly measure its time; however, to avoid noise due to CPU speed, we instead use the cumulative number of nodes created by the A* search. To estimate refinement time, recall that REFINE performs a backtracking search, and so over many calls to REFINE, the potentially several that fail will dominate the one or zero that succeed. Therefore, we estimate refinement time as a large constant (10^3 in our experiments) that captures the average cost of an exhaustive backtracking search. Finally, we use a large constant t_{upper} (10^5 in our experiments) to penalize in the case where no abstract plan succeeds (Line 9).

What is the ideal choice for n_{abstract}, the maximum number of abstract plans to consider within ETPT? From an efficiency perspective, $n_{\text{abstract}} = 1$ is ideal, but otherwise, it is not obvious whether to prefer the value of n_{abstract} that will eventually be used with PLAN at evaluation time, or to instead prefer $n_{\text{abstract}} = \infty$. On one hand, we want ETPT to be as much of a mirror image of PLAN as possible; on the other hand, some experimentation we conducted suggests that a larger value of n_{abstract} can smooth the objective landscape, which makes search easier. In practice, it may be advisable to treat n_{abstract} as a hyperparameter.

In summary, our surrogate objective J_{surr} calculates and combines two characteristics of a candidate predicate set Ψ: (1) abstract plan cost "error," i.e., $|\text{COST}(\hat{\pi}) - \text{COST}(\pi^*)|$; and (2) abstract planning time, i.e., number of nodes created during A*. The first feature uses only the costs of the demonstrated plans, while the second feature does not use the demonstrated plans at all. In Appendix A.7, we conduct an empirical analysis to further unpack the contribution of these two features to the overall surrogate objective, finding them to be helpful together but insufficient individually.

5.2 Local Search over Candidate Predicate Sets

With our surrogate objective J_{surr} established, we turn to the question of how to best optimize it. We use a simple hill-climbing search, initialized with $\Psi_0 \leftarrow \Psi_G$, and adding a single new predicate ψ from the pool on each step i:

$$\Psi_{i+1} \leftarrow \underset{\psi \notin \Psi_i}{\text{argmin}}\, J_{\text{surr}}(\Psi_i \cup \{\psi\}).$$

We repeat until no improvement can be found, and use the last predicate set as our final Ψ. See Figure 3 for an example taken from our experiments in the Blocks environment.

Designing a Grammar of Predicates Designing a grammar of predicates can be difficult, since there is a tradeoff between the expressivity of the grammar and the practicality of searching over it. For our experiments, we found that a simple grammar similar to that of Pasula, Zettlemoyer, and Kaelbling (2007) suffices, which includes single-feature inequalities, logical negation, and universal quantification. See Section A.4 for a full description and Figure 1 and Appendix A.7 for examples.

The costs accumulated over the production rules lead us to a final cost associated with each predicate ψ, denoted $\text{PEN}(\psi)$, where a higher cost represents a predicate with higher complexity. We use the costs to regularize J_{surr} during local search, with a weight small enough to primarily prevent the addition of "neutral" predicates that neither harm nor hurt J_{surr}. The regularization term is $J_{\text{reg}}(\Psi) \triangleq w_{\text{reg}} \sum_{\psi \in \Psi} \text{PEN}(\psi)$, where $w_{\text{reg}} = 10^{-4}$ in our experiments. To generate our candidate predicate set for local search, we enumerate n_{grammar} (200 in experiments) predicates from the grammar, in order of increasing cost.

6 Experiments

Our experiments are designed to answer the following questions: **(Q1)** To what extent do our learned abstractions help both the effectiveness and the efficiency of planning, and how do they compare to abstractions learned using other objective functions? **(Q2)** How do our learned state abstractions compare in performance to manually designed state abstractions? **(Q3)** How data-efficient is learning, with respect to the number of demonstrations? **(Q4)** Do our abstractions vary as we change the planner configuration, and if so, how?

Experimental Setup We evaluate 10 methods across four robotic planning environments. All results are averaged over 10 random seeds. For each seed, we sample a set of 50 *evaluation tasks* that involve more objects and harder goals than were seen at training. Demonstrations are collected by bilevel planning with manually defined abstractions (see Manual method below). Planning is always limited to a 10-second timeout. See Appendix A.6 for additional details.

Environments We now briefly describe the environments, with further details in Appendix A.5. The first three environments were established in prior work by Silver et al. (2021), but in that work, all predicates were manually defined; we use the same predicates in the Manual baseline.

- **PickPlace1D.** A robot must pick blocks and place them onto target regions along a table surface. All pick and place poses are in a 1D line. Evaluation tasks require 1-4 actions to solve.
- **Blocks.** A robot in 3D must interact with blocks on a table to assemble them into towers. This is a robotic version of the classic blocks world domain. Evaluation tasks require 2-20 actions to solve.
- **Painting.** A robot in 3D must pick, wash, dry, paint, and place widgets into either a box or a shelf. Evaluation tasks require 11-25 actions to solve.
- **Tools.** A robot operating on a 2D table surface must assemble contraptions with screws, nails, and bolts, using a provided set of screwdrivers, hammers, and wrenches respectively. This environment has physical constraints that cannot be modeled by our predicate grammar. Evaluation tasks require 7-20 actions to solve.

Methods We evaluate our method, six baselines, a manually designed state abstraction, and two ablations. Note that the Bisimulation, Branching, Boltzmann, and Manual baselines differ from Ours only in predicate learning.

- **Ours.** Our main approach.
- **Bisimulation.** A baseline that learns abstractions by approximately optimizing the *bisimulation criteria* (Givan,

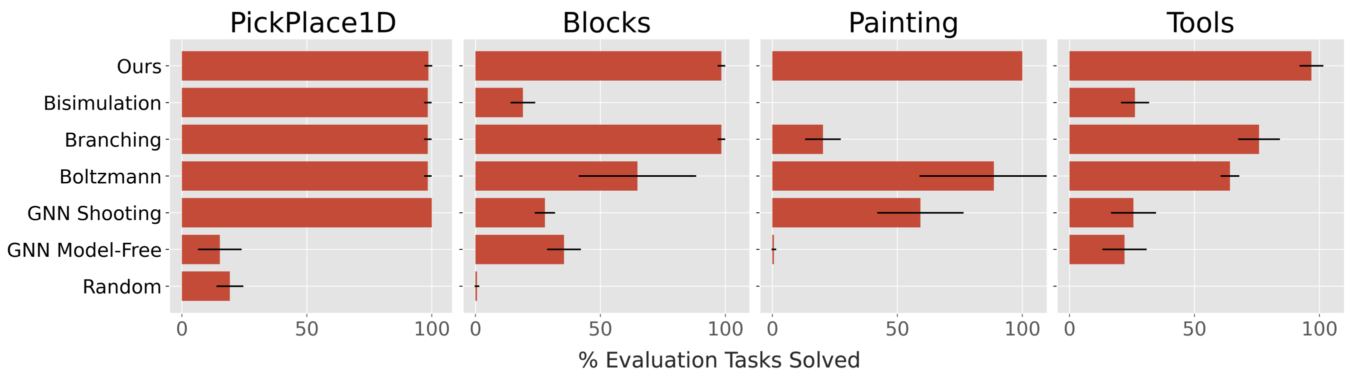


Figure 4: Ours versus baselines. Percentage of 50 evaluation tasks solved under a 10-second timeout, for all four environments. All results are averaged over 10 seeds. Black bars denote standard deviations. Learning times and additional metrics are reported in Appendix A.7.

	Ours			**Manual**			**Down Eval**			**No Invent**		
Environment	Succ	Node	Time	Succ	Node	Time	Succ	Node	Time	Succ	Node	Time
PickPlace1D	98.6	4.8	0.006	98.4	6.5	0.045	98.6	4.8	0.008	39.6	14.1	1.369
Blocks	98.4	2949	0.296	98.6	2941	0.251	98.2	2949	0.318	3.2	427.7	1.235
Painting	100.0	501.8	0.470	99.6	2608	0.464	98.8	489.0	0.208	0.0	–	–
Tools	96.8	1897	0.457	100.0	4771	0.491	42.8	152.5	0.060	0.0	–	–

Table 1: Ours versus Manual and ablations. Percentage of 50 evaluation tasks solved under a 10-second timeout (Succ), number of nodes created during GENABSTRACTPLAN (Node), and wall-clock planning time in seconds (Time). All results are averaged over 10 seeds. The Node and Time columns average over *solved tasks only*. Standard deviations are provided in Appendix A.7.

Dean, and Greig 2003), as in prior work (Curtis et al. 2021). Specifically, this baseline learns abstractions that minimize the number of transitions in the demonstrations where the abstract transition model F is applicable but makes a misprediction about the next abstract state. Note that because goal predicates are given, goal distinguishability is satisfied under any abstraction.

- **Branching.** A baseline that learns abstractions by optimizing the *branching factor* of planning. Specifically, this baseline learns predicates that minimize the number of applicable operators over demonstration states.
- **Boltzmann.** A baseline that assumes the demonstrator is acting *noisily rationally* under (unknown) optimal abstractions (Baker, Saxe, and Tenenbaum 2009). For any candidate abstraction, we compute the likelihood of the demonstration under a Boltzmann policy using the planning heuristic as a surrogate for the true cost-to-go.
- **GNN Shooting.** A baseline that trains a graph neural network (Battaglia et al. 2018) policy. This GNN takes in the current state x, abstract state s, and goal g. It outputs an action a, via a one-hot vector over $\mathcal{C}$ corresponding to which controller to execute, one-hot vectors over all objects at each discrete argument position, and a vector of continuous arguments. We train the GNN using behavior cloning on the data $\mathcal{D}$. At evaluation time, we sample trajectories by treating the outputted continuous arguments as the mean of a Gaussian with fixed variance. We use the transition model f to check if the goal is achieved, and repeat until the planning timeout is reached.
- **GNN Model-Free.** A baseline that uses the same GNN, but directly executes the policy instead of shooting.
- **Random.** A baseline that simply executes a random controller with random arguments on each step. No learning.
- **Manual.** An oracle approach that plans with manually designed predicates for each environment.
- **Down Eval.** An ablation of Ours that uses $n_{\text{abstract}} = 1$ during evaluation only, in PLAN (Algorithm 1).
- **No Invent.** An ablation of Ours that uses $\Psi = \Psi_G$, i.e., only goal predicates are used for the state abstraction.

Results and Discussion We provide real examples of learned predicates and operators for all environments in Appendix A.7. Figure 4 shows that our method solves many more held-out tasks within the timeout than the baselines. A major reason for this performance gap is that our surrogate objective J_{surr} explicitly approximates the efficiency of planning. The lackluster performance of the bisimulation baseline is especially notable because of its prevalence in the literature (Pasula, Zettlemoyer, and Kaelbling 2007; Jetchev, Lang, and Toussaint 2013; Bonet and Geffner 2019; Curtis et al. 2021). We examined its failure modes more closely and found that it consistently selects good predicates, but not *enough* of them. This is because requiring the operators to be a perfect predictive model in the abstract spaces is often not enough to ensure good planning performance. For example, in the Blocks environment, the goal predicates together with the predicate `Holding(?block)` are enough to satisfy bisimulation on our data, while other predicates like `Clear(?block)` and `HandEmpty()` are useful from a planning perspective. Examining the GNN baselines, we see that while shooting is beneficial versus using the GNN model-free, the performance is generally far worse than Ours. Additional experimentation we conducted sug-

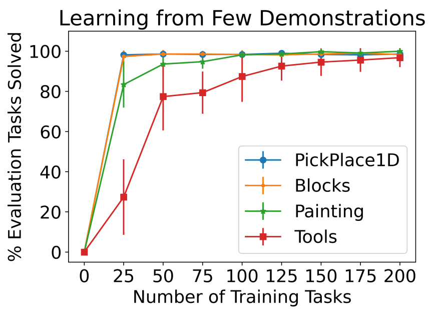


Figure 5: Data-efficiency of our approach.

gests that the GNN gets better with around an order of magnitude more data.

Figure 5 illustrates the data efficiency of Ours. Each point shows a mean over 10 seeds, with standard deviations shown as vertical bars. We often obtain very good evaluation performance within just 50 demonstrations.

In Table 1, the results for No Invent show that, as expected, the goal predicates alone are completely insufficient for most tasks. Comparing Ours to Down Eval shows that assuming downward refinability at evaluation time works for PickPlace1D, Blocks, and Painting, but not for Tools. We also find that the learned predicates (Ours) are on par with, and sometimes *better than*, hand-designed predicates (Manual). For instance, consider PickPlace1D, where the learned predicates are 7.5x better. The manually designed predicates were `Held(?block)` and `HandEmpty()`, and the always-given goal predicate `Covers(?block, ?target)`. In addition to inventing two predicates that are equivalent to `Held` and `HandEmpty`, Ours invented two more: `P3(?block)` $\triangleq$ $\forall$`?t.`$\neg$`Covers(?block, ?t)`, and `P4(?target)` $\triangleq$ $\forall$`?b.`$\neg$`Covers(?b, ?target)`. Intuitively, `P3` means "the given block is not on any target," while `P4` means "the given target is clear." `P3` gets used in an operator precondition for picking, which reduces the branching factor of abstract search. This precondition is sensible because there is no use in moving a block once it is already on its target. `P4` prevents considering non-refinable abstract plans that "park" objects on targets that must be covered by other objects.

In Appendix A.8, we describe an additional experiment where we vary the AI planning heuristic used in abstract search. We analyze a case in Blocks where variation in the invented predicates appears inconsequential upon initial inspection, but actually has substantial impact on planning efficiency. This result underscores the benefit of using a surrogate objective for predicate invention that is sensitive to downstream planning efficiency.

7 Related Work

Our work continues a long line of research on learning state abstractions for decision-making (Bertsekas, Castanon et al. 1988; Andre and Russell 2002; Jong and Stone 2005; Li, Walsh, and Littman 2006; Abel, Hershkowitz, and Littman 2017; Zhang et al. 2020). Most relevant are works that learn symbolic abstractions compatible with AI planners (Lang, Toussaint, and Kersting 2012; Jetchev, Lang, and Toussaint 2013; Ugur and Piater 2015; Asai and Fukunaga 2018; Bonet and Geffner 2019; Asai and Muise 2020; Ahmetoglu et al. 2020; Umili et al. 2021). Our work is particularly influenced by Pasula, Zettlemoyer, and Kaelbling (2007), who use search through a concept language to invent symbolic state and action abstractions, and Konidaris, Kaelbling, and Lozano-Perez (2018), who discover symbolic abstractions by leveraging the initiation and termination sets of options that satisfy an abstract subgoal property. The objectives used in these prior works are based on variations of autoencoding, prediction error, or bisimulation, which stem from the perspective that the abstractions should *replace* planning in the original transition space, rather than *guide* it.

Recent works have also considered learning abstractions for multi-level planning, like those in the task and motion planning (TAMP) (Gravot, Cambon, and Alami 2005; Garrett et al. 2021) and hierarchical planning (Bercher, Alford, and Höller 2019) literature. Some of these efforts consider learning symbolic action abstractions (Zhuo et al. 2009; Nguyen et al. 2017; Silver et al. 2021; Aineto, Jiménez, and Onaindia 2022) or refinement strategies (Chitnis et al. 2016; Mandalika et al. 2019; Chitnis, Kaelbling, and Lozano-Pérez 2019; Wang et al. 2021; Chitnis et al. 2022; Ortiz-Haro et al. 2022); our operator and sampler learning methods take inspiration from these prior works. Recent efforts by Loula et al. (2019) and Curtis et al. (2021) consider learning both state and action abstractions for TAMP, like we do (Loula et al. 2019, 2020; Curtis et al. 2021). The main distinguishing feature of our work is that our abstraction learning framework explicitly optimizes an objective that considers downstream planning efficiency.

8 Conclusion and Future Work

In this paper, we have described a method for learning predicates that are explicitly optimized for efficient bilevel planning. Key areas for future work include (1) learning better abstractions from even fewer demonstrations by performing active learning to gather more data online; (2) expanding the expressivity of the grammar to learn more sophisticated predicates; (3) applying these ideas to partially observed planning problems; and (4) learning the controllers that we assumed given in this work.

For (1), we hope to investigate how relational exploration algorithms (Chitnis et al. 2020) might be useful as a mechanism for an agent to decide what actions to execute, toward the goal of building better state and action abstractions. For (2), we can take inspiration from program synthesis, especially methods that can learn programs with continuous parameters (Ellis et al. 2020). For (3) we could draw insights from recent advances in task and motion planning in the partially observed setting (Garrett et al. 2020). Finally, for (4), we recently proposed a method for learning controllers from demonstrations assuming known predicates (Silver et al. 2022). If we can remove the latter assumption, we will have a complete pipeline for learning predicates, operators, samplers, and controllers for bilevel planning.

Acknowledgements

We gratefully acknowledge support from NSF grant 2214177; from AFOSR grant FA9550-22-1-0249; from ONR MURI grant N00014-22-1-2740; from the MIT-IBM Watson Lab; and from the MIT Quest for Intelligence. Tom, Nishanth and Willie are supported by NSF Graduate Research Fellowships. We thank Michael Katz, Christian Muise, Aidan Curtis, Jiayuan Mao, Zhutian Yang, and Amber Li for helpful comments on an earlier draft.

References

Abel, D.; Hershkowitz, D. E.; and Littman, M. L. 2017. Near optimal behavior via approximate state abstraction. *arXiv preprint arXiv:1701.04113*.

Ahmetoglu, A.; Seker, M. Y.; Piater, J.; Oztop, E.; and Ugur, E. 2020. Deepsym: Deep symbol generation and rule learning from unsupervised continuous robot interaction for planning. *arXiv preprint arXiv:2012.02532*.

Aineto, D.; Jiménez, S.; and Onaindia, E. 2022. A Comprehensive Framework for Learning Declarative Action Models. *Journal of Artificial Intelligence Research*, 74: 1091–1123.

Andre, D.; and Russell, S. J. 2002. State abstraction for programmable reinforcement learning agents. In *AAAI/IAAI*, 119–125.

Asai, M.; and Fukunaga, A. 2018. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *AAAI*.

Asai, M.; and Muise, C. 2020. Learning neural-symbolic descriptive planning models via cube-space priors: The voyage home (to STRIPS). *arXiv preprint arXiv:2004.12850*.

Bacchus, F. 2001. AIPS 2000 planning competition: The fifth international conference on artificial intelligence planning and scheduling systems. *Ai magazine*, 22(3): 47–47.

Baker, C. L.; Saxe, R.; and Tenenbaum, J. B. 2009. Action understanding as inverse planning. *Cognition*.

Battaglia, P. W.; Hamrick, J. B.; Bapst, V.; Sanchez-Gonzalez, A.; Zambaldi, V.; Malinowski, M.; Tacchetti, A.; Raposo, D.; Santoro, A.; Faulkner, R.; et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.

Bercher, P.; Alford, R.; and Höller, D. 2019. A Survey on Hierarchical Planning-One Abstract Idea, Many Concrete Realizations. In *IJCAI*, 6267–6275.

Bertsekas, D. P.; Castanon, D. A.; et al. 1988. Adaptive aggregation methods for infinite horizon dynamic programming. *IEEE Transactions on Automatic Control*.

Bonet, B.; and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence*, 129(1-2): 5–33.

Bonet, B.; and Geffner, H. 2019. Learning first-order symbolic representations for planning from the structure of the state space. *arXiv preprint arXiv:1909.05546*.

Chitnis, R.; Hadfield-Menell, D.; Gupta, A.; Srivastava, S.; Groshev, E.; Lin, C.; and Abbeel, P. 2016. Guided search for task and motion plans using learned heuristics. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 447–454. IEEE.

Chitnis, R.; Kaelbling, L. P.; and Lozano-Pérez, T. 2019. Learning quickly to plan quickly using modular meta-learning. In *2019 International Conference on Robotics and Automation (ICRA)*, 7865–7871. IEEE.

Chitnis, R.; Silver, T.; Tenenbaum, J.; Kaelbling, L. P.; and Lozano-Pérez, T. 2020. GLIB: Efficient exploration for relational model-based reinforcement learning via goal-literal babbling. *arXiv preprint arXiv:2001.08299*.

Chitnis, R.; Silver, T.; Tenenbaum, J. B.; Lozano-Pérez, T.; and Kaelbling, L. P. 2022. Learning Neuro-Symbolic Relational Transition Models for Bilevel Planning. In *The IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Clevert, D.-A.; Unterthiner, T.; and Hochreiter, S. 2015. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*.

Cropper, A.; and Muggleton, S. H. 2016. Learning Higher-Order Logic Programs through Abstraction and Invention. In *IJCAI*, 1418–1424.

Curtis, A.; Silver, T.; Tenenbaum, J. B.; Lozano-Perez, T.; and Kaelbling, L. P. 2021. Discovering State and Action Abstractions for Generalized Task and Motion Planning. *arXiv preprint arXiv:2109.11082*.

Dantam, N. T.; Kingston, Z. K.; Chaudhuri, S.; and Kavraki, L. E. 2016. Incremental task and motion planning: A constraint-based approach. In *Robotics: Science and systems*, volume 12, 00052. Ann Arbor, MI, USA.

Ellis, K.; Wong, C.; Nye, M.; Sable-Meyer, M.; Cary, L.; Morales, L.; Hewitt, L.; Solar-Lezama, A.; and Tenenbaum, J. B. 2020. Dreamcoder: Growing generalizable, interpretable knowledge with wake-sleep bayesian program learning. *arXiv preprint arXiv:2006.08381*.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4): 189–208.

Garrett, C. R.; Chitnis, R.; Holladay, R.; Kim, B.; Silver, T.; Kaelbling, L. P.; and Lozano-Pérez, T. 2021. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4: 265–293.

Garrett, C. R.; Paxton, C.; Lozano-Pérez, T.; Kaelbling, L. P.; and Fox, D. 2020. Online replanning in belief space for partially observable task and motion problems. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 5678–5684. IEEE.

Givan, R.; Dean, T.; and Greig, M. 2003. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 147(1-2): 163–223.

Gravot, F.; Cambon, S.; and Alami, R. 2005. aSyMov: a planner that deals with intricate symbolic and geometric problems. In *Robotics Research. The Eleventh International Symposium*, 100–110.

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

Helmert, M.; and Domshlak, C. 2009. Landmarks, critical paths and abstractions: what's the difference anyway? In *Nineteenth International Conference on Automated Planning and Scheduling*.

Jetchev, N.; Lang, T.; and Toussaint, M. 2013. Learning grounded relational symbols from continuous data for abstract reasoning. In *Proceedings of the 2013 ICRA Workshop on Autonomous Learning*.

Jong, N. K.; and Stone, P. 2005. State Abstraction Discovery from Irrelevant State Variables. In *IJCAI*.

Katz, M.; Sohrabi, S.; Udrea, O.; and Winterer, D. 2018. A Novel Iterative Approach to Top-k Planning. In *Proceedings of the Twenty-Eigth International Conference on Automated Planning and Scheduling (ICAPS 2018)*. AAAI Press.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Konidaris, G.; and Barto, A. 2009. Efficient skill learning using abstraction selection. In *Twenty-First International Joint Conference on Artificial Intelligence*.

Konidaris, G.; Kaelbling, L. P.; and Lozano-Perez, T. 2018. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61: 215–289.

Lang, T.; Toussaint, M.; and Kersting, K. 2012. Exploration in relational domains for model-based reinforcement learning. *The Journal of Machine Learning Research*, 13(1): 3725–3768.

Lavrac, N.; and Dzeroski, S. 1994. Inductive Logic Programming. In *Logic Programming Workshop*, 146–160.

Li, L.; Walsh, T. J.; and Littman, M. L. 2006. Towards a Unified Theory of State Abstraction for MDPs. *ISAIM*.

Loula, J.; Allen, K.; Silver, T.; and Tenenbaum, J. 2020. Learning constraint-based planning models from demonstrations. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 5410–5416. IEEE.

Loula, J.; Silver, T.; Allen, K. R.; and Tenenbaum, J. 2019. Discovering a symbolic planning language from continuous experience. In *Annual Meeting of the Cognitive Science Society (CogSci)*, 2193.

Mandalika, A.; Choudhury, S.; Salzman, O.; and Srinivasa, S. 2019. Generalized lazy search for robot motion planning: Interleaving search and edge evaluation via event-based toggles. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 745–753.

Marthi, B.; Russell, S. J.; and Wolfe, J. A. 2007. Angelic Semantics for High-Level Actions. In *ICAPS*, 232–239.

Menon, A.; Tamuz, O.; Gulwani, S.; Lampson, B.; and Kalai, A. 2013. A machine learning framework for programming by example. In *International Conference on Machine Learning*, 187–195. PMLR.

Nguyen, C.; Reifsnyder, N.; Gopalakrishnan, S.; and Munoz-Avila, H. 2017. Automated learning of hierarchical task networks for controlling minecraft agents. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, 226–231. IEEE.

Ortiz-Haro, J.; Ha, J.-S.; Driess, D.; and Toussaint, M. 2022. Structured deep generative models for sampling on constraint manifolds in sequential manipulation. In *Conference on Robot Learning*, 213–223. PMLR.

Pasula, H. M.; Zettlemoyer, L. S.; and Kaelbling, L. P. 2007. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research*, 29: 309–352.

Ramírez, M.; and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Ren, T.; Chalvatzaki, G.; and Peters, J. 2021. Extended Tree Search for Robot Task and Motion Planning. *arXiv preprint arXiv:2103.05456*.

Silver, T.; Athalye, A.; Tenenbaum, J. B.; Lozano-Perez, T.; and Kaelbling, L. P. 2022. Learning Neuro-Symbolic Skills for Bilevel Planning. In *Conference on Robot Learning*.

Silver, T.; Chitnis, R.; Tenenbaum, J.; Kaelbling, L. P.; and Lozano-Pérez, T. 2021. Learning symbolic operators for task and motion planning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3182–3189. IEEE.

Srivastava, S.; Fang, E.; Riano, L.; Chitnis, R.; Russell, S.; and Abbeel, P. 2014. Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE international conference on robotics and automation (ICRA)*, 639–646. IEEE.

Stahl, I. 1993. Predicate invention in ILP—an overview. In *European Conference on Machine Learning*, 311–322.

Ugur, E.; and Piater, J. 2015. Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2627–2633. IEEE.

Umili, E.; Antonioni, E.; Riccio, F.; Capobianco, R.; Nardi, D.; and De Giacomo, G. 2021. Learning a Symbolic Planning Domain through the Interaction with Continuous Environments. *ICAPS PRL Workshop*.

Wang, Z.; Garrett, C. R.; Kaelbling, L. P.; and Lozano-Pérez, T. 2021. Learning compositional models of robot skills for task and motion planning. *The International Journal of Robotics Research*, 40(6-7): 866–894.

Zhang, A.; McAllister, R.; Calandra, R.; Gal, Y.; and Levine, S. 2020. Learning invariant representations for reinforcement learning without reconstruction. *arXiv preprint arXiv:2006.10742*.

Zhi-Xuan, T.; Mann, J.; Silver, T.; Tenenbaum, J.; and Mansinghka, V. 2020. Online bayesian goal inference for boundedly rational planning agents. *Advances in Neural Information Processing Systems*, 33: 19238–19250.

Zhuo, H. H.; Hu, D. H.; Hogg, C.; Yang, Q.; and Munoz-Avila, H. 2009. Learning HTN method preconditions and action models from partial observations. In *Twenty-First International Joint Conference on Artificial Intelligence*.