Learning Sketches for Decomposing Planning Problems into Subproblems of Bounded Width

Dominik Drexler[1], **Jendrik Seipp**[1], **Hector Geffner**[3,2,1]

[1]Linköping University, Linköping, Sweden
[2]Universitat Pompeu Fabra, Barcelona, Spain
[3]Institució Catalana de Recerca i Estudis Avançats (ICREA), Barcelona, Spain
{dominik.drexler, jendrik.seipp}@liu.se, hector.geffner@upf.edu

Abstract

Recently, sketches have been introduced as a general language for representing the subgoal structure of instances drawn from the same domain. Sketches are collections of rules of the form $C \mapsto E$ over a given set of features where C expresses Boolean conditions and E expresses qualitative changes. Each sketch rule defines a subproblem: going from a state that satisfies C to a state that achieves the change expressed by E or a goal state. Sketches can encode simple goal serializations, general policies, or decompositions of bounded width that can be solved greedily, in polynomial time, by the SIW$_\text{R}$ variant of the SIW algorithm. Previous work has shown the computational value of sketches over benchmark domains that, while tractable, are challenging for domain-independent planners. In this work, we address the problem of learning sketches automatically given a planning domain, some instances of the target class of problems, and the desired bound on the sketch width. We present a logical formulation of the problem, an implementation using the ASP solver Clingo, and experimental results. The sketch learner and the SIW$_\text{R}$ planner yield a domain-independent planner that learns and exploits domain structure in a crisp and explicit form.

Introduction

Classical planners manage to solve problems that span exponentially large state spaces by exploiting problem structure. Domain-independent methods make implicit assumptions about structure, such as subgoals being independent (delete-relaxation) or having low width (width-based search). Domain-dependent methods, on the other hand, usually make problem structure explicit in the form of hierarchies that express how tasks decompose into subtasks (Erol, Hendler, and Nau 1994; Georgievski and Aiello 2015; Bercher, Alford, and Höller 2019).

An alternative, simpler language for representing problem structure explicitly has been introduced recently in the form of *sketches* (Bonet and Geffner 2021). Sketches are collections of rules of the form $C \mapsto E$ defined over a given set of Boolean and numerical domain features Φ where C expresses Boolean conditions on the features, and E expresses qualitative changes in their values. Each sketch rule expresses a subproblem: the problem of going from a state s whose feature values satisfy the condition C, to a state s' where the feature values change in agreement with E.

The language of sketches is powerful, as sketches can encode everything from simple goal serializations to full general policies. Indeed, the language of general policies is the language of sketches but with a slightly different semantics where the subgoal states s' to be reached from a state s are restricted to be one step away from s (Bonet and Geffner 2018; Francès, Bonet, and Geffner 2021). More interestingly, sketches can split problems into subproblems of *bounded width* (Lipovetzky and Geffner 2012; Lipovetzky 2021) which can then be solved greedily, in polynomial time, by a variant of the SIW algorithm, called SIW$_\text{R}$ (Bonet and Geffner 2021). The computational value of sketches crafted by hand has been shown over several planning domains which, while tractable, are challenging for domain-independent planners (Drexler, Seipp, and Geffner 2021).

In this work, we build on these threads (general policies, sketches, and width) to address the problem of *learning sketches automatically*. For this, the inputs are the planning domain, some instances, and the desired bound k on width, usually $k = 0, 1, 2$. The width k of the sketch relative to a class of problems bounds the width of the resulting subproblems that can be solved greedily in time and space that are exponential in k. In order to address the problem of learning sketches, we present a logical formulation, an implementation of the learner on top of the ASP solver Clingo (Gebser et al. 2019), and experimental results. We start with an example and a review of planning, width, and sketches.

Example

For an illustration of the concepts and results, before presenting the formal definitions, let us recall a simple domain called Delivery (Bonet and Geffner 2021), where an agent moves in a grid to pick up packages and deliver them to a target cell, one by one. A general policy π for this class of instances can be expressed in terms of the set of features $\Phi = \{H, p, t, n\}$ that express "holding a package", "distance to the nearest package", "distance to the target cell", and "number of undelivered packages", respectively. A domain-independent method for generating a large pool of features from the domain predicates that include these four is given by Bonet, Francès, and Geffner (2019). Provided with the features in Φ, a **general policy** for the whole class $\mathcal{Q}_D$ of Delivery problems (any grid size, any number of packages in any location, and any location of the agent or the target) is given by the rules:

$\{\neg H, p > 0\} \mapsto \{p\downarrow, t?\}$; go to nearest pkg
$\{\neg H, p = 0\} \mapsto \{H\}$; pick it up
$\{H, t > 0\} \mapsto \{t\downarrow\}$; go to target
$\{H, n > 0, t = 0\} \mapsto \{H?, n\downarrow, p?\}$; deliver pkg

The rules say to perform any action that decreases the distance to the nearest package ($p\downarrow$), no matter the effect on the distance to target ($t?$), when not holding a package ($\neg H$); to pick a package when possible, making H true; to go to the target cell when holding a package, decrementing t; and to drop the package at the target, decrementing n, making H false, and affecting p. Expressions $p?$ and $m?$ mean that p and m can change in any way, while no mention of a feature in the effect of a rule means that the value of the feature must not change.

One can show that the general policy above solves any instance of Delivery, or alternatively, that it represents a **width-zero sketch**, where the subproblem of going from a non-goal state s to a state s' satisfying a rule $C \mapsto E$ can always be done in one step, leading greedily to the goal ($n = 0$). A pair of states (s, s') satisfies a rule $C \mapsto E$ when the feature values in s satisfy the conditions in C, and the change in feature values from s to s' is compatible with the changes and no-changes expressed in E. A **width-2 sketch** can be defined instead by means of a single rule and a single feature $\Phi = \{n\}$:

$\{n > 0\} \mapsto \{n\downarrow\}$; deliver pkg

The subproblem of going from a state s where $n(s) > 0$ holds, to a state s' for which $n(s') < n(s)$ holds, has width bounded by 2, as these subproblems, in the worst case, involve moving to the location of a package, picking it up, moving to the target, and dropping it. Halfway between the width-0 sketch represented by general policies, and the width-2 sketch above, is the **width-1 sketch** defined with two rules and two features, $\Phi = \{H, n\}$ as:

$\{\neg H\} \mapsto \{H\}$; pick pkg
$\{H, n > 0\} \mapsto \{H?, n\downarrow\}$; deliver pkg

The first rule captures the subproblem of getting hold of a package, which may involve moving to the package and picking it up, with width 1; the second rule captures the subproblem of delivering the package being held, which may involve moving to the target and dropping it there, also with width 1. Since every non-goal state is covered by one rule or the other, the width of the sketch is 1, which means that any instance of Delivery can be solved greedily by solving subproblems of width no greater than 1 in linear time.

These are all handcrafted sketches. The methods to be formulated below will learn similar sketches when given the planning domain, a few instances, and the desired bound on sketch width.

Background

We review classical planning, width, and sketches drawing from Lipovetzky and Geffner (2012), Bonet and Geffner (2021), and Drexler, Seipp, and Geffner (2021).

Classical Planning

A *planning problem* or *instance* is a pair $P = (D, I)$ where D is a first-order *domain* with action schemas defined over predicates, and I contains the objects in the instance and two sets of ground literals, the initial and goal situations *Init* and *Goal*. The initial situation is consistent and complete, meaning either a ground literal or its complement is in *Init*. An instance P defines a state model $S(P) = (S, s_0, G, Act, A, f)$ where the states in S are the truth valuations over the ground atoms represented by the set of literals that they make true, the initial state s_0 is *Init*, the set of goal states G are those that make the goal literals in *Goal* true, and the actions *Act* are the ground actions obtained from the schemas and objects. The ground actions in $A(s)$ are the ones that are applicable in a state s; namely, those whose preconditions are true in s, and the state transition function f maps a state s and an action $a \in A(s)$ into the successor state $s' = f(a, s)$. A *plan* π for P is a sequence of actions $a_0, \ldots, a_n$ that is executable in s_0 and maps the initial state s_0 into a goal state; i.e., $a_i \in A(s_i)$, $s_{i+1} = f(a_i, s_i)$, and $s_{n+1} \in G$. A state s is *solvable* if there exists a plan starting at s, otherwise it is *unsolvable* (also called *dead-end*). Furthermore, a state s is *alive* if it is solvable and it is not a goal state. The *length* of a plan is the number of its actions, and a plan is *optimal* if there is no shorter plan. Our objective is to find suboptimal plans for *collections* of instances $P = (D, I)$ over fixed domains D denoted as $\mathcal{Q}_D$ or simply as $\mathcal{Q}$.

Width

The simplest width-based search method for solving a planning problem P is IW(1). It is a standard breadth-first search in the rooted directed graph associated with the state model $S(P)$ with one modification: IW(1) prunes a newly generated state if it does not make an atom true for the first time in the search. The procedure IW(k) for $k > 1$ is like IW(1) but prunes a state if a newly generated state does not make a collection of up to k atoms true for the first time. Underlying the IW algorithms is the notion of *problem width* (Lipovetzky and Geffner 2012):

Definition 1 (Width) *The* ***width*** $w(P)$ *of a classical planning problem* P *is the minimum* k *for which there exists a* ***sequence*** $t_0, t_1, \ldots, t_m$ ***of atom tuples*** t_i *from* P*, each consisting of at most* k *atoms, such that:*

1. t_0 *is true in the initial state* s_0 *of* P*,*
2. *any optimal plan for* t_i *can be extended into an optimal plan for* t_{i+1} *by adding a single action,* $i = 1, \ldots, n-1$*,*
3. *if* π *is an optimal plan for* t_m*,* π *is an optimal plan for* P*.*

If a problem P is unsolvable, $w(P)$ is set to the number of variables in P, and if P is solvable in at most one step, $w(P)$ is set to 0 (Bonet and Geffner 2021). Chains of tuples $\theta = (t_0, t_1, \ldots, t_m)$ that comply with conditions 1–2 are called **admissible**, and the size of θ is the size $|t_i|$ of the largest tuple in the chain. The width $w(P)$ is thus the minimum size of an admissible chain for P that is also optimal (condition 3). Furthermore, the width of a conjunction of atoms T (or arbitrary set of states S' in P) is the width of a problem P' that is like P but with the goal T (resp. S').[1]

The IW(k) algorithm expands up to N^k nodes, generates up to bN^k nodes, and runs in time and space $O(bN^{2k-1})$ and $O(bN^k)$, respectively, where N is the number of atoms and b is a bound on the branching factor of the problem P. IW(k) is guaranteed to solve P optimally if $w(P) \leq k$. If the width of P is not known, the **IW** algorithm can be run instead which calls IW(k) iteratively for $k = 0, 1, \ldots, N$ until the problem is solved, or found to be unsolvable.

For problems with conjunctive goals, the **SIW** algorithm (Lipovetzky and Geffner 2012) starts at the initial state $s = s_0$ of P, and performs an IW search from s to find a shortest path to a state s' such that $\#g(s') < \#g(s)$ where $\#g(s)$ counts the number of unsatisfied top-level goals of P in state s. If s' is not a goal state, s is set to s' and the loop repeats.

Sketches

A **feature** is a function of the state over a class of problems $\mathcal{Q}$. The features considered in the language of sketches are Boolean, taking values in the Boolean domain, or numerical, taking values in the non-negative integers. For a set of features $\Phi = \{f_1, \ldots, f_N\}$ and a state s of some instance P in $\mathcal{Q}$, we denote the **feature valuation** determined by a state s as $f(s) = (f_1(s), \ldots, f_N(s))$, and arbitrary feature valuations as f and f'.

A **sketch rule** over features Φ has the form $C \mapsto E$ where C consists of Boolean feature conditions, and E consists of feature effects. A Boolean (feature) condition is of the form p or $\neg p$ for a Boolean feature p in Φ, or $n = 0$ or $n > 0$ for a numerical feature n in Φ. A feature effect is an expression of the form p, $\neg p$, or $p?$ for a Boolean feature p in Φ, and $n\downarrow$, $n\uparrow$, or $n?$ for a numerical feature n in Φ. The syntax of sketch rules is the syntax of the policy rules used to define generalized policies (Bonet and Geffner 2018), but their semantics is different. In policy rules, the effects have to be delivered in one step by state transitions, while in sketch rules, they can be delivered by longer state sequences.

A pair of feature valuations of two states $(f(s), f(s'))$, referred to as (f, f'), **satisfies a sketch rule** $C \mapsto E$ iff 1) C is true in f, 2) the Boolean effects p ($\neg p$) in E are true in f', 3) the numerical effects are satisfied by the pair (f, f'); i.e., if $n\downarrow$ in E (resp. $n\uparrow$), then the value of n in f' is smaller (resp. larger) than in f, and 4) features that do not occur in E have the same value in f and f'. Adding the effects $p?$ and $n?$ allows the values of features p and n to change in any way. In contrast, the value of features that do not occur in E must be the same in s and s'.

A sketch is a collection of sketch rules that establishes a "preference ordering" '$\prec$' over feature valuations where $f' \prec f$ if the pair of feature valuations (f, f') satisfies a rule. If the sketch is **terminating**, then this preference order is a strict partial order: irreflexive and transitive. Checking termination requires time that is exponential in the number of features (Bonet and Geffner 2021).

Following Bonet and Geffner, we do not use these orderings explicitly but the associated problem **decompositions**. The set of **subgoal states** $G_R(s)$ associated with a sketch R in a state s of a problem $P \in Q$, is the set of states s' that comprise the goal states of P along with those with feature valuation $f(s')$ such that the pair $(f(s), f(s'))$ satisfies a rule in R. The set of states s' in $G_R(s)$ that are closest to s is denoted as $G^*_R(s)$.

Sketch Width

The $\mathbf{SIW_R}$ algorithm is a variant of SIW that uses a given sketch R for solving problems P in $\mathcal{Q}$. $\mathrm{SIW_R}$ starts at the initial state $s = s_0$ of P and then runs an IW search to find a state s' in $G_R(s)$. If s' is not a goal state, then s is set to s', and the loop repeats until a goal state is reached. The $\mathrm{SIW_R}(k)$ algorithm is like $\mathrm{SIW_R}$ but calls the procedure IW(k) internally, not IW.

For bounding the complexity of these algorithms, let us assume without loss of generality that the class of problems $\mathcal{Q}$ is *closed* in the sense that if P belongs to $\mathcal{Q}$ so do the problems P' that are like P but with initial states that are reachable in P and which are not dead-ends. Then the width of the sketch R over $\mathcal{Q}$ can be defined as follows (Bonet and Geffner 2021):[2]

Definition 2 (Sketch width) *The **width of sketch** R over a closed class of problems $\mathcal{Q}$ is $w_R(\mathcal{Q}) = \max_{P \in \mathcal{Q}} w(P')$ where P' is P but with goal states $G^*_R(s)$ and s is the initial state of both, provided that $G^*_R(s)$ does not contain dead-end states.*

If the sketch width is bounded, $\mathrm{SIW_R}(k)$ solves the instances in $\mathcal{Q}$ in polynomial time:[3]

Theorem 3 *If $w_R(\mathcal{Q}) \leq k$ and the sketch R is terminating, then SIW_R (k) solves the instances in $\mathcal{Q}$ in $O(bN^{|\Phi|+2k-1})$ time and $O(bN^k + N^{|\Phi|+k})$ space, where $|\Phi|$ is the number of features, N is the number of ground atoms, and b is the branching factor.*

For these bounds, the features are assumed to be linear in N; namely, they must have at most a linear number of values in each instance, all computable in linear time.

Extensions

We finish this review with a slight generalization of Theorem 3 that is worth making explicit. For this, let us introduce a variant of the notion of width, called *satisficing width*

[1] In the literature, a chain is admissible when it complies with conditions 1–3. The reason for dropping condition 3 will become clear when we introduce the notion of "satisficing width".

[2] Our definition is simpler than those used by Bonet and Geffner (2021), and Drexler, Seipp, and Geffner (2021), as it avoids a recursive condition of the set of subproblems P'. However, it adds an extra condition that involves the dead-end states, the states from which the goal cannot be reached.

[3] Algorithm $\mathrm{SIW_R}(k)$ is needed here instead of $\mathrm{SIW_R}$ because the latter does not ensure that the subproblems P' are solved optimally. This is because IW(k') may solve problems of width k non-optimally if $k' < k$.

or *s-width*. A problem P has satisficing width $\leq k$ if there is an admissible chain of tuples $\tau : t_0, \ldots, t_m$ of size no greater than k such that the optimal plans for tuple t_m are plans for P. In such a case, we say that τ *is an admissible k-chain* for P. The difference to the (standard) notion of width is that optimal plans for t_m are required to be plans for P but not optimal. The result is that IW(k) will solve problems of s-width bounded by k, written $ws(P) \leq k$, but not optimally. A convenient property is that a problem P with s-width bounded by k has the same bound when the set of goal states of P is extended with more states. The length of the plans computed by IW(k) when k bounds the s-width of P is bounded in turn by the length m of the shortest admissible k-chain $t_0, \ldots, t_m$ for P. If the subproblem of reaching a state s' in $G_R(s)$ has s-width k, and $G^k_R(s)$ stands for the states s' in $G_R(s)$ that are no more than m steps away from s, the *satisficing width* of the a sketch R can be defined as:

Definition 4 (Sketch s-width) *The* ***s-width of sketch*** R *over a closed class of solvable problems $\mathcal{Q}$ is bounded by k, $ws_R(\mathcal{Q}) \leq k$, if $\max_{P \in \mathcal{Q}} ws(P') \leq k$ where P' is P but with goal states $G^k_R(s)$, and s is the initial state of P, provided that $G^k_R(s)$ does not contain dead-end states.*

This definition just replaces the width of subproblems by the weaker s-width, and the subgoal states $G^*_R(s)$ by $G^k_R(s)$.

Let us finally say that a sketch R is state (resp. feature) **acyclic** in $\mathcal{Q}$ when there is no sequence of states $s_1, \ldots, s_n$ over a problem in $\mathcal{Q}$ (resp. feature valuations $f(s_0)$, …, $f(s_n)$), $s_{i+1} \in G_R(s_i)$, such that $s_n = s_j$, $j < n$ (resp. $f(s_n) = f(s_j)$, $j < n$). Bonet and Geffner (2021) showed that termination implies feature acyclicity, and it is direct to show that feature acyclicity implies state acyclicity. Theorem 3 can then be rephrased as:

Theorem 5 *If $ws_R(\mathcal{Q}) \leq k$ and the sketch R is (state) acyclic in $\mathcal{Q}$, then SIW$_R$(k) solves the instances in $\mathcal{Q}$ in $O(bN^{|\Phi|+2k-1})$ time and $O(bN^k + N^{|\Phi|+k})$ space, where $|\Phi|$ is the number of features, N is the number of ground atoms, and b is the branching factor.*

Learning Sketches: Formulation

We turn to the problem of learning sketches given a set of instances $\mathcal{P}$ of the target class of problems $\mathcal{Q}$ and the desired bound k on sketch width. We roughly follow the approach for learning general policies (Bonet, Francès, and Geffner 2019; Francès, Bonet, and Geffner 2021) by constructing a theory $T_{k,m}(\mathcal{P}, \mathcal{F})$ from $\mathcal{P}$, k, a bound m on the number of sketch rules, and a finite pool of features $\mathcal{F}$ obtained from the domain predicates and a fixed grammar.

Theory

Symbols s, s', s'', f, v, t, and i refer to reachable states, features, Boolean values, tuples of at most k atoms, and sketch rules $C_i \mapsto E_i$, respectively. States and tuples are unique to each training instance P_j in $\mathcal{P}$ and thus are not tagged with their instance. The variables (atoms) in $T_{k,m}(\mathcal{P}, \mathcal{F})$ are

- $select(f)$ for features f in pool $\mathcal{F}$,
- $cond(i, f, v)$: for f, rule $i = 1, \ldots, m$, v Bool or '?',
- $eff(i, f, v)$: for f, rule $i = 1, \ldots, m$, v Bool or '?',
- $subgoal(s, t)$: for t of width $\leq k$ from s,
- $subgoals(s, t, s')$: if optimal plan from s to t ends in s',
- $sat\_rule(s, s', i)$: for s' reachable from s, $i = 1, \ldots m$.

When true, these atoms represent that f is a feature used in the sketch, that the i-th rule $C_i \mapsto E_i$ has f as a condition with value v, or no condition if $v = ?$, respectively an effect on f with value v, possibly '?',[4] that t is a subgoal selected from s, possibly leading to state s', and that the transition from s to s' (not necessarily a 1-step transition) satisfies rule i. **Constraints** C1–C8 capture these meanings:

C1 $cond(i, f, v)$, $eff(i, f, v)$ use unique v, imply $select(f)$
C2 $\vee_t subgoal(s, t)$, each alive s has some subgoal t
C3 $subgoal(s, t)$ iff $\wedge_{s'} subgoals(s, t, s')$
C4 $subgoals(s, t, s')$ implies $\vee_{i=1,m} sat\_rule(s, s', i)$
C5 $sat\_rule(s, s'', i)$ implies $\vee_{t: d(s,t) < d(s,s'')} subgoal(s, t)$
C6 $sat\_rule(s, s', i)$ implies $\vee_{t: d(s,t) \leq d(s,s')} subgoal(s, t)$
C7 $sat\_rule(s, s', i)$ iff (s, s') compatible with rule i
C8 collection of rules $i = 1, \ldots, m$ is terminating

In the constraints above, s is always an alive state, t is a tuple in the tuple graph rooted at s, s' is a state that results from an optimal plan for t from s, and s'' is a dead-end state. Checking the constraints involves constructing a tuple graph (Lipovetzky and Geffner 2012) and labeling dead-ends in the training instances in a preprocessing phase that is exponential in k. The interpretation of the constraints is direct. The first constraint says that a feature is selected if it is used in some rule. Constraints 2–4 say that every (non-goal) solvable state s "looks" at some subgoal t of width no greater than k from s, and that if s' may result from an optimal plan from s to t, then the transition (s, s') must satisfy one of the m rules. Constraint 5 says that if a transition from s to a dead-end state s'' satisfies a rule, then there must be a selected subgoal t of s that is closer from s than s'' ($d(s,t)$ and $d(s, s'')$ encode these distances, known after preprocessing). Constraint 6 says that if a pair of states (s, s') satisfies a rule, then s' is a state that results from an optimal plan from s to a subgoal t of s, or it is further from s than any such state. Constraint 7, not fully spelled out, captures the conditions under which a pair of states (s, s') satisfies sketch rule i given by the atoms $cond(i, f, v)$ and $eff(i, f, v)$. Last, Constraint 8, not fully spelled out either, demands that the sketch defined by these atoms is *structurally terminating* (Bonet and Geffner 2021). Encoding structural termination requires checking acyclicity in the policy graph that has size exponential in $|\mathcal{F}|$. The resulting theory is sound and complete in the following sense:

Theorem 6 (Soundness and Completeness) *A terminating sketch R with rules $C_i \mapsto E_i$, $i = 1, \ldots, m$, over features $F \in \mathcal{F}$, has sketch width $w_R(\mathcal{P}^*) \leq k$ over the closed class of problems $\mathcal{P}^*$ iff the theory $T_{k,m}(\mathcal{P}, \mathcal{F})$ is satisfiable and has a model where the rules that are true are exactly those in R.*

[4] For a numerical feature f, a condition with value $v = 0$ stands for $f = 0$ while an effect with the same value stands for $f\downarrow$. Similarly for $v = 1$.

We provide a proof for Theorem 6 in an extended version of the paper (Drexler, Seipp, and Geffner 2022). The notation $\mathcal{P}^*$ is used to denote the *closure* of the problems in $\mathcal{P}$; i.e. for any problem P in $\mathcal{P}$, $\mathcal{P}^*$ also includes the problems P' that are like P but with initial state s being a solvable state reachable from the initial state of P.

Learning Sketches: ASP Implementation

We implemented the theory $T_{k,m}(\mathcal{P}, \mathcal{F})$ expressed by constraints C1–C8 for learning sketches as an answer set program (Brewka, Eiter, and Truszczyński 2011; Lifschitz 2019) in Clingo (Gebser et al. 2012). Listing 1 shows the code. We include two approximations for scalability. First, we replace constraint C8 about termination by a suitable acyclicity condition (Gebser, Janhunen, and Rintanen 2014), that prevents a sequence of states $s_{i+1} \in G^k_R(s_i)$ from forming a cycle, where R is the learned sketch. Second, we omit constraint C6 that ensures that the width is bounded by k. The choice of the subgoals t still ensures that such subproblems will have an s-width bounded by k. The two approximations are not critical as even the exact theory $T_{k,m}(\mathcal{P}, \mathcal{F})$ does not guarantee acyclicity and sketch width bounded by k on the *test problems*. However, we will show in our experiments that the learned sketches are acyclic and have width bounded by k over all instances. The optimization criterion minimizes the sum of the number of sketch rules plus the sum of complexities of the selected features.

Experiments

We use the ASP implementation above to learn sketches for several tractable classical planning domains from the International Planning Competition (IPC). To learn a sketch, we use two Intel Xeon Gold 6130 CPUs, holding a total of 32 cores and 384 GiB of memory and set a time limit of seven days (wall-clock time). For evaluating the learned sketches, we limit time and memory by 30 minutes and 8 GiB. Our source code, benchmarks, and experimental data are available online (Drexler, Seipp, and Geffner 2022).

Data and Feature Generation. For each domain, we use a PDDL generator (Seipp, Torralba, and Hoffmann 2022) to generate a set of training instances small enough to be fully explored using breadth-first search. For domains with a randomized instance generator, we generate up to 200 instances for the same parameter configuration to capture sufficient variation. Afterwards, we remove unsolvable instances or instances with more than 10 000 states. We construct the feature pool $\mathcal{F}$ by automatically composing description logics constructors up to a feature complexity of 8, using the DLplan library (Drexler, Francès, and Seipp 2022) and the same bound as the one used by Francès, Bonet, and Geffner (2021). Compared to theirs, our grammar is slightly richer. However, since the "distance" features (Bonet, Francès, and Geffner 2019) significantly increase the size of the feature pool, we only include them for the domains where the ASP is unsolvable without them (indicated by an asterisk in the $|\mathcal{F}|$ column in Table 1). We provide details about the feature grammar in an extended version of the paper (Drexler, Seipp, and Geffner 2022). We set the maximum number of sketch rules m to 6.

Incremental Learning. Instead of feeding all instances to the ASP at the same time, we incrementally add only the smallest instance that the last learned sketch fails to solve. In detail, we order the training instances by the number of states in increasing order, breaking ties arbitrarily. The first sketch is the empty sketch $R^0 = \emptyset$. Then, in each iteration $i = 1, 2, \ldots$, we find the smallest instance I in the ordering which the sketch R^{i-1} obtained in the previous iteration results in a subproblem of width not bounded by k or in a cycle $s_1, \ldots, s_n = s_1$ of subgoal states $s_{i+1} \in G_R(s_i)$ with $i = 1, \ldots, n-1$. These tests can be performed because the training instances are small. If such an I exists, we use it as the only training instance if it is the largest among all previous training instances, otherwise we add it and proceed with the next iteration. If no such I exists, the sketch solves all training instances, and we return the sketch.

Learning Results. Table 1 shows results for the learning process. We learn sketches of width 0 in 6 out of the 9 domains, of width 1 in all 9 domains, and of width 2 in 8 out of 9 domains. In all cases, very few and very small training instances are sufficient for learning the sketches as indicated by the number of instances and the number of states that are actually used (columns $|P_i|$ and $|S|$ in Table 1). The choice of bound k has a strong influence because both the solution space and the number of subgoals (tuples) grows with k. Also, the number of rules decreases as the bound k increases. The most complex features overall have a complexity 7 (requiring the application of 7 grammar rules), the highest number of features in a sketch is 4, and the highest number of sketch rules is 5, showing that the learned sketches are very compact. We describe and analyze some of the learned sketches in the next section.

Search Results. Table 2 shows the results of SIW_R searches (Drexler, Seipp, and Geffner 2021) on large, unseen test instances using the learned sketches.[5] As a reference point, we include the results for the domain-independent planners LAMA (Richter and Westphal 2010) and Dual-BFWS (Lipovetzky and Geffner 2017). For Childsnack, Gripper, Miconic and Visitall, we use the Autoscale 21.11 instances for testing (Torralba, Seipp, and Sievers 2021). For the other domains, where no Autoscale instances are available, we generate 30 instances ourselves, with the number of objects varying between 10 and 100. Table 2 shows that the learned sketches of width $k = 1$ yield solutions to the 30 instances of each of the domains. Some of these domains, such as Childsnack, Spanner and Visitall, are not trivial for LAMA and Dual-BFWS, which only manage to solve 9, 0, and 29 instances, and 5, 0, and 25 instances, respectively. In all cases, the maximum effective width of the SIW_R search is bounded by the width parameter k, showing that the properties of the learned sketches generalize beyond the training set.

[5] Theorem 3 requires a $\text{SIW}_\text{R}(k)$ search rather than a SIW_R search but this a minor difference for theoretical reasons.

Listing 1: Full ASP code for learning sketches: constraints C1-C8 satisfied. Optimization in lines 24-25 for finding a simplest solutions measured by number of sketch rules plus sum of feature complexities.

```
1  % C1: construct rules and select features.
2  { select(F) } :- feature(F).           { rule(1..max_sketch_rules) }.
3  { c_eq(R, F); c_gt(R, F); c_unk(R, F) } = 1 :- rule(R), numerical(F).
4  { c_pos(R, F); c_neg(R, F); c_unk(R, F) } = 1 :- rule(R), boolean(F).
5  { e_dec(R, F); e_inc(R, F); e_unk(R, F); e_bot(R, F) } = 1 :- rule(R), numerical(F).
6  { e_pos(R, F); e_neg(R, F); e_unk(R, F); e_bot(R, F) } = 1 :- rule(R), boolean(F).
7  % C4 and C7: good and bad state pairs must comply with rules and selected features.
8  { good(R, I, S, S’) } :- rule(R), s_distance(I, S, S’, _).
9  c_satisfied(R, F, I, S) :- { c_eq(R, F) : V = 0; c_gt(R, F) : V > 0; c_pos(R, F) : V = 1;
       c_neg(R, F) : V = 0; c_unk(R, F) } = 1, rule(R), feature_valuation(F, I, S, V),
       s_distance(I, S, S’, _).
10 e_satisfied(R, F, I, S, S’) :- { e_dec(R, F) : V > V’; e_inc(R, F) : V < V’;
       e_pos(R, F) : V’ = 1; e_neg(R, F) : V’ = 0; e_bot(R, F) : V = V’; e_unk(R, F) } = 1,
       rule(R), feature_valuation(F, I, S, V), feature_valuation(F, I, S’, V’),
       s_distance(I, S, S’, _).
11 :- { not c_satisfied(R, F, I, S); not e_satisfied(R, F, I, S, S’) } != 0, select(F), good(
       R, I, S, S’).
12 :- { not c_satisfied(R, F, I, S) : select(F); not e_satisfied(R, F, I, S, S’) : select(F)
       } = 0, rule(R), s_distance(I, S, S’, _), not good(R, I, S, S’).
13 % C2: there must be one subgoal tuple for each state with unbounded width.
14 { subgoal(I, S, T) : tuple(I, S, T) } = 1 :- solvable(I, S), exceed(I, S).
15 % C3: states underlying subgoal tuples must be good.
16 :- { good(R, I, S, S’) : rule(R) } = 0, subgoal(I, S, T), contain(I, S, T, S’).
17 % C5: good pairs to dead-end states must be at larger distance than subgoal tuple.
18 :- D <= D’, s_distance(I, S, S’, D), t_distance(I, S, T, D’), subgoal(I, S, T),
       good(_, I, S, S’), solvable(I, S), unsolvable(I, S’).
19 % C8: ensure acyclicity.
20 order(I, S, S’) :- solvable(I, S), solvable(I, S’), good(_, I, S, S’), order(I, S’).
21 order(I, S) :- solvable(I, S), order(I, S, S’) : good(_, I, S, S’), solvable(I, S),
       solvable(I, S’).
22 :- solvable(I, S), not order(I, S).
23 % Optimization objective: smallest number of rules plus the sum of feature complexities.
24 #minimize { C,complexity(F, C) : complexity(F, C), select(F) }.
25 #minimize { 1,rule(R) : rule(R) }.
```

Plans. Comparing the plans found by SIW_R to the ones obtained with LAMA and Dual-BFWS, we see that the SIW_R plans have 1) half the length for Childsnack, Spanner, and Visitall, 2) roughly the same length for Delivery, Blocks-on, Miconic, and Reward, and 3) about three times the length for Blocks-clear and Gripper. In Blocks-clear, the width-0 sketch defines that unstacking from any tower (not just the one with the target block) is good until the target block is clear. In Gripper, transporting a single ball instead of two balls at a time adds two extra move actions per ball.

Sketch Analysis

In this section, we describe and analyze some of the learned sketches and prove that they all have width bounded by k and are acyclic. We provide proofs of these claims in an extended version of the paper (Drexler, Seipp, and Geffner 2022).

Gripper

In Gripper there are two rooms a and b. A robot can move between a and b, and pick up and drop balls. The goal is to move all balls from room a to b. We analyze the three learned sketches for the bounds $k = 0, 1, 2$. The learned sketch R^2_Φ for $k = 2$ has features $\Phi = \{g\}$, where g is the number of well-placed balls, and a single rule $r_1 = \{\} \mapsto \{g\uparrow\}$ saying that increasing the number of well-placed balls is good. These subproblems have indeed width bounded by 2. The learned sketch R^1_Φ for $k = 1$ has features $\Phi = \{g_a, g\}$, where g_a is the number of balls in room a and g is the number of balls that are either in room a or b. The rules are

$$r_1 = \{\} \mapsto \{g?, g_a\downarrow\}$$
$$r_2 = \{\} \mapsto \{g\uparrow\}$$

where r_1 says that picking up a ball in room a is good, and r_2 says that dropping a ball in room b is good. The learned sketch R^0_Φ for $k = 0$ has features $\Phi = \{B, c\}$, where B is true iff the robot is in room b and c is the number of carried balls, and rules

$$r_1 = \{c = 0\} \mapsto \{c?, \neg B\}$$
$$r_2 = \{B\} \mapsto \{B?, c\downarrow\}$$
$$r_3 = \{\neg B, c > 0\} \mapsto \{B?\}$$

where r_1 says that moving to room a or picking up a ball in room a is good when not carrying a ball, r_2 says that dropping a ball in room b is good, and r_3 says that moving to room b while carrying a ball is good.

Theorem 7 *The sketches R^k_Φ for Gripper are acyclic and have width k for $k = 0, 1, 2$.*

	$w=0$								$w=1$								$w=2$							
Domain	M	T	$\|P_i\|$	$\|S\|$	$\|\mathcal{F}\|$	C	$\|\Phi\|$	$\|R\|$	M	T	$\|P_i\|$	$\|S\|$	$\|\mathcal{F}\|$	C	$\|\Phi\|$	$\|R\|$	M	T	$\|P_i\|$	$\|S\|$	$\|\mathcal{F}\|$	C	$\|\Phi\|$	$\|R\|$
Blocks-clear	1	3	1	22	233	2	2	2	1	4	1	22	233	4	1	1	1	3	1	22	233	4	1	1
Blocks-on	26	14k	1	22	1011	7	3	3	9	105	1	22	1011	4	2	2	13	146	1	22	1011	4	1	1
Childsnack	–	–	–	–	–	–	–	–	122	228k	3	792	629	6	4	5	–	–	–	–	–	–	–	–
Delivery	–	–	–	–	–	–	–	–	17	521	1	96	474	4	2	2	3	18	1	20	287	4	1	1
Gripper	2	19	1	28	301	4	2	3	3	60	1	28	301	4	2	2	7	48	1	28	301	4	1	1
Miconic	–	–	–	–	–	–	–	–	1	5	1	32	119	2	2	2	2	6	1	32	119	2	1	1
Reward	3	46	2	26	916*	6	2	2	1	4	1	12	210	2	1	1	10	95	1	48	427	2	1	1
Spanner	12	2k	3	227	658	7	2	2	3	22	1	74	424	5	1	1	6	38	1	74	424	5	1	1
Visitall	3	54	2	36	722*	5	2	2	1	1	1	3	10	2	1	1	1	1	1	3	10	2	1	1

Table 1: Learning step. We show the peak memory in GiB after learning (M), the time in seconds for solving the ASP in parallel on 32 CPU cores (T), the number of training instances used in the encoding ($|P_i|$), the total number of states considered in the encoding ($|S|$), the number of Boolean and numerical features ($|\mathcal{F}|$) where * denotes that the distance feature was included, the largest complexity of a feature $f \in \Phi$ (C), the number of features ($|\Phi|$), and the number of sketch rules ($|R|$). We use "–" to indicate that the learning procedure failed because of insufficient resources.

	$w=0$				$w=1$				$w=2$				LAMA		BFWS	
Domain	S	T	AW	MW	S	T	AW	MW	S	T	AW	MW	S	T	S	T
Blocks-clear (30)	30	3	0.00	0	30	5	0.80	1	30	4	0.80	1	30	4	30	6
Blocks-on (30)	30	3	0.00	0	30	6	1.00	1	30	3	0.98	1	30	4	30	25
Childsnack (30)	–	–	–	–	30	1	0.10	1	–	–	–	–	9	2	5	658
Delivery (30)	–	–	–	–	30	1	1.00	1	30	4	1.66	2	30	1	30	1
Gripper (30)	30	4	0.00	0	30	3	0.50	1	30	656	2.00	2	30	1	30	6
Miconic (30)	–	–	–	–	30	5	0.53	1	30	132	2.00	2	30	7	30	25
Reward (30)	30	4	0.00	0	30	2	1.00	1	30	1	1.00	1	30	2	30	1
Spanner (30)	30	3	0.00	0	30	4	0.24	1	30	3	0.24	1	0	–	0	–
Visitall (30)	26	1360	0.00	0	30	20	0.00	1	30	21	0.00	1	29	213	25	833

Table 2: Testing step. We show the number of solved instances (S), the maximum time for solving an instance for which all algorithms find a solution, excluding algorithms that solve no instance at all (T), the average effective width (AW), and the maximum effective width (MW).

Blocks-on

The Blocks-on domain works like the standard Blocksworld domain, where a set of blocks can be stacked on top of each other or placed on the table. In contrast to the standard domain, Blocks-on tasks just require to place a single specific block on top of another block, a width-2 task. The learned sketch R_Φ for $k = 1$ has features $\Phi = \{H, g\}$, where H is true iff a block is being held and g is the number of blocks that are at their goal location. The sketch rules are

$$r_1 = \{\} \mapsto \{\neg H, g\uparrow\}$$
$$r_2 = \{H\} \mapsto \{\neg H, g?\}$$

where r_1 says that well-placing the block mentioned in the goal or unstacking towers on the table is good, and r_2 says that not holding a block is good. Starting in states s_0 where H is false, rule r_1 looks for subgoal states s_1 where H is false and g increased. Starting in states s'_0 where H is true, rule r_2 is also active which looks for subgoal states s'_1 where H is false and g has any value. From such states s'_1, however, rule r_1 takes over with $s_0 = s'_1$, and *no subgoal state* where H is true is reached again. If such states need to be traversed in the solution of the subproblems, they will not be encountered as subgoal states $s_{i+1} \in G_R(s_i)$.

Theorem 8 *The sketch R_Φ for Blocks-on is acyclic and has width* 1.

Childsnack

In Childsnack (Vallati, Chrpa, and McCluskey 2018), there is a kitchen, a set of tables, a set of plates, a set of children, all sitting at some table waiting to be served a sandwich. Some children are gluten allergic and hence they must be served a gluten-free sandwich that can be made with gluten-free bread and gluten-free content. The sandwiches are produced in the kitchen but can be moved to the tables using one of several plates. The learned sketch R_Φ for $k = 1$ has features $\Phi = \{sk, ua, gfs, s\}$, where sk is the number of *sandwiches* at the *kitchen*, ua is the number of *unserved* and *allergic* children, gfs is the number of *gluten-free sandwiches*, and s is the number of *served* children. The sketch rules are

$$r_1 = \{\} \mapsto \{sk?, ua?, gfs\uparrow, s?\}$$
$$r_2 = \{\} \mapsto \{sk\downarrow, ua?, gfs?, s?\}$$
$$r_3 = \{\} \mapsto \{sk?, ua\downarrow, gfs?, s?\}$$
$$r_4 = \{ua = 0\} \mapsto \{sk\uparrow, ua?, gfs?, s?\}$$
$$r_5 = \{ua = 0\} \mapsto \{sk?, ua?, gfs?, s\uparrow\}$$

which say that making a gluten free sandwich is good, moving a sandwich from the kitchen on a tray is good, serving a gluten-allergic child is good, making any sandwich is good if all gluten-allergic children have been served, serving any child is good if all gluten-allergic children have been served.

Theorem 9 *The sketch R_Φ for Childsnack is acyclic and has width* 1.

Miconic

In Miconic (Koehler and Schuster 2000), there are passengers, each waiting at some floor, who want to take an elevator to a target floor. The learned sketch R_Φ for $k = 1$ has features $\Phi = \{b, g\}$, where b is the number of boarded passengers and g is the number of served passengers. The sketch rules are

$$r_1 = \{\} \mapsto \{b?, g\uparrow\}$$
$$r_2 = \{\} \mapsto \{b\uparrow, g?\}$$

where r_1 says that moving a passenger to their target floor is good, and r_2 says that letting some passenger board is good.

Theorem 10 *Sketch R_Φ for Miconic is acyclic and has width* 1.

Related Work

Sketches provide a language for expressing control knowledge by hand or for learning it. Other languages have been developed for the first purpose, including Golog, LTL, and HTNs; we focus on the **learning problem**.

Features, General Policies, Heuristics. Sketches were introduced by Bonet and Geffner (2021) and used by hand by Drexler, Seipp, and Geffner (2021). The sketch language is the language of general policies (Bonet and Geffner 2018) that has been used for learning as well (Martín and Geffner 2004; Bonet, Francès, and Geffner 2019; Francès, Bonet, and Geffner 2021). The description logic features have also been used to learn linear value functions that can be used to solve problems greedily (Francès et al. 2019; de Graaff, Corrêa, and Pommerening 2021) and dead-end classifiers (Ståhlberg, Francès, and Seipp 2021). The use of numerical features that can be incremented and decremented qualitatively is inspired by QNPs (Srivastava et al. 2011; Bonet and Geffner 2020). Other works aimed at learning generalized policies or plans include planning programs (Segovia, Jiménez, and Jonsson 2016), logical programs (Silver et al. 2020), and deep learning approaches (Groshev et al. 2018; Bajpai, Garg, and Mausam 2018; Toyer et al. 2020), some of which have been used to learn heuristics (Shen, Trevizan, and Thiébaux 2020; Karia and Srivastava 2021).

HTNs. Hierarchical task networks explicitly decompose tasks into simpler tasks, and a number of methods for learning them have been studied (Zhuo, Muñoz-Avila, and Yang 2014; Hogg, Muñoz-Avila, and Kuter 2016). These methods, however, learn decompositions using slightly different inputs like annotated traces and decompositions. Jonsson (2009) developed a powerful approach for learning policies in terms of a hierarchy of macros, but the approach is restricted to domains with certain causal graphs.

Intrinsic Rewards and (Hierarchical) RL. Intrinsic rewards have been introduced for improving exploration in reinforcement learning (Singh et al. 2010), and several authors have addressed the problem of learning intrinsic rewards over families of problems. Interestingly, the title of one of the papers is a question "What can learned intrinsic rewards capture?" (Zheng et al. 2020). The answer to this question in our setting is clean and simple: intrinsic rewards are supposed to capture common subgoal structure. Lacking a language to talk about families of problems and a language to talk about subgoal structure, however, the answer that the authors provide is less crisp: learned intrinsic rewards are supposed to speed up (deep) RL. The problem of subgoal structure also surfaces in *hierarchical RL* that aims at learning and exploiting hierarchical structure in RL (Barto and Mahadevan 2003; Kulkarni et al. 2016).

Conclusions

We have developed a formulation for learning sketches automatically given instances of the target class of problems $\mathcal{Q}$ and a bound k on the sketch width. The work builds on prior works that introduced the ideas of general policies, sketches, problem width, and description logic features. The learning formulation guarantees a bounded width on the training instances but the experiments show that this and other properties generalize to entire families of problems $\mathcal{Q}$, some of which are challenging for current domain-independent planners. The properties ensure that all problems in $\mathcal{Q}$ can be solved in polynomial time by a variant of the SIW algorithm. This is possibly *the first general method for learning how to decompose planning problems into subproblems with a polynomial complexity that is controlled with a parameter*. Three limitations of the proposed learning approach are 1) it cannot be applied to intractable domains, 2) it yields large (grounded) ASP programs that are often difficult to solve, and 3) it deals with collections of problems encoded in PDDL-like languages, even if the problem of learning subgoal structure arises in other settings such as RL. Three goals for the future are to improve scalability, to deal with non-PDDL domains, taking advantage of recent approaches that learn such domains from data, and to use sketches for learning hierarchical policies.

Acknowledgements

This work was partially supported by an ERC Advanced Grant (grant agreement no. 885107), by project TAILOR, funded by EU Horizon 2020 (grant agreement no. 952215), and by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. Hector Geffner is a Wallenberg Guest Professor at Linköping University, Sweden. The computations were enabled by resources provided by the Swedish National Infrastructure for Computing (SNIC), partially funded by the Swedish Research Council through grant agreement no. 2018-05973.

References

Bajpai, A.; Garg, S.; and Mausam. 2018. Transfer of Deep Reactive Policies for MDP Planning. In *Proc. NeurIPS 2018*.

Barto, A. G.; and Mahadevan, S. 2003. Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Dynamic Systems*, 13: 41–77.

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

Bonet, B.; Francès, G.; and Geffner, H. 2019. Learning Features and Abstract Actions for Computing Generalized Plans. In *Proc. AAAI 2019*, 2703–2710.

Bonet, B.; and Geffner, H. 2018. Features, Projections, and Representation Change for Generalized Planning. In *Proc. IJCAI 2018*, 4667–4673.

Bonet, B.; and Geffner, H. 2020. Qualitative Numeric Planning: Reductions and Complexity. *JAIR*, 69: 923–961.

Bonet, B.; and Geffner, H. 2021. General Policies, Representations, and Planning Width. In *Proc. AAAI 2021*, 11764–11773.

Brewka, G.; Eiter, T.; and Truszczyński, M. 2011. Answer set programming at a glance. *Communications of the ACM*, 54(12): 92–103.

de Graaff, R.; Corrêa, A. B.; and Pommerening, F. 2021. Concept Languages as Expert Input for Generalized Planning: Preliminary Results. In *ICAPS 2021 Workshop on Knowledge Engineering for Planning and Scheduling*.

Drexler, D.; Francès, G.; and Seipp, J. 2022. Description Logics State Features for Planning (DLPlan). https://doi.org/10.5281/zenodo.5826139.

Drexler, D.; Seipp, J.; and Geffner, H. 2021. Expressing and Exploiting the Common Subgoal Structure of Classical Planning Domains Using Sketches. In *Proc. KR 2021*, 258–268.

Drexler, D.; Seipp, J.; and Geffner, H. 2022. Proofs, Code, and Data for the ICAPS 2022 Paper "Learning Sketches for Decomposing Planning Problems into Subproblems of Bounded Width". https://doi.org/10.5281/zenodo.6381592.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1994. HTN Planning: Complexity and Expressivity. In *Proc. AAAI 1994*, 1123–1128.

Francès, G.; Bonet, B.; and Geffner, H. 2021. Learning General Planning Policies from Small Examples Without Supervision. In *Proc. AAAI 2021*, 11801–11808.

Francès, G.; Corrêa, A. B.; Geissmann, C.; and Pommerening, F. 2019. Generalized Potential Heuristics for Classical Planning. In *Proc. IJCAI 2019*, 5554–5561.

Gebser, M.; Janhunen, T.; and Rintanen, J. 2014. ASP Encodings of Acyclicity Properties. In *Proc. KR 2014*, 634–637.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2012. *Answer Set Solving in Practice*. Morgan & Claypool Publishers.

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

Georgievski, I.; and Aiello, M. 2015. HTN Planning. *AIJ*, 222: 124–156.

Groshev, E.; Goldstein, M.; Tamar, A.; Srivastava, S.; and Abbeel, P. 2018. Learning Generalized Reactive Policies Using Deep Neural Networks. In *Proc. ICAPS 2018*, 408–416.

Hogg, C.; Muñoz-Avila, H.; and Kuter, U. 2016. Learning Hierarchical Task Models from Input Traces. *Computational Intelligence*, 32(1): 3–48.

Jonsson, A. 2009. The role of macros in tractable planning. *JAIR*, 36: 471–511.

Karia, R.; and Srivastava, S. 2021. Learning Generalized Relational Heuristic Networks for Model-Agnostic Planning. In *Proc. AAAI 2021*, 8064–8073.

Koehler, J.; and Schuster, K. 2000. Elevator Control as a Planning Problem. In *Proc. AIPS 2000*, 331–338.

Kulkarni, T. D.; Narasimhan, K. R.; Saeedi, A.; and Tenenbaum, J. 2016. Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation. In *Proc. NIPS 2016*, 3675–3683.

Lifschitz, V. 2019. *Answer Set Programming*. Springer.

Lipovetzky, N. 2021. Width-Based Algorithms for Common Problems in Control, Planning and Reinforcement Learning. In *Proc. IJCAI 2021*, 4956–4960.

Lipovetzky, N.; and Geffner, H. 2012. Width and Serialization of Classical Planning Problems. In *Proc. ECAI 2012*, 540–545.

Lipovetzky, N.; and Geffner, H. 2017. Best-First Width Search: Exploration and Exploitation in Classical Planning. In *Proc. AAAI 2017*, 3590–3596.

Martín, M.; and Geffner, H. 2004. Learning Generalized Policies from Planning Examples Using Concept Languages. *Applied Intelligence*, 20(1): 9–19.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *JAIR*, 39: 127–177.

Segovia, J.; Jiménez, S.; and Jonsson, A. 2016. Generalized Planning with Procedural Domain Control Knowledge. In *Proc. ICAPS 2016*, 285–293.

Seipp, J.; Torralba, Á.; and Hoffmann, J. 2022. PDDL Generators. https://doi.org/10.5281/zenodo.6382173.

Shen, W.; Trevizan, F.; and Thiébaux, S. 2020. Learning Domain-Independent Planning Heuristics with Hypergraph Networks. In *Proc. ICAPS 2020*, 574–584.

Silver, T.; Allen, K. R.; Lew, A. K.; Kaelbling, L.; and Tenenbaum, J. 2020. Few-Shot Bayesian Imitation Learning with Logical Program Policies. In *Proc. AAAI 2020*, 10251–10258.

Singh, S. P.; Lewis, R. L.; Barto, A. G.; and Sorg, J. 2010. Intrinsically motivated reinforcement learning: An evolutionary perspective. *IEEE Transactions on Autonomous Mental Development*, 2: 70–82.

Srivastava, S.; Zilberstein, S.; Immerman, N.; and Geffner, H. 2011. Qualitative Numeric Planning. In *Proc. AAAI 2011*, 1010–1016.

Ståhlberg, S.; Francès, G.; and Seipp, J. 2021. Learning Generalized Unsolvability Heuristics for Classical Planning. In *Proc. IJCAI 2021*, 4175–4181.

Torralba, Á.; Seipp, J.; and Sievers, S. 2021. Automatic Instance Generation for Classical Planning. In *Proc. ICAPS 2021*, 376–384.

Toyer, S.; Thiébaux, S.; Trevizan, F.; and Xie, L. 2020. ASNets: Deep Learning for Generalised Planning. *JAIR*, 68: 1–68.

Vallati, M.; Chrpa, L.; and McCluskey, T. L. 2018. What you always wanted to know about the deterministic part of the International Planning Competition (IPC) 2014 (but were too afraid to ask). *Knowledge Engineering Review*, 33.

Zheng, Z.; Oh, J.; Hessel, M.; Xu, Z.; Kroiss, M.; van Hasselt, H.; Silver, D.; and Singh, S. 2020. What Can Learned Intrinsic Rewards Capture? In *Proc. ICML 2020*, 11436–11446.

Zhuo, H. H.; Muñoz-Avila, H.; and Yang, Q. 2014. Learning hierarchical task network domains from partially observed plan traces. *AIJ*, 212: 134–157.