Journal of Artificial Intelligence Research 80 (2024) 171–208 Submitted 12/2023; published 05/2024

Expressing and Exploiting Subgoal Structure in Classical Planning Using Sketches

Dominik Drexler DOMINIK.DREXLER@LIU.SE
Jendrik Seipp JENDRIK.SEIPP@LIU.SE
Linköping University, Sweden

Hector Geffner HECTOR.GEFFNER@ML.RWTH-AACHEN.DE
RWTH Aachen University, Germany
Linköping University, Sweden

Abstract

Width-based planning methods deal with conjunctive goals by decomposing problems into subproblems of low width. Algorithms like SIW thus fail when the goal is not easily serializable in this way or when some of the subproblems have a high width. In this work, we address these limitations by using a simple but powerful language for expressing finer problem decompositions introduced recently by Bonet and Geffner, called *policy sketches*. A policy sketch R over a set of Boolean and numerical features is a set of sketch rules $C \mapsto E$ that express how the values of these features are supposed to change. Like general policies, policy sketches are domain general, but unlike policies, the changes captured by sketch rules do not need to be achieved in a single step. We show that many planning domains that cannot be solved by SIW are provably solvable in low polynomial time with the $\mathrm{SIW}_{\mathrm{R}}$ algorithm, the version of SIW that employs user-provided policy sketches. Policy sketches are thus shown to be a powerful language for expressing domain-specific knowledge in a simple and compact way and a convenient alternative to languages such as HTNs or temporal logics. Furthermore, they make it easy to express general problem decompositions and prove key properties of them like their width and complexity.

1. Introduction

The success of width-based methods in classical planning is the result of two main ideas: the use of conjunctive goals for decomposing a problem into subproblems, and the observation that the width of the subproblems is often bounded and small (Lipovetzky & Geffner, 2012). When these assumptions do not hold, pure width-based methods struggle and need to be extended with heuristic estimators or landmark counters that yield finer problem decompositions (Lipovetzky & Geffner, 2017a, 2017b). These hybrid approaches have resulted in state-of-the-art planners but run into shortcomings of their own: unlike pure width-based search methods, they require declarative, PDDL-like action models and thus cannot plan with black box simulators (Lipovetzky, Ramirez, & Geffner, 2015; Shleyfman, Tuisov, & Domshlak, 2016; Geffner & Geffner, 2015), and they produce decompositions that are ad-hoc and difficult to understand. Variations of these approaches, where the use of declarative action models is replaced by polynomial searches, have pushed the scope of pure-width based search methods (Francès, Ramírez, Lipovetzky, & Geffner, 2017), but they do not fully overcome their basic limits: *top goals that are not easily serializable or that have a high width.* These are indeed the limitations of one of the simplest width-based search methods, Serialized Iterated Width (SIW) that greedily achieves top goal first, one at a time, using IW searches (Lipovetzky & Geffner, 2012).

In this work, we address the limitations of the SIW algorithm differently by using a simple but powerful language for expressing richer problem decompositions recently introduced by Bonet and Geffner (2021), called **policy sketches**. A policy sketch is a set of sketch rules over a set of Boolean and numerical features of the form $C \mapsto E$ that express how the values of the features are supposed to change. Like **general policies** (Bonet & Geffner, 2018), sketches are general and not tailored to specific instances of a domain, but unlike policies, the feature changes expressed by sketch rules represent subgoals that do not need to be achieved in a single step.

We pick up here where Bonet and Geffner left off and show that many benchmark planning domains that SIW cannot solve are provably solvable in low polynomial time with the SIW_R algorithm, the version of SIW that makes use of user-provided policy sketches. Policy sketches are thus shown to be a powerful **language for expressing domain-specific knowledge** in a simple and compact way and a convenient alternative to languages such as HTNs or temporal logics. Bonet and Geffner introduce the language of sketches and the theory behind them; we show their use and the properties that follow from them. As we will see, unlike HTNs and temporal logics, sketches can be used to **express and exploit the common subgoal structure of planning domains** without having to express how subgoals are to be achieved. Also, by being simple and succinct, they provide a convenient target language for learning the subgoal structure of domains automatically, although this problem, related to the problem of learning general policies (Bonet, Francès, & Geffner, 2019; Francès, Bonet, & Geffner, 2021), is outside the scope of this article. In this work, we use sketches to solve domains in polynomial time, which excludes intractable domains. Indeed, intractable domains do not have general policies nor sketches of bounded width and require non-polynomial searches. Sketches and general policies, however, are closely related: sketches provide the skeleton of a general policy, or a general policy with "holes" that are filled by searches that can be shown to be polynomial (Bonet & Geffner, 2021).

This article is an extended version of a conference paper published at the International Conference on Knowledge Representation and Reasoning (Drexler, Seipp, & Geffner, 2021) with the following main extensions: first, we prove properties of the policy sketches for all seven benchmark domains that we consider (the conference paper only contains two proofs). We also correct the previously reported upper bounds on the sketch width for the Grid and TPP domains. Second, we provide detailed information about the description logic grammar and the state features that we use in the policy sketches. Third, in our experiments, we evaluate the new SIW_R algorithm in both a grounded and a lifted planner and compare both implementations to baseline planners on an additional set of harder benchmarks. Fourth, we release the code for constructing and evaluating description logic features in the form of an open-source C++ library (with Python bindings), called *DLPlan*. Finally, we expand the related work section to cover general policies, reward machines, landmarks and polynomial algorithms for domain-independent planning.

Since the publication of the original conference paper, there have been several works by us and others building on the ideas presented here. In particular, we have shown that it is possible to learn sketches automatically (Drexler, Seipp, & Geffner, 2022) and that we can use them as building blocks for learning hierarchical policies (Drexler, Seipp, & Geffner, 2023b). Daggelinckx (2023) used temporal logic to exhaustively generate *all* well-formed sketches up to a given size and Grundke (2022) compared policy sketches to other forms of domain-specific knowledge. Finally, Dalmau-Moreno, García, Gómez, and Geffner (2023) used policy sketches for combined task and motion planning. To keep the size of this article manageable, we focus exclusively on handcrafted sketches and their analysis here, but discuss related work in Section 7.

The article is organized as follows. We review the notions of width (Section 2), policy sketches (Section 3), and sketch width (Section 4), following Bonet and Geffner (2021). In Section 5, we show that it is possible to write compact and transparent policy sketches for many domains and establish their widths. We analyze the performance of the SIW_R algorithm using these sketches in Section 6. Finally, we compare sketches to HTNs and temporal logics and briefly discuss the challenge of learning sketches automatically (Section 7), before summarizing the main contributions in Section 8.

2. Planning and Width

A *classical planning problem* or instance $P = (D, I)$ is assumed to be given by a first-order domain D with action schemas defined over some domain predicates, and instance information I describing a set of objects, and two sets of ground literals describing the initial situation $Init$ and goal description $Goal$. The initial situation is assumed to be complete such that either L or its complement is in $Init$. A problem 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 includes all of those that make the goal atoms in $Goal$ true, the actions Act are the ground actions obtained from the action schemas and the objects, the actions $A(s)$ applicable in state s are 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 $s_{n+1} \in G$. The cost of a plan is assumed to be given by its length, and a plan is optimal if there is no shorter plan. We are interested in solving *collections* of well-formed instances $P = (D, I)$ over fixed domains D denoted as $\mathcal{Q}_D$ or simply as $\mathcal{Q}$.

The most basic width-based search method for solving a planning problem P is IW(1). It performs 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 r 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 & Geffner, 2012):

Definition 1 (Width)**.** *Let P be a classical planning problem with initial state s_0 and goal states G. The **width** $w(P)$ of P is the minimum k for which there exists a **sequence** $t_0, t_1, \ldots, t_m$ **of atom tuples** t_i, each consisting of at most k atoms, such that:*

1. *t_0 is true in initial state s_0 of P,*
2. *all optimal plans for t_i can be extended into an optimal plan for t_{i+1} by adding a single action, $i = 1, \ldots, m-1$,*
3. *if π is an optimal plan for t_m, then 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 & Geffner, 2021). Chains of tuples $\theta = (t_0, t_1, \ldots, t_m)$ that comply with conditions 1–3 are called **admissible**, and the size of θ is the size $|t_i|$ of the largest tuple in the chain. We talk about the third condition by saying that t_m implies G in the admissible chain $t_1, t_2, \ldots, t_m$. The width $w(P)$ is thus k if k is the minimum size of an admissible chain for

P. If the width of a problem P is $w(P) = k$, IW(k) finds an optimal (shortest) plan for P in time and space that are exponential in k and not in the number of problem variables N as breadth-first search.

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 bounds the branching factor in problem P. IW(k) is guaranteed to solve P optimally (shortest path) 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.

While IW and IW(k) algorithms are not practical by themselves, they are building blocks for other methods. Serialized Iterated Width or **SIW** (Lipovetzky & Geffner, 2012), starts at the initial state $s = s_0$ of P, and then performs an IW search from s to find a shortest path to state s' such that $\#g(s') < \#g(s)$ where $\#g$ counts the number of top goals of P that are not true in s. If s' is not a goal state, s is set to s' and the loop repeats until a goal state is reached.

In practice, the IW(k) searches in SIW are limited to $k \leq 2$ or $k \leq 3$, so that SIW solves a problem or fails in low polynomial time. SIW performs well in many benchmark domains but fails in problems where the width of some top goal is not small, or the top goals cannot be serialized greedily. More recent methods address these limitations by using width-based notions (novelty measures) in complete best-first search algorithms (Lipovetzky & Geffner, 2017a; Francès et al., 2017), yet they also struggle in problems where some top goals have high width. In this work, we take a different route: we keep the greedy polynomial searches underlying SIW but consider a richer class of problem decompositions expressed through sketches. The resulting planner SIW_R is **not** domain-independent like SIW, but it illustrates that a bit of domain knowledge can go a long way in the effective solution of arbitrary domain instances.

3. Features and 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 Φ of features and a state s of some instance P in $\mathcal{Q}$, $f(s)$ is the **feature valuation** determined by a state s. A Boolean feature valuation over Φ refers instead to the valuation of the expressions p and $n = 0$ for Boolean and numerical features p and n in Φ. If f is a feature valuation, $b(f)$ will denote the Boolean feature valuation determined by f where the values of numerical features are just compared with 0.

The set of features Φ **distinguish** or **separate the goals** in $\mathcal{Q}$ if there is a set $B_\mathcal{Q}$ of Boolean feature valuations such that s is a goal state of an instance $P \in \mathcal{Q}$ iff $b(f(s)) \in B_\mathcal{Q}$. For example, if $\mathcal{Q}_{clear}$ is the set of all blocks world instances with stack/unstack operators and common goal $clear(x) \land handempty$ for some block x, and $\Phi = \{n(x), H\}$ are the features that track the number of blocks above x and whether the gripper is holding a block, then there is a single Boolean goal valuation that makes the expression $n(x) = 0$ true and H false.

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 & 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 **policy sketch** R_Φ is a collection of sketch rules over the features Φ and the sketch is well-formed if it is built from features that **distinguish the goals** in $\mathcal{Q}$, and is **terminating** (to be made precise below). A **well-formed sketch** for a class of problems $\mathcal{Q}$ defines a serialization over $\mathcal{Q}$; namely, a "preference" ordering '$\prec$' over the feature valuations that is irreflexive and transitive, and which is given by the smallest strict partial order that satisfies $f' \prec f$ if f' is not a goal valuation and the pair of feature valuations (f, f') **satisfies a sketch rule** $C \mapsto E$. This happens when: 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 than in f, i.e., $f'_n < f_n$ (resp. $f_n > f'_n$), and 4) Features that do not occur in E have the same value in f and f'. Effects $p?$ and $n?$ do not constrain the value of the features p and n in any way, and by including them in E, we say that they can change in any way, as opposed to features that are not mentioned in E whose values in f and f' must be the same. Following Bonet and Geffner, we do not use the serializations determined by sketches but their associated problem **decompositions**.

The set of **subgoal states** $G_r(s)$ associated with a sketch rule $r : C \mapsto E$ in R_Φ and a state s for a problem $P \in Q$, is the set of states s' that are either goal states of P or those with feature valuations $f(s')$ such that $(f(s), f(s'))$ satisfies the sketch rule r. Intuitively, when in a state s, the subgoal states s' in $G_r(s)$ provide a stepping stone in the search for plans connecting s to the goal of P. Furthermore $G^*_r(s)$ denotes the set of subgoal states in $G_r(s)$ that are closest to s, $G_R(s)$ denotes the set of subgoal states associated with a sketch R, i.e., $G_R(s) = \bigcup_{r \in R} G_r(s)$, and $G^*_R(s)$ denotes the set of subgoal states in $G_R(s)$ that are closest to s.

4. Serialized Iterated Width with Sketches

The **SIW$_\mathbf{R}$** algorithm is a variant of the SIW algorithm that uses a given sketch $R = R_\Phi$ for solving problems $P \in \mathcal{Q}$. SIW$_\mathrm{R}$ starts at the state $s := s_0$, where s_0 is the initial state of P, and then performs an IW search to find a state s' that is closest from s such that s' is a goal state of P or a subgoal state in $G_r(s)$ for some sketch rule r in R. If s' is not a goal state, then s is set to s', $s := s'$, and the loop repeats until a goal state is reached. The features define subgoal states through the sketch rules but otherwise play no role in the IW searches.

The only difference between SIW and SIW$_\mathrm{R}$ is that in SIW each IW search finishes when the goal counter $\#g$ is decremented, while in SIW$_\mathrm{R}$, when a goal or subgoal state is reached. The behavior of plain SIW can be emulated in SIW$_\mathrm{R}$ using the single sketch rule $\{\#g > 0\} \mapsto \{\#g\downarrow\}$ in R when the goal counter $\#g$ is the only feature, and the rule $\{\#g > 0\} \mapsto \{\#g\downarrow, p?, n?\}$, when p and n are the other features. This last rule says that it is always "good" to decrease the goal counter independently of the effects on other features, or alternatively, that decreasing the goal counter is a subgoal from any state s where $\#g(s)$ is positive.

The complexity of SIW$_\mathrm{R}$ over a class of problems $\mathcal{Q}$ can be bounded in terms of the **width of the sketch** R_Φ, which is given by the width of the possible subproblems that can be encountered during the execution of SIW$_\mathrm{R}$ when solving a problem P in $\mathcal{Q}$. For this, let us define the set $S_R(P)$ of reachable states in P when following the sketch $R = R_\Phi$ recursively as follows: 1) the initial state s of P is in $S_R(P)$, 2) the subgoal states $s' \in G^*_R(s)$ are in $S_R(P)$ if $s \in S_R(P)$. The states in $S_R(P)$ are called the R-reachable states in P. The width of sketch R is then defined as follows (Bonet & Geffner, 2021).

Definition 2 (Sketch width)**.** *Let $R = R_\Phi$ be a well-formed sketch for a class of problems $\mathcal{Q}$. The width of the sketch R at state s of problem $P \in \mathcal{Q}$, denoted $w_R(P[s, G^*_R(s)])$, is the width k of the subproblem $P[s, G^*_R(s)]$ that is like P but with initial state s and goal states $G^*_R(s)$. The* ***width of the sketch*** *R over $\mathcal{Q}$ is $w_R(\mathcal{Q}) = \max_{P,s} w_R(P[s, G^*_R(s)])$ for $P \in \mathcal{Q}$ and $s \in S_R(P)$.*[1]

The time complexity of SIW_R can then be expressed as follows, under the assumption that the features are all linear (Bonet & Geffner, 2021).

Theorem 3. *If the width $w_R(\mathcal{Q}) = k$ of a well-formed sketch $R = R_\Phi$ then SIW_R solves any $P \in \mathcal{Q}$ in $O(bN^{|\Phi|+2k-1})$ time and $O(bN^k + N^{|\Phi|+k})$ space.*

A feature is linear if it can be computed in linear time and can take a linear number of values at most. In both cases, the linearity is in the number of atoms N in the problem P in $\mathcal{Q}$. If the features are not linear but polynomial in P, the bounds on SIW_R remain polynomial as well (both k and Φ are constants).

Bonet and Geffner introduce and study the language of sketches as a variation of the language of general policies and their relation to the width and serialized width of planning domains. They illustrate the use of sketches in a simple example (Delivery) but focus mainly on the theoretical aspects. Here we focus instead on their use for modeling domain-specific knowledge in the standard planning benchmarks as an alternative to languages like HTNs.

5. Sketches for Classical Planning Domains

In this section, we present policy sketches for seven classical planning domains from the International Planning Competition (IPC). All of the chosen domains are solvable suboptimally in polynomial time, but plain SIW fails to solve them. There are two main reasons why SIW fails. First, SIW fails if achieving a single goal atom has too large width, and second, SIW fails if greedy goal serialization generates such avoidable subproblems, including reaching unsolvable states.

We provide a handcrafted sketch for each of the domains and show that it is well-formed and has small sketch width. These sketches allow SIW_R to solve all instances of the domain in low polynomial time and space by Theorem 3. Furthermore, we impose a low polynomial complexity bound on the runtime of the evaluation of each feature, i.e., at most cubic in the number of grounded atoms. Limiting feature complexity is required since without any limit, one could use a numerical feature that encodes the optimal value function $V^*(s)$, i.e., the perfect goal distance of all states s. Using this feature in combination with the sketch rule $\{V^* > 0\} \mapsto \{V^*\!\downarrow\}$ would make all problems trivially solvable. Even with linear and quadratic features, we can capture complex state properties such as distances between objects.

Aside from limiting their computational complexity, we impose no further assumptions on the features. However, we found that all features required for the sketches below can be defined with a description logic grammar (Baader, Calvanese, McGuinness, Nardi, & Patel-Schneider, 2003; Francès et al., 2021) over primitive PDDL predicates. Using a common representation for the features simplifies evaluating them for a given state and makes it easy to check that they can indeed be evaluated in low polynomial time. We present the description logic grammar and the feature

1. This definition changes the one by Bonet and Geffner slightly by restricting the reachable states s to those that are R-reachable; i.e., part of $S_R(P)$. This distinction is convenient when $\mathcal{Q}$ does not contain all possible "legal" instances P but only those in which the initial situations complies with certain conditions (e.g., robot arm is empty). In those cases, the sketches for $\mathcal{Q}$ do not have to cover all reachable states.

representations in the appendix. For the presentation and theoretical analysis of the sketches the feature representation is irrelevant, so we do not discuss it further in the main part of the article.

5.1 Proving Termination and Sketch Width

For each sketch introduced below we show that it uses goal-separating features, is terminating and has bounded and small sketch width. Showing that the features are goal separating is usually direct.

Proving **termination** is required to ensure that by iteratively moving from a state s to a subgoal state $s' \in G^*_R(s)$ we cannot get trapped in a cycle. The conditions under which a sketch R_Φ is terminating are similar to those that ensure that a general policy π_Φ is terminating (Srivastava, Zilberstein, Immerman, & Geffner, 2011; Bonet & Geffner, 2020b, 2021), and can be determined in polynomial time in the size of the policy graph $G(R_\Phi)$ using the SIEVE procedure where R_Φ is interpreted as a general policy (Srivastava et al., 2011; Bonet & Geffner, 2020b).

The **policy graph** $G(R_\Phi)$, or simply $G(R)$, for sketch R over features Φ has a node for each of the $2^{|\Phi|}$ Boolean feature valuations over Φ, and edges $b \to b'$ labeled with E if b is not a goal valuation and (b, b') is compatible with a rule $C \mapsto E$ in the sketch.

The SIEVE procedure was originally introduced for numerical features only. In the following, we extend it to also work for Boolean features. We say that a rule $C \mapsto E$ *changes* a Boolean feature p if $p \in C$ and $\neg p \in E$ or vice versa. SIEVE then iteratively checks whether the feature changes on the edges $b \to b'$ in a strongly connected component (SCC) of $G(R)$ do not allow for infinite execution, i.e., at least some numerical feature n decreases ($n\downarrow$) in the SCC and never increases ($n\uparrow$), or at least one Boolean feature p changes from true to false (resp. false to true) in the SCC. The procedure then iteratively eliminates such edges, effectively breaking SCCs into smaller SCCs. If the largest SCC has size one, then the sketch R over features Φ is terminating; otherwise, the sketch is not terminating.

Often, however, a simple syntactic procedure suffices that eliminates sketch rules, one after the other, until none is left. This syntactic procedure is sound but not complete in general. The procedure iteratively applies one of the following cases until no rule is left (the sketch terminates) or until no further cases apply (there may be an infinite loop in the sketch):

1. Eliminate a rule r if it decreases a numerical feature n ($n\downarrow$) that no other remaining rule can increase ($n\uparrow$ or $n?$), and mark n to indicate that n changes finitely often,

2. Eliminate a rule r if it changes the value of a Boolean feature p that no other remaining rule changes in the opposite direction, and mark p to indicate that p changes finitely often,

3. Eliminate a rule $r = C \mapsto E$ that decreases a numerical feature n or that changes a Boolean feature p to true (false) such that for all other remaining rules $r' = C' \mapsto E'$ it holds that if E' changes the feature in the opposite direction, i.e., $n\uparrow$, $n?$ or changes p to false (true), there must be a condition on a numerical feature m or Boolean feature q in C that is marked and is complementary to the one in C', e.g., $m > 0 \in C$ and $m = 0 \in C'$ or $q \in C$ and $\neg q \in C'$.

Theorem 4. *The incomplete termination test procedure is sound in the sense that if it returns that a sketch R is terminating then R is indeed terminating.*

Proof. We must show that if the incomplete procedure removes a rule $r \in R$ then the SIEVE algorithm removes all edges with label r in the policy graph $G(R)$. The SIEVE algorithm iteratively removes edges from the strongly connected components (SCCs) of the policy graph $G(R)$. It removes an edge with label r from an SCC if the rule r decreases a numerical feature n, i.e., $n\downarrow$, that is never incremented in the SCC, i.e., $n?$ or $n\uparrow$. Originally, SIEVE does not consider Boolean features in the elimination. We allow SIEVE to remove an edge with label r from an SCC if the rule r changes a Boolean feature p that is never changed in the opposite direction in the SCC. Since r changes a Boolean feature in the SCC that no other rule changes in the opposite direction then the rule r cannot be part of a cycle. Next, consider the cases 1–3 from above:

1. The eliminated rule r decreases n and no other rule r' increases n. Hence, in all SCCs of $G(R)$, the rule r decreases n and it is never increased in the SCC. Thus, SIEVE removes all edges labeled r.

2. The eliminated rule r changes p which no other rule r' changes in the opposite direction. Hence, in all SCCs of $G(R)$, the rule r changes p which is never changed in the opposite direction in the SCC. Thus, SIEVE removes all edges labeled r.

3. Consider marked numerical feature n and marked Boolean feature p. Then there does not exist an SCC that contains Boolean feature valuations b, b' where $n = 0 \in b, n > 0 \in b'$, or $p \in b, \neg p \in b'$, or $\neg p \in b, p \in b'$ because those change only finitely many times. The eliminated rule $r = C \mapsto E$ has condition either $n = 0, n > 0, \neg p$ or p and has only edges in one SCC C_1 that decrease a numerical feature m or change a Boolean feature q. The rules r' that increase m or change q in the opposite direction have complementary conditions and therefore, have edges in other SCCs C_2. Thus, SIEVE removes all edges with label r because r decreases m and no other rule r' increases m in C_1 or r changes p and no other rule r' changes p in the opposite direction in C_1. □

Showing that a sketch for problem class $\mathcal{Q}$ has **sketch width** k requires to prove that for all R-reachable states s in all problem instances $P \in \mathcal{Q}$, the width of $P[s]$ is bounded by k. Remember that $P[s]$ is like P but with initial state s, and goal states G of P combined with goal states $G_r(s)$ of all $r \in R$. We need to show that the first subproblem $P_i = P[s]$ with initial state s of P and $i = 0$ has width at most k. Then for any closest subgoal state s' of P_i, we need to show that the next subproblem $P_{i+1} = P[s']$ has width at most k until all closest subgoal states are goals of P. We prove the width of each subproblem P_i by providing an admissible chain $t_1, \ldots, t_m$ of size at most k where all optimal plans for t_m are also optimal plans for P_i. We overapproximate the set of R-reachable states where necessary to make the proofs more compact. This implies that our results provide an upper bound on the actual sketch width, which is small and tight. Furthermore, regardless of which rule r defines the closest subgoal $G_r(s)$ for an R-reachable state s, we show that $P[s]$ with subgoal $G_r(s)$ in the R-reachable state s has width k. This suffices because the rule that defines the closest subgoals satisfies the optimality in the third part of the definition of width.

5.2 Floortile

In the Floortile domain (Linares López, Celorrio, & Olaya, 2015), a set of robots have to paint a set of tiles in a grid. As is done in the IPC tasks, we consider a simpler version of the domain where the robots have to paint a rectangular portion of a rectangular grid. There can be at most one robot a on each tile t at any time and the predicate $\textit{robot-at}(a, t)$ is true iff a is on tile t. If there is no robot on tile t then t is marked as clear, i.e., $\textit{clear}(t)$ holds. Robots can move left, right, up or down, if the target tile is clear. Each robot a is equipped with a brush that is configured to either paint in $black$ or $white$, e.g., $robot\text{-}has(a, black)$ is true iff the brush of robot a is configured to paint in $black$. It is possible to change the color infinitely often. The goal is to paint a rectangular subset of the grid in chessboard style. If a tile t has color c then the predicate $painted(t, c)$ holds and additionally the tile is marked as not clear, i.e., $clear(t)$ does not hold. A robot a can only paint tile t if a is on a tile t' that is below or above t and t is clear, i.e., $robot\text{-}at(a, t')$ holds, $up(t', t)$ or $down(t', t)$ holds, and $clear(t)$ holds.

Consider the set of features $\Phi = \{g, v\}$ where g counts the number of unpainted tiles that need to be painted and v represents that the following condition is satisfied: for each tile t_1 that remains to be painted there exists a sequence of tiles $t_1, \ldots, t_n$ such that each t_i with $i = 1, \ldots, n-1$ remains to be painted, t_n does not need to be painted, and for all pairs t_{i-1}, t_i with $i = 2, \ldots, n$ holds that t_i is above t_{i-1}, i.e., $up(t_{i-1}, t_i)$, or for all pairs t_{i-1}, t_i with $i = 2, \ldots, n$ it holds that t_i is below t_{i-1}, i.e., $down(t_{i-1}, t_i)$. Intuitively, v is true iff a given state is solvable. The set of sketch rules R^F_Φ contains the single rule

$$r = \{v, g > 0\} \mapsto \{g\downarrow\}$$

which says that painting a tile such that the invariant v remains satisfied is good.

Theorem 5. *The sketch R^F_Φ for the Floortile domain is well-formed and has width* 2.

Proof. Recall that a sketch is well-formed if it uses goal-separating features and is terminating. The features Φ are goal separating because the feature valuation $g = 0$ holds in state s iff s is a goal state. The sketch R_Φ is terminating because r decreases the numerical feature g and no other rule increases g.

It remains to show that the sketch width is 2. Consider a Floortile instance P with states S. If the initial state s is a solvable non-goal state, then the feature conditions of r are true, and the subgoal $G_r(s)$ is nonempty. If we reach such a subgoal state, then either the feature conditions of r remain true because the invariant remains satisfied or the overall goal was reached. Next, we show that $P[s]$ with subgoal $G_r(s)$ in R-reachable state s has width 2. Consider states $S_1 \subseteq S$ where the feature conditions of rule r are true, i.e., solvable states where a tile t must be painted in a color c. We do a three-way case distinction over states S_1.

First, consider states $S^1_1 \subseteq S_1$ where some robot a on tile t_1 that is configured to color c, can move to tile t_n above or below t to paint it. The singleton tuple $painted(t, c)$ implies $G_r(s)$ in $s \in S^1_1$ in the admissible chain that consists of moving a from t_1 to t_n, while decreasing the distance to t_n in each step, and painting t, i.e.,

$$(robot\text{-}at(a, t_1), \ldots, robot\text{-}at(a, t_n), painted(t, c)).$$

Second, consider states $S^2_1 \subseteq S_1$ where the robot a must reconfigure its color from c' to c before painting. The tuple $(robot\text{-}at(a, t_n), painted(t, c))$ implies $G_r(s)$ in $s \in S^2_1$ in the admissible chain that consists of reconfiguring the color, and then moving closer and painting as before, i.e.,

$$\begin{aligned}&((robot\text{-}at(a, t_1), robot\text{-}has(a, c')),\\ &\ (robot\text{-}at(a, t_1), robot\text{-}has(a, c)), \ldots,\\ &\ (robot\text{-}at(a, t_n), robot\text{-}has(a, c)),\\ &\ (robot\text{-}at(a, t_n), painted(t, c))).\end{aligned}$$

We observe that reconfiguring requires an admissible chain of size 2 because of serializing the reconfiguring and the moving part. Therefore, in the following case, we assume that the robot must reconfigure its color.

Third, consider states $S_1^3 \subseteq S_1$ where robot a is standing on t and there is a sequence of robots $a_1, \ldots, a_n$ such that a can only paint t if each $a_1, \ldots, a_n$ moves in such a way that tile t' above or below t becomes clear. Using the fact that a rectangular portion inside a rectangular grid has to be painted, it follows that the set of tiles that must not be painted is pairwise connected. Therefore, we can move each robot a_i from its current tile t_i' to t_i in such a way that after moving each robot, tile t becomes clear. The tuple $(\textit{robot-at}(a, t'), \textit{painted}(t, c))$ implies $G_r(s)$ in $s \in S_1^3$ in the admissible chain that consists of moving each robot a_i from t_i' to t_i in such a way that moving all of them clears tile t', followed by moving a to t', and painting t, i.e.,

$$\begin{aligned}
&((\textit{robot-at}(a, t), \textit{robot-has}(a, c')),\\
&\ (\textit{robot-at}(a, t), \textit{robot-has}(a, c)),\\
&\ (\textit{robot-at}(a_1, t_1), \textit{robot-has}(a, c)), \ldots,\\
&\ (\textit{robot-at}(a_n, t_n), \textit{robot-has}(a, c)),\\
&\ (\textit{robot-at}(a, t'), \textit{robot-has}(a, c)),\\
&\ (\textit{robot-at}(a, t'), \textit{painted}(t, c)))
\end{aligned}$$

We obtain sketch width 2 because all tuples in admissible chains have size of at most 2. □

5.3 TPP

In the Traveling Purchaser Problem (TPP) domain, there is a set of places that can either be markets or depots, a set of trucks, and a set of goods (Gerevini, Haslum, Long, Saetti, & Dimopoulos, 2009). The places are connected via a roads, allowing trucks to drive between them. If a truck t is at place p, then atom $at(t, p)$ holds. Each market p sells specific quantities of goods, e.g., atom $\textit{on-sale}(g, p, 2)$ represents that market p sells two quantities of good g. If there is a truck t available at market p, it can buy a fraction of the available quantity of good g, making g available to be loaded into t, while the quantity available at p decreases accordingly, i.e., atom $\textit{on-sale}(g, p, 1)$ and $\textit{ready-to-load}(g, p, 1)$ hold afterwards. The trucks can unload the goods at any depot, effectively increasing the number of stored goods, e.g., atom $\textit{stored}(g, 1)$ becomes false, and $\textit{stored}(g, 2)$ becomes true, indicating that two quantities of good g are stored. The goal is to store specific quantities of specific goods.

SIW fails in TPP because loading sufficiently many quantities of a single good can require buying and loading them from different markets. Making the goods available optimally requires taking the direct route to each market followed by buying the quantity of goods. Thus, the problem width is bounded by the number of quantities needed.

Consider the set of features $\Phi = \{b, l, n\}$ where b is the number of ready-to-be-loaded goods that are not bought at a market and of which some quantity remains to be stored, l is the number of goods of which some quantity remains to be stored and are not loaded in a truck, and n is the sum of quantities of goods that remain to be stored. The sketch rules in R_Φ^T are defined as

$$\begin{aligned} r_1 &= \{b > 0\} \mapsto \{b\downarrow\} \\ r_2 &= \{l > 0\} \mapsto \{b?, l\downarrow\} \\ r_3 &= \{n > 0\} \mapsto \{b?, l?, n\downarrow\}, \end{aligned}$$

where rule r_1 says that buying any quantity of a good that remains to be stored is good, rule r_2 says that loading any quantity of a good that remains to be stored is good, and rule r_3 says that storing any quantity of a good for which have not yet stored enough is good.[2]

Theorem 6. *The sketch R_Φ^T for the TPP domain is well-formed and has width* 1.

Proof. The features are goal separating because $n = 0$ holds in state s iff s is a goal state. We show that the sketch R_Φ is terminating by iteratively eliminating rules: r_3 decreases the numerical feature n which no other rule increments, so we eliminate r_3 and mark n. Next, r_2 decreases l which no other rule increments, so we eliminate r_2 and mark l. Now only r_1 remains, and we can eliminate it since it decreases b, which is never incremented.

It remains to show that the sketch width is 1. Consider any TPP instance P. In the initial state s, the feature conditions of at least one rule r are true and the corresponding subgoal $G_r(s)$ is nonempty. In the subgoal states $s' \in G_{r_1}(s)$ of some state s the feature conditions of r_2 must be true, the set of subgoal states $G_{r_2}(s')$ is nonempty, and the feature conditions of r_1 can remain true and the set of subgoal states remains nonempty. Similarly, in the subgoal states $s' \in G_{r_2}(s)$ of some state s the feature conditions of r_3 must be true, the set of subgoal states $G_{r_3}(s')$ is nonempty, and the feature conditions of r_1 and r_2 can remain true and the set of subgoal states remains nonempty. At some point, the subgoal of r_3 is the overall goal of the problem. Next, we show that the sketch has width 1.

First, we consider rule r_1. Intuitively, we show that buying a good, that is not yet ready to be loaded but of which some quantity remains to be stored in a depot, has width at most 1. Consider states $S_1 \subseteq S$ where the feature conditions of r_1 are true, i.e., states where there is no good g bought and therefore ready to be loaded in a truck but of which some quantity remains to be stored in a depot. With $G_{r_1}(s)$ we denote the subgoal states of r_1 in $s \in S_1$, i.e., states where some quantity q_b of g is bought and therefore ready to be loaded. The tuple $\textit{ready-to-load}(g, p, q_b)$ implies $G_{r_1}(s)$ in $s \in S_1$ in the admissible chain that consists of moving a truck t from its current place p_1 to the closest market p_n where nonzero quantities q_a of g are available, ordered descendingly by their distance to p_n, buying $q_b \leq q_a$ quantities of g, i.e., $(\textit{at}(t, p_1), \ldots, \textit{at}(t, p_n), \textit{ready-to-load}(g, p_n, q_b))$.

Second, we consider rule r_2. Intuitively, we show that loading a good that is not yet loaded but of which some quantity remains to be stored in a depot has width at most 1. Consider states $S_2 \subseteq S$ where the feature conditions of r_2 are true, and some quantity of a good g that remains to be stored is ready to be loaded (see above), i.e., states where there is no truck that has g loaded, but of which

2. The theorem for the TPP sketch in our previous work contains an error (Drexler et al., 2021). The original sketch only contained the features l and n, and the rules r_2 and r_3, and we claimed that the subproblem of loading a quantity of a good that remains to be stored has width 1. This is wrong because making a good available to be loaded requires *any* truck to be at the market, but all those optimal plans cannot be extended into an optimal plan for loading the good into *one* specific truck. However, one can show that the *effective* width of such a subproblem is 1 because it is irrelevant into which truck we load a good. To obtain a width-1 sketch, we must add another feature b and rule r_1. The rules r_2, r_3 are semantically identical as before, but have an additional effect $b?$ for allowing the Boolean feature b to change arbitrarily.

some quantity remains to be stored in a depot, and there is some nonzero quantity q_b of g at place p_n ready to be loaded. With $G_{r_2}(s)$ we denote the subgoal states of r_2 in $s \in S_2$, i.e., states where some quantity q_l of g is loaded into a truck t. The tuple $\mathit{loaded}(g, t, q_l)$ implies $G_{r_2}(s)$ in $s \in S_2$ in the admissible chain that consists of moving t from its current place p_1 to the closest market p_n, ordered descendingly by their distance to p_n, loading $q_l \leq q_b$ quantities of g, i.e., $(\mathit{at}(t, p_1), \ldots, \mathit{at}(t, p_n), \mathit{loaded}(g, t, q_l))$.

Last, we consider rule r_3. Intuitively, we show that storing a good of which some quantity remains to be stored in a depot has width at most 1. Consider states $S_3 \subseteq S$ where the feature conditions of r_3 are true and some quantity of a good that remains to be stored is loaded (see above), i.e., states where some quantity of a good g remains to be stored in a depot, and some nonzero quantity q_l of g is loaded into a truck t because it has width 1 (see above). With $G_{r_3}(s)$ we denote the subgoal states of r_3 in $s \in S_3$, i.e., states where the remaining quantity of g that remains to be stored has decreased. The tuple $\mathit{stored}(g, q'_s)$ implies $G_{r_3}(s)$ in $s \in S_3$ in the admissible chain that consists of moving t from its current place p_1 to the closest depot at place p_n, ordered descendingly by their distance to p_n, storing $q'_l \leq q_l$ quantities of g, i.e., $(\mathit{at}(t, p_1), \ldots, \mathit{at}(t, p_n), \mathit{stored}(g, q'_s))$.

We obtain sketch width 1 because all tuples in admissible chains have a size of at most 1. $\square$

5.4 Barman

In the Barman domain (Linares López et al., 2015), there is a set of shakers, a set of shots, and a set of dispensers where each dispenses a different ingredient. There are recipes of cocktails, each consisting of two ingredients, e.g., the recipe for cocktail c consists of ingredients i_1, i_2. The goal is to serve beverages, i.e., ingredients and/or cocktails. A beverage b is served in shot g if g contains b. An ingredient i can be filled into shot g using one of the dispensers if g is clean. Producing a cocktail c with a shaker t requires both ingredients i_1, i_2 of c to be in t. In such a situation, shaking t produces c. Pouring a cocktail from t into shot g requires g to be clean. The barman robot has two hands which limits the number of shots and shakers it can hold at the same time. Therefore, the barman often has to put down an object before it can grasp a different object. For example, assume that the barman holds the shaker t and some shot g' and assume that ingredient i must be filled into shot g. Then the barman has to put down either t or g' so that it can pick up g with hand h. As in the Barman tasks from previous IPCs, we assume that there is only a single shaker and that it is initially empty.

Consider the set of features $\Phi = \{g, u, c_1, c_2\}$ where g is the number of unserved beverages, u is the number of used shots, i.e., shots with a beverage different from the one mentioned in the goal, c_1 is true iff the first recipe ingredient of an unserved cocktail is in the shaker, and c_2 is true iff both recipe ingredients of an unserved cocktail are in the shaker. We define the following sketch rules for R^B_Φ:

$$\begin{aligned}
r_1 &= \{\neg c_1\} \mapsto \{u?, c_1\}, \\
r_2 &= \{c_1, \neg c_2\} \mapsto \{u?, c_2\}, \\
r_3 &= \{u > 0\} \mapsto \{u\downarrow\}, \\
r_4 &= \{g > 0\} \mapsto \{g\downarrow, c_1?, c_2?\}.
\end{aligned}$$

Rule r_1 says that filling an ingredient into the shaker is good if this ingredient is mentioned in the first part of the recipe of an unserved cocktail. Rule r_2 says the same for the second ingredient, after the first ingredient has been added. Requiring the ingredients to be filled into the shaker in a fixed order ensures bounded width, even for arbitrary-sized recipes. Rule r_3 says that cleaning shots is good and rule r_4 says that serving an ingredient or cocktail is good.

Theorem 7. *The sketch R^B_Φ for the Barman domain is well-formed and has width* 2.

Proof. The features Φ are goal separating because $g = 0$ holds in state s iff s is a goal state. We show that the sketch is terminating by iteratively eliminating rules: first, we eliminate r_4 because it decreases the numerical feature g that no rule increases. Next, rules r_1 and r_2 can be eliminated because both change a Boolean feature that no remaining rule changes in the opposite direction. Last, we eliminate the r_3 because it decrements the numerical feature u.

It remains to show that the sketch width is 2. Consider any Barman instance P with states S. In the initial state s the feature conditions of r_4 are true, and the subgoal $G_{r_4}(s)$ is nonempty. Note that using r_4 to reach a subgoal decreases the number of unserved beverages until the overall goal is reached. Hence, r_4 can be seen as the goal counter. If the beverage to be served is a cocktail or if the shots are dirty, then this subproblem can be further decomposed into smaller subproblems using rules r_1, r_2, r_3 as follows. Producing a cocktail requires filling the shaker with correct ingredients and can be achieved by successively reaching the subgoals defined by rules r_1 and r_2. Next, if the shot required for serving was made dirty during this process, then r_3 defines the subgoal of cleaning it again. Finally, r_4 defines the subgoal of serving the cocktail.

We first consider rule r_3. Intuitively, we show that shots are cleaned with width at most 1. Consider all states $S_1 \subseteq S$ where the feature conditions of r_3 are true, i.e., states where there is a used shot g such that $\mathit{used}(g, b)$ holds for some beverage b that is not supposed to be in g according to the goal description. With $G_{r_3}(s)$ we denote the subgoal states of r_3 in $s \in S_1$, i.e., states where g is clean. We do a case distinction over states S_1. First, consider states $S^1_1 \subseteq S_1$ where the barman is holding g in hand h. The tuple $\mathit{clean}(g)$ implies $G_{r_3}(s)$ for all $s \in S^1_1$ in the admissible chain that consists of cleaning g, i.e., $(\mathit{holding}(h, g), \mathit{clean}(g))$. Second, consider states $S^2_1 \subseteq S_1$ where the barman must grasp g with empty hand h first. The same tuple $\mathit{clean}(g)$ implies $G_{r_3}(s)$ for all $s \in S^2_1$ in the admissible chain that consists of picking g, and cleaning g, i.e., $(\mathit{ontable}(g), \mathit{holding}(h, g), \mathit{clean}(g))$. Last, consider states $S^3_1 \subseteq S_1$ where the barman must exchange g' with g in hand h first. The same tuple $\mathit{clean}(g)$ implies $G_{r_3}(s)$ for all $s \in S^3_1$ in the admissible chain that consists of putting down g', picking up g, cleaning g, i.e., $(\mathit{holding}(h, g'), \mathit{ontable}(g'), \mathit{holding}(h, g), \mathit{clean}(g))$. It also follows that we can reduce the set of R-reachable states in our analysis to those where the container is already grasped if only a single container is affected.

Next, we consider rule r_1. Intuitively, we show that filling the first ingredient into the shaker for producing a required cocktail has width at most 2. Consider states $S_2 \subseteq S$ where the feature conditions of r_1 are true and required shots are clean, i.e., states where no ingredient i_1 consistent with the first part of some unserved cocktail c's recipe is in the shaker t. We do not need to consider states where required shots are not clean because a shot can be cleaned with width 1 (see above). With $G_{r_1}(s)$ we denote the subgoal states of r_1 in s, i.e., states where an ingredient i_1 consistent with the first recipe part of some unserved cocktail c is inside t. The tuple $(\mathit{contains}(t, i_1), \mathit{shaker\text{-}level}(t, l1))$ implies $G_{r_1}(s)$ in the admissible chain that consists of cleaning t, putting down t, picking a clean shot g, filling i_1 into g using the corresponding dispenser, and pouring g into t, i.e.,

$$\begin{aligned}&((\mathit{holding}(h,t), \mathit{shaker\text{-}level}(t,l2)),\\ &(\mathit{holding}(h,t), \mathit{empty}(t)),\\ &(\mathit{holding}(h,t), \mathit{clean}(t)),\\ &(\mathit{ontable}(t), \mathit{clean}(t)),\\ &(\mathit{holding}(h,g), \mathit{clean}(t)),\\ &(\mathit{contains}(g,i_1), \mathit{clean}(t)),\\ &(\mathit{contains}(t,i_1), \mathit{shaker\text{-}level}(t,l1))).\end{aligned}$$

Note that the feature conditions of rule r_2 are true in states s' in subgoal $G_{r_1}(s)$ with nonempty subgoal $G_{r_2}(s')$.

Next, we consider rule r_2. Intuitively, we show that filling the second ingredient into the shaker for producing a required cocktail has width at most 1. Consider states $S_3 \subseteq S$ where the feature conditions of r_2 are true and required shots are clean, i.e., states where the first ingredient consistent with the recipe of an unserved cocktail c is in the shaker t, and required shots are clean because a shot can be cleaned with width 1 (see above). With $G_{r_2}(s)$ we denote the subgoal states of r_2 in s, i.e., states where an ingredient i_2 is inside t such that both ingredients in t are consistent with the recipe of an unserved cocktail c. The tuple $(\mathit{contains}(t,i_2), \mathit{shaker\text{-}level}(t,l2))$ implies $G_{r_2}(s)$ in the admissible chain that consists of putting down t, grasping g, filling i_2 into g using the corresponding dispenser, and pouring g into t, i.e.,

$$\begin{aligned}&((\mathit{holding}(h,t), \mathit{shaker\text{-}level}(t,l1)),\\ &(\mathit{ontable}(t), \mathit{shaker\text{-}level}(t,l1)),\\ &(\mathit{holding}(h,g), \mathit{ontable}(t)),\\ &(\mathit{contains}(g,i_2), \mathit{ontable}(t)),\\ &(\mathit{contains}(t,i_2), \mathit{shaker\text{-}level}(t,l2))).\end{aligned}$$

Finally, we consider rule r_4, where we show intuitively that serving a beverage has width at most 1. We do a case distinction over all states S_4 where the feature conditions of r_4 are true, i.e., states where there is an unserved ingredient or an unserved cocktail. First, consider states $S_4^1 \subseteq S_4$ where there is an unserved ingredient i. $G_{r_4}^1(s)$ is the set of subgoal states for r_4 in $s \in S_4^1$ where i is served. The tuple $\mathit{contains}(g,i)$ implies $G_{r_4}^1(s)$ in the admissible chain that consists of filling i into g using the corresponding dispenser, i.e., $(\mathit{clean}(g), \mathit{contains}(g,i))$. Last, consider states $S_4^2 \subseteq S_4$ where there is an unserved cocktail c, respective ingredients are in the shaker using the results of rule r_1, r_2, and required shots are clean using the results of rule r_3. With $G_{r_4}^2(s)$ we denote the subgoal states of r_4 in $s \in S_4^2$ where c is served. The tuple $\mathit{contains}(g,c)$ implies $G_{r_4}^2(s)$ in the admissible chain that consists of putting down g (or any other shot) because shaking requires only the shaker t to be held, shaking t, and pouring t into g, i.e.,

$$(\mathit{holding}(h,g), \mathit{ontable}(g), \mathit{contains}(t,c), \mathit{contains}(g,c))$$

We obtain sketch width 2 because all tuples in admissible chains have a size of at most 2. $\square$

5.5 Grid

In the Grid domain (McDermott, 2000), a single robot operates in a grid-structured world. There are keys and locks distributed over the grid cells. The robot can move to a cell c above, below, left or right of its current cell if c does not contain a closed lock or another robot. The robot can drop, pick or exchange keys at its current cell and can only hold a single key e at any time. Keys and locks have different shapes and the robot, holding a matching key, can open a lock when standing on a neighboring cell. The goal is to move keys to specific target locations that can be locked initially. Initially, it is possible to reach every lock for the unlock operation. SIW fails in this domain when goals need to be undone, i.e., a key has to be picked up from its target location to open a lock that is necessary for picking or moving a different key.

Consider the set of features $\Phi = \{l, k, o, t\}$ where l is the number of locked grid cells, k is the number of misplaced keys, o is true iff the robot holds a key for which there is a closed lock, and t is true iff the robot holds a key that must be placed at some target grid cell. We define the sketch rules for R^G_Φ as:

$$\begin{aligned}
r_1 &= \{l > 0\} \mapsto \{l\downarrow, k?, o?, t?\} \\
r_2 &= \{l = 0, k > 0\} \mapsto \{k\downarrow, o?, t?\} \\
r_3 &= \{l > 0, \neg o\} \mapsto \{o, t?\} \\
r_4 &= \{l = 0, \neg t\} \mapsto \{o?, t\}
\end{aligned}$$

Rule r_1 says that unlocking grid cells is good. Rule r_2 says that placing a key at its target cell is good after opening all locks. Rule r_3 says that picking up a key that can be used to open a locked grid cell is good if there are locked grid cells. Rule r_4 says that picking up a misplaced key is good after opening all locks.

Theorem 8. *The sketch R^G_Φ for the Grid domain is well-formed and has width* 2.[3]

Proof. The features Φ are goal separating because the feature valuation $k = 0$ holds in state s iff s is a goal state. We show that the sketch is terminating by iteratively eliminating rules: r_1 decreases l which no other rule increases, so we eliminate r_1 and mark l. Now r_2 can be eliminated because it decreases k which no remaining rule increases. We can now eliminate $r_3 = C \mapsto E$ because it changes the Boolean feature o and the only other remaining rule $r_4 = C' \mapsto E'$ may restore the value of o, but this can only happen finitely often, since l is marked and $l > 0 \in C$ and $l = 0 \in C'$. Now only r_4 remains and we can eliminate it since it changes t, which is never changed back.

It remains to show that the sketch width is 1. Consider any Grid instance P with states S. Note that depending on the initial state s the feature conditions of at least one rule r are true in s and its subgoal $G_r(s)$ are nonempty. We first consider rule r_3. Intuitively, we show that picking up a key that can be used to open some closed lock has width 1. Consider states $S_1 \subseteq S$ where the

3. The proof for the Grid sketch in our previous work (Drexler et al., 2021) contains an error: we claimed that the subproblems of moving a key to its target cell have width 1. This is wrong because placing a key at a target location by exchanging it with another key that is at its goal location, results in a state that is not a subgoal state. Hence, the correct atom tuple in the admissible chain of rule r_2, which also captures that the arm must be empty, is $(at(e, c_n), \mathit{arm\text{-}empty}())$ of size 2. A sketch of width 1 can be obtained by splitting the problem into moving to the target location of the key with width 1 and then dropping the key with width 0. Since the exchange action can be simulated with a pick and drop action, the above sketch still has a width of 1 for the domain variant where the exchange action is removed entirely.

feature conditions of r_3 are true, i.e., states where there is a closed lock and the robot does not hold a key e that can be used to open a closed lock. With $G_{r_3}(s)$ we denote the subgoal states of r_3 in $s \in S_1$, i.e., states where the robot holds e. The tuple $\mathit{holding}(e)$ implies $G_{r_3}(s)$ in $s \in S_1$ in the admissible chain that consists of changing the position of the robot from the current position c_1 to the position c_n of e ordered by the distance to c_n, and followed by exchanging or picking e, i.e., $(\mathit{at\text{-}robot}(c_1), \dots, \mathit{at\text{-}robot}(c_n), \mathit{holding}(e))$. Note that the feature conditions of r_1 are true in states s' in subgoal $G_{r_3}(s)$ with nonempty subgoal states $G_{r_1}(s')$ because the number of closed locks remains greater than 0.

Next, we consider rule r_1. Intuitively, we show that opening a closed lock has width 1. Consider states $S_2 \subseteq S$ where feature conditions of r_1 are true and the robot holds a key e that can be used to open a closed lock d. We can transform states where the robot holds no key into a state from S_2 by letting it pick a key with width 1 (see above). With $G_{r_1}(s)$ we denote the subgoal states of r_1 in $s \in S_2$, i.e., states where d is open. The tuple $\mathit{open}(d)$ implies $G_{r_1}(s)$ in $s \in S_2$ in the admissible chain that consists of changing the position of the robot from its current position c_1 to a position c_n next to lock d ordered by the distance to c_n, i.e., $(\mathit{at\text{-}robot}(c_1), \dots, \mathit{at\text{-}robot}(c_n), \mathit{open}(d))$. Note that either there are still closed locks that can be opened by repeated usage of rules r_1 and r_3 or the feature conditions of r_2 or r_4 are true in states s' in the subgoal $G_{r_1}(s)$ with nonempty subgoal states $G_{r_2}(s')$ or $G_{r_4}(s')$ respectively because there are misplaced keys. Hence, it remains to show that if all locks are open then well-placing keys has width 1.

Next, we consider rule r_4. Intuitively, we show that picking up a key that is not at its target cell has width 1. Consider states $S_3 \subseteq S$ where the feature conditions of r_4 are true, i.e., states where all locks are open and the robot does not hold a misplaced key. With $G_{r_4}(s)$ we denote the subgoal states of r_4 in $s \in S_3$, i.e., states where the robot holds e. The tuple $\mathit{holding}(e)$ implies $G_{r_4}(s)$ in $s \in S_3$ in the admissible chain that consists of changing the position of the robot from the current position c_1 to the position c_n of e ordered by the distance to c_n, and followed by exchanging or picking e, i.e., $(\mathit{at\text{-}robot}(c_1), \dots, \mathit{at\text{-}robot}(c_n), \mathit{holding}(e))$. Note that the feature conditions of r_2 are true in states s' in subgoal $G_{r_4}(s)$ with nonempty subgoal states $G_{r_2}(s')$ because the number of misplaced keys remains greater than 0.

Finally, we consider rule r_2. Intuitively, we show that moving a key to its target cell has width 2. Consider states $S_4 \subseteq S$ where the feature conditions of r_2 are true and the robot holds a misplaced key e. As before, we can transform states $s' \notin S_4$ into such a state s by picking up e with width 1. With $G_{r_2}(s)$ we denote the subgoal states of r_2 in $s \in S_4$, i.e., states where e is at its target cell. The tuple $(\mathit{at}(e, c_n), \mathit{arm\text{-}empty}())$ implies $G_{r_2}(s)$ in $s \in S_4$ in the admissible chain that consists of changing the position of the robot from its current position c_1 to the key's target cell c_n ordered by the distance to c_n, followed by dropping e at c_n, i.e., $(\mathit{at\text{-}robot}(c_1), \dots, \mathit{at\text{-}robot}(c_n), (\mathit{at}(e, c_n), \mathit{arm\text{-}empty}()))$.

We obtain sketch width 2 because all tuples in admissible chains have size at most 2. $\square$

5.6 Childsnack

In the Childsnack domain (Vallati, Chrpa, & McCluskey, 2018), there is a set of contents, a set of trays, a set of gluten-free breads, a set of regular breads that contain gluten, a set of gluten-allergic children, a set of children without gluten allergy, and a set of tables where the children sit. The goal is to serve the gluten-allergic children with sandwiches made of gluten-free bread and the non-allergic children with either type of sandwich.

The Childsnack domain has large bounded width because moving an empty tray is possible at any given time. The goal serialization fails because it gets trapped in deadends when serving non-allergic children with gluten-free sandwiches while leaving insufficiently many gluten-free sandwiches for the allergic children.

Consider the set of features $\Phi = \{c_g, c_r, s_g^k, s^k, s_g^t, s^t\}$ where c_g is the number of unserved gluten-allergic children, c_r is the number of unserved non-allergic children, s_g^k holds iff there is a gluten-free sandwich in the kitchen, s^k holds iff there is any sandwich in the kitchen, s_g^t holds iff there is a gluten-free sandwich on a tray, and s^t holds iff there is any sandwich on a tray. We define the following sketch rules R_Φ^C:

$$
\begin{aligned}
r_1 &= \{c_g > 0, \neg s_g^k, \neg s_g^t\} \mapsto \{s_g^k, s^k\}\\
r_2 &= \{c_g = 0, c_r > 0, \neg s^k, \neg s^t\} \mapsto \{s^k\}\\
r_3 &= \{c_g > 0, s_g^k, \neg s_g^t\} \mapsto \{s_g^k?, s^k?, s_g^t, s^t\}\\
r_4 &= \{c_g = 0, c_r > 0, s^k, \neg s^t\} \mapsto \{s_g^k?, s^k?, s_g^t?, s^t\}\\
r_5 &= \{c_g > 0, s_g^t\} \mapsto \{c_g\downarrow, s_g^t?, s^t?\}\\
r_6 &= \{c_g = 0, c_r > 0, s^t\} \mapsto \{c_r\downarrow, s_g^t?, s^t?\}
\end{aligned}
$$

Rule r_1 says that making a gluten-free sandwich is good if there is an unserved gluten-allergic child and if there is no other gluten-free sandwich currently being served. Rule r_2 says the same thing for non-allergic children after all gluten-allergic children have been served and the sandwich to be made is not required to be gluten free. Rules r_3 and r_4 say that putting a gluten-free (resp. regular) sandwich from the kitchen onto a tray is good if there is none on a tray yet. Rule r_5 says that serving gluten-allergic children before non-allergic children is good if there is a gluten-free sandwich available on a tray. Rule r_6 says that serving non-allergic children afterwards is good.

Theorem 9. *The sketch R_Φ^C for the Childsnack domain is well-formed and has width* 1.

Proof. The features are goal separating because the feature valuations $c_g = 0$ and $c_r = 0$ hold in state s iff s is a goal state. We show that the sketch is terminating by iteratively eliminating rules: r_5 decreases the numerical feature c_g which no other rule increments, so we eliminate r_5 and mark c_g. Similarly, r_6 decreases the numerical feature c_r which no other rule increments, so we eliminate r_6 and mark c_r. Then rules r_4 changes s^t and no remaining rules changes s^t in the opposite direction, so we eliminate r_4. Likewise, we eliminate r_3 because it changes s_g^t, which no remaining rule can change back. Last, we eliminate rules r_1 and r_2 because they change s_g^k resp. s^k, and no remaining rule can change the values in the opposite direction.

It remains to show that the sketch width is 1. Consider any Childsnack instance with states S. Note that if there is an unserved gluten-allergic child in the initial state then rules r_1, r_3, r_5 define subgoals for serving a gluten-allergic child. If there is no unserved gluten-allergic child but there is an unserved non-allergic child then rules r_2, r_4, r_6 define subgoals for serving a non-allergic child. In the following, we first show that serving a gluten-free sandwich to a gluten-allergic child has width 1 and deduce the case of serving a non-allergic child from it.

We first consider rule r_1. Intuitively, we show that producing a gluten-free sandwich has width 1. Consider states $S_3 \subseteq S$ where the feature conditions of r_1 are true, i.e., states where there is an unserved gluten-allergic child c and there is no gluten-free sandwich available in $kitchen$ nor on a tray. With $G_{r_1}(s)$ we denote the subgoal states of r_1 in $s \in S_3$, i.e., states where gluten-free sandwich s is available in $kitchen$. The tuple $no\text{-}gluten\text{-}sandwich(s)$ implies $G_{r_1}(s)$ in $s \in S_3$ in the admissible chain that consists of producing s, i.e., $(notexists(s), no\text{-}gluten\text{-}sandwich(s))$. Note that the feature conditions of r_3 are true in states s' in the subgoal $G_{r_1}(s)$ with nonempty subgoal $G_{r_3}(s')$.

Next, we consider rule r_3. Intuitively, we show that moving a gluten-free sandwich from the kitchen onto a tray has width 1. Consider states $S_2 \subseteq S$ where the feature conditions of r_3 are true, i.e., states where there is an unserved gluten-allergic child c and there is a gluten-free sandwich s available in $kitchen$. With $G_{r_3}(s)$ we denote the subgoal states of r_3 in $s \in S_2$, i.e., states where s is on p. The tuple $ontray(s,p)$ implies $G_{r_3}(s)$ in $s \in S_2$ in the admissible chain that consists of moving p from t to $kitchen$, putting s onto p, i.e., $(at(p,t), at(p, kitchen), ontray(s,p))$. Note that the feature conditions of r_5 are true in states s' in the subgoal $G_{r_3}(s)$ with nonempty subgoal $G_{r_5}(s')$.

Next, we consider rule r_5. Intuitively, we show that serving a gluten-allergic child if there is a gluten-free sandwich is available on a tray has width 1. Consider states $S_1 \subseteq S$ where the feature conditions of r_5 are true, i.e., states where there is an unserved gluten-allergic child c and there is a gluten-free sandwich s on a tray p. With $G_{r_5}(s)$ we denote the subgoal states of r_5 in $s \in S_1$, i.e., states where c is served. The tuple $served(c)$ implies $G_{r_5}(s)$ in $s \in S_1$ in the admissible chain that consists of moving p from $kitchen$ to t, serving c with s, i.e., $(ontray(s,p), at(p,t), served(c))$. Note that if all gluten allergic children are served in this way by using rules r_1, r_3, r_5 then either G was reached or there are unserved non-allergic children. In the latter case, the problem is very similar to the one we considered above and rules r_2, r_4, r_6 define the corresponding subgoals to serve a non-allergic child. We omit the details but provide the admissible chains that are necessary to conclude the proof: the tuple $served(c)$ implies $G_{r_6}(s)$ in the admissible chain $(ontray(s,p), at(p,t), served(c))$. The tuple $ontray(s,p)$ implies $G_{r_4}(s)$ in the admissible chain $(at(p,t), at(p, kitchen), ontray(s,p))$. The tuple $at\text{-}kitchen\text{-}sandwich(s)$ implies $G_{r_2}(s)$ in the admissible chain $(notexists(s)), at\text{-}kitchen\text{-}sandwich(s))$.

We obtain sketch width 1 because all tuples in admissible chains have a size of at most 1. $\square$

5.7 Driverlog

In the Driverlog domain (Long & Fox, 2003), there is a set of drivers, trucks, packages, road locations and path locations. The two types of locations form two strongly connected graphs and the two sets of vertices overlap. The road graph is only traversable by trucks, while the path graph is only traversable by drivers. A package can be delivered by loading it into a truck, driving the truck to the target location of the package followed by unloading the package. Driving the truck requires a driver to be in the truck. Not only packages, but also trucks and drivers can have goal locations. SIW fails because it can be necessary to undo previously achieved goals, like moving a truck away from its destination to transport a package. The following sketch induces a goal ordering such that an increasing subset of goal atoms never needs to be undone.

Consider the set of features $\Phi = \{p, t, d_g, d_t, b, l\}$ where p is the number of misplaced packages, t is the number of misplaced trucks, d_g is the sum of all distances of drivers to their respective goal locations, d_t is the minimum distance of any driver to a misplaced truck, b is true iff there is a driver inside of a truck, and l is true iff there is a misplaced package in a truck. We define the sketch rules R_Φ^D as follows:

$$\begin{aligned}
r_1 &= \{p > 0, \neg b\} \mapsto \{d_g?, d_t?, b\} \\
r_2 &= \{p > 0, \neg l\} \mapsto \{t?, d_g?, d_t?, l\} \\
r_3 &= \{p > 0\} \mapsto \{p\downarrow, t?, d_g?, d_t?, l?\} \\
r_4 &= \{p = 0, t > 0, d_t > 0\} \mapsto \{d_g?, d_t\downarrow, b?\} \\
r_5 &= \{p = 0, t > 0, d_t = 0\} \mapsto \{t\downarrow, d_g?, d_t?\} \\
r_6 &= \{p = 0, t = 0, d_g > 0\} \mapsto \{d_g\downarrow, b?\}
\end{aligned}$$

Rule r_1 says that letting a driver board any truck is good if there are undelivered packages and there is no driver boarded yet. Rule r_2 says that loading an undelivered package is good. Rule r_3 says that delivering a package is good. Rule r_4 says that moving any driver closer to being in a misplaced truck is good after having delivered all packages. Rule r_5 says that driving a misplaced truck to its target location is good once all packages are delivered. Rule r_6 says that moving a misplaced driver closer to its target location is good after having delivered all packages and trucks.

Theorem 10. *The sketch R_Φ^D for the Driverlog domain is well-formed and has width* 1.

Proof. The features are goal separating because the feature valuations $p = 0, t = 0, d_g = 0$ hold in state s iff s is a goal state. We show that the sketch is terminating by iteratively eliminating rules: r_3 decreases the numerical feature p that no other remaining rule increments, so we eliminate r_3 and mark p. We can now eliminate $r_5 = C \mapsto E$ because it decreases the numerical feature t and the only other remaining rule $r_2 = C' \mapsto E'$ arbitrarily changes t, but this can only happen finitely many times, since p is marked and $p = 0 \in C$ and $p > 0 \in C'$. Next, we can eliminate r_2 because it sets the Boolean feature l and no other remaining rule changes l in the opposite direction. We can now eliminate $r_4 = C \mapsto E$ because it decreases the numerical feature d_t and the only other remaining rule $r_1 = C' \mapsto E'$ arbitrarily changes d_t, but this can only happen finitely many times, since p is marked and $p = 0 \in C$ and $p > 0 \in C'$. Next, we can now eliminate $r_6 = C \mapsto E$ because it decreases the numerical feature d_g and the only other remaining rule $r_1 = C' \mapsto E'$ arbitrarily changes d_g, but this can only happen finitely many times, since p is marked and $p = 0 \in C$ and $p > 0 \in C'$. Last, we eliminate the remaining rule r_1 because it sets the Boolean feature b to true.

It remains to show that the sketch width is 1. Consider any Driverlog instance with states S. If there are misplaced packages in the initial state, then rule r_3 decrements the number of misplaced packages. Therefore, we show that moving packages to their target location has width 1. Consider states $S_1 \subseteq S$ where there is a misplaced package p at location c_m with target location c_o. We do a three-way case distinction over all states S_1 and show that moving a package to its target location has width 1. First, consider rule r_1. Intuitively, we show that boarding some driver into a truck has width 1. Consider states $S_1^1 \subseteq S_1$ where the feature conditions of rule r_1 are true, i.e., states there is no driver boarded into any truck. With $G_{r_1}(s)$ we denote the subgoal states of r_1 in $s \in S_1^1$, i.e., states where a driver d is boarded into a truck t. The tuple $driving(d, t)$ implies $G_{r_1}(s)$ in $s \in S_1^1$ in admissible chain that consists of moving d from c_1 to c_n, each step decreasing the distance to c_n, boarding d into t, i.e.,

$$(at(d, c_1), \ldots, at(d, c_n), driving(d, t)).$$

Second, consider rule r_2. Intuitively, we show that loading a misplaced package into a truck has width 1. Consider states $S_1^2 \subseteq S_1$ where the feature conditions of rule r_2 are true and where d is boarded into truck t at location l_n, i.e., no misplaced package is loaded, and d is boarded into t at location l_n because boarding has d into t if there is a misplaced package has width 1 (see above). With $G_{r_2}(s)$ we denote the subgoal states of r_2 in $s \in S_1^2$, i.e., states where p is loaded into t. The tuple $in(p, t)$ implies $G_{r_2}(s)$ in $s \in S_1^2$ in the admissible chain that consists of driving t from c_n to c_m, each step decreasing the distance to c_m, loading p into t, i.e.,

$$(at(t, c_n), \dots, at(t, c_m), in(p, t)).$$

Third, consider rule r_3. Intuitively, we show that moving a package to it target location has width 1. Consider states $S_1^3 \subseteq S_1$ where the feature conditions of rule r_3 are true and where p and d is in t at c_m, i.e., states where p and d is in t at c_m because loading driver and misplaced package has width 1 (see above). With $G_{r_3}(s)$ we denote the subgoal states of r_3 in $s \in S_1^3$, i.e., states where p is at location c_o. The tuple $at(p, c_o)$ implies $G_{r_3}(s)$ in the admissible chain that consists of driving t from c_m to c_o, each step decreasing the distance to c_o, and unloading p, i.e.,

$$(at(t, c_m), \dots, at(t, c_o), at(p, c_o)).$$

Now, consider states S_2 where all packages are at their respective target location and there is a misplaced truck t at location l_n with target location l_m. This can either be the case in the initial state or after moving the packages because it requires to use trucks. We do a two-way case distinction over all states S_2 and show that moving a truck to its target location has width 1. Consider rule r_4. Intuitively, we show that boarding a driver into a misplaced truck without using any truck has width 1. Consider states $S_2^1 \subseteq S_2$ where the feature conditions of rule r_4 are true, i.e., where there is a driver d at location c_1 with nonzero distance until being boarded into t. With $G_{r_4}(s)$ we denote the subgoal states of r_4 in $s \in S_2^1$, i.e., states where d is one step closer to being boarded into t. There are three possible admissible chains that must be considered. (1) unboarding d from some truck t', i.e., tuple $at(d, c_1)$ implies $G_{r_4}(s)$ in $s \in S_2^1$ in the admissible chain $(driving(d, t'), at(d, c_1))$, (2) moving d closer to c_n over c_{i-1} to c_i, i.e., tuple $at(d, c_i)$ implies $G_{r_4}(s)$ in $s \in S_2^1$ in the admissible chain $(at(d, c_{i-1}), at(d, c_i))$, and (3) boarding d into t at c_n, i.e., $driving(d, t)$ implies $G_{r_4}(s)$ in $s \in S_2^1$ in the admissible chain $(at(d, c_n), driving(d, t))$. Second, consider rule r_5. Intuitively, we show that moving a misplaced truck to its target location has width 1. Consider states $S_2^2 \subseteq S_2$ where the feature conditions of rule r_5 are true and where some driver is boarded into t, i.e., states where d is boarded intro t at c_n. With $G_{r_5}(s)$ we denote the subgoal states of r_5 in $s \in S_2^2$, i.e., states where t is at its target location. The tuple $at(t, c_n)$ implies $G_{r_5}(s)$ in $s \in S_2^2$ in the admissible chain that consists of moving t from c_n to c_m, each step decreasing the distance to c_m, i.e., $(at(t, c_n), \dots, at(t, c_m))$.

Now, consider states S_3 where all packages and trucks are at their respective target location and there is a misplaced driver d boarded or unboarded at location l_1 with target location l_n. This can either be the case in the initial state or after moving the packages and trucks. Consider rule r_6. Intuitively, we show that moving a driver to its target location without using any truck has width 1. With $G_{r_6}(s)$ we denote the subgoal states of r_6 in $s \in S_3$, i.e., states where d is at its target location. There are two possible admissible chains that must be considered. (1) unboarding d at location c_1, i.e., tuple $at(d, c_1))$ implies $G_{r_6}(s)$ in $s \in S_3$ in the admissible chain $(driving(d, t), at(d, c_1))$, and (2) moving d closer from location c_{i-1} to c_i, i.e., tuple $at(d, c_i)$ implies $G_{r_6}(s)$ in $s \in S_3$ in the admissible chain $(at(d, c_{i-1}), at(d, c_i))$.

We obtain sketch width 1 because all tuples in admissible chains have size 1. Note that when dropping rules r_1 and r_2, as well as features l and b, the sketch width becomes 2 because we must merge the three admissible chains of the first subproblem. When merging, tuples of size 2 must be considered, each consisting of a location and whether the driver drives the truck or whether the package is loaded. □

5.8 Schedule

In the Schedule domain (Bacchus, 2001), there is a set of objects that can have different values for the following attributes: shape, color, surface condition, and temperature. Also, there is a set of machines where each is capable of changing an attribute with the side effect that other attributes change as well. For example, rolling an object changes its shape to cylindrical and has the side effect that the color changes to uncolored, any surface condition is removed, and the object becomes hot. Often, there are multiple different work steps for achieving a specific attribute of an object. For example, both rolling and lathing change an object's shape to cylindrical. But rolling makes the object hot, while lathing keeps its temperature cold. Some work steps are only possible if the object is cold. Multiple work steps can be scheduled to available machines, which sets the machine's status to occupied. All machines become available again after a single do-time-step action. The goal is to change the attributes of objects.

SIW fails in Schedule because it gets trapped into deadends when an object's temperature becomes hot, possibly blocking other required attribute changes. The following sketch uses this observation and defines an ordering over achieved attributes where first, the desired shapes are achieved, second, the desired surface conditions are achieved, and third, the desired colors are achieved.

Consider the set of features $\Phi = \{p_1, p_2, p_3, h, o\}$ where p_1 is the number of objects with the wrong shape, p_2 is the number of objects with the wrong surface condition, p_3 is the number of objects with the wrong color, h is the number of hot objects, and o is true iff there is an object scheduled or a machine occupied. We define the following sketch rules R^S_Φ:

$$\begin{aligned}
r_1 &= \{p_1 > 0\} \mapsto \{p_1\downarrow, p_2?, p_3?, o\}\\
r_2 &= \{p_1 = 0, p_2 > 0\} \mapsto \{p_2\downarrow, p_3?, o\}\\
r_3 &= \{p_1 = 0, p_2 = 0, p_3 > 0\} \mapsto \{p_3\downarrow, o\}\\
r_4 &= \{o\} \mapsto \{\neg o\}
\end{aligned}$$

Rule r_1 says that achieving an object's goal shape is good. Rule r_2 says that achieving an object's goal surface condition is good after achieving all goal shapes. Rule r_3 says that achieving an object's goal color is good after achieving all goal shapes and surface conditions. Rule r_4 says that making objects and machines available is good. Note that r_4 does not decrease the sketch width but it decreases the search time by decreasing the search depth. Note also that h never occurs in any rule because we want its value to remain constant.

Theorem 11. *The sketch R^S_Φ for the Schedule domain is well-formed and has width* 0.[4]

Proof. The features are goal separating because the feature valuations $p_1 = 0, p_2 = 0, p_3 = 0$ hold in state s iff s is a goal state. We show that the sketch is terminating by iteratively eliminating rules: Rule r_1 decreases the numerical feature p_1 that no other remaining rule increments, so we eliminate r_1 and mark p_1. Rule r_2 decreases the numerical feature p_2 that no other remaining rule increments,

4. In previous work (Drexler et al., 2021) we missed the fact that every subproblem is solved in at most one step and hence, the sketch width is 0.

so we eliminate r_2 and mark p_2. Rule r_3 decreases the numerical feature p_3 that no other remaining rule increments, so we eliminate r_3 and mark p_3. Last, we eliminate the only remaining rule r_4 because it sets the Boolean feature o to false.

It remains to show that the sketch width is 0. Consider any Schedule instance with states S. First, consider states $S_1 \subseteq S$ where the feature conditions of r_4 are true, i.e., there is either a scheduled object or a machine occupied. This can be the case in the initial state or if an object is scheduled to be processed by a machine. With $G_{r_4}(s)$ we denote the subgoal states of r_4 in $s \in S_1$, i.e., states where no object is scheduled and no machine is occupied and all objects have the same shape, surface condition, color, and temperature. The action that performs a time step always reaches a subgoal state in $G_{r_4}(s)$ in a single step.

Now, consider states $S_2 \subseteq S$ where the feature conditions of r_1 are true and there is no object scheduled and no occupied machine, i.e., states where there is an object a that has shape x that is not the shape y mentioned in the goal, and there is no object scheduled and no occupied machine because this can be achieved with width 0 (see above). With $G_{r_1}(s)$ we denote the set of subgoal states of r_1 in $s \in S_2$, i.e., states where a has shape y and all objects have the same temperature. The action that changes the shape of an object to its goal shape while not changing the temperature of an object reaches a subgoal state in $G_{r_1}(s)$ in a single step.

Now, consider states $S_3 \subseteq S$ where the feature conditions of r_2 are true, there is no object scheduled and no machine occupied, and objects have their correct shape, i.e., states where there is an object a that has surface x that is not the surface y mentioned in the goal, there is no object scheduled and no occupied machine because this can be achieved with width 0 (see above), and all objects have their correct shape because changing the shape has width 0 (see above). With $G_{r_2}(s)$ we denote the set of subgoal states of r_2 in $s \in S_3$, i.e., states where a has surface y and all objects have the same shape and temperature. The action that changes the surface condition of an object to the goal surface condition while not changing a correct shape or the temperature of an object reaches a subgoal state in $G_{r_2}(s)$ in a single step.

Now, consider states $S_4 \subseteq S$ where the feature conditions of r_3 are true, there is no object scheduled, no machine occupied, and all objects have their correct shape and surface, i.e., states where there is an object a that has color x that is not the color y mentioned in the goal, there is no object scheduled and no occupied machine because this can be achieved with width 0 (see above), all objects have their correct shape because changing the shape has width 0 (see above), and all objects have their desired surface because changing the surface has width 0 (see above). With $G_{r_3}(s)$ we denote the set of subgoal states of r_3 in $s \in S_3$, i.e., states where a has color y and all objects have the same shape, surface condition and temperature. The action that changes the color of an object to the goal color while not changing the shape, surface condition or the temperature of an object reaches a subgoal state in $G_{r_3}(s)$ in a single step. Note that r_3 achieves the goal when the color of the last object changes to the color mentioned in the goal.

We obtain sketch width 0 because each subproblem is solvable in a single step. $\square$

6. Experiments

Even though the focus of our work is on proving polynomial runtime bounds for planning domains theoretically, we evaluate in this section how these runtime guarantees translate into practice. We implemented two versions of SIW_R: one version, denoted by $\text{SIW}_\text{R}^\text{G}$, is based on the LAPKT planning system (Ramirez, Lipovetzky, & Muise, 2015) and grounds the input task to a propositional representation before the search, the other version, denoted by $\mathrm{SIW}_{\mathrm{R}}^{\mathrm{L}}$, is implemented in the Mimir planning system (Ståhlberg, 2023) and searches on the lifted task representation directly. Both versions use the DLPlan library (Drexler, Francès, & Seipp, 2022) to represent and evaluate features (see the appendix for details). We use the Lab toolkit (Seipp, Pommerening, Sievers, & Helmert, 2017) for running experiments on Intel Xeon Gold 6130 CPU cores. For each planner run, we limit time and memory by 30 minutes and 3 GiB.

We collected benchmark tasks for the domains analyzed above from two different sources. The first source is the satisficing track of previous IPCs. Since many of these tasks are trivial for state-of-the-art planners, we also consider tasks from the new Autoscale benchmark set (Torralba, Seipp, & Sievers, 2021). The Autoscale benchmark set is optimized to contain tasks where current state-of-the-art planners show differences in coverage.[5] The Grid$_*$ tasks in the Autoscale benchmark set are different from the Grid tasks in the IPC benchmark set. The difference is that in Grid$_*$ tasks, it is not always possible to reach every lock for the unlock operation. Hence, rule r_3 can define picking a wrong key, increasing the width of these subproblems to 2. However, a sketch of width 1 can be obtained by picking only keys for which there exists a locked door that is reachable, which can be computed by using transitive closure on the connectivity relation. As discussed in the section about the Grid domain, dropping a key at its target location by exchanging it with a key at the current location increases the width of a subproblem from 1 to 2. Therefore, for the domain Grid (resp. Grid$_*$), we also include a simplified domain Grid$^{\mathrm{S}}$ (resp. Grid$_*^{\mathrm{S}}$) where we removed the action to exchange the key that is being held with a key at the current location.

The main question we want to answer empirically is how much an SIW search benefits from using policy sketches. To this end, we compare SIW(2) to $\mathrm{SIW}_{\mathrm{R}}(2)$, which uses the sketches presented above. We use a width bound of $k = 2$ because SIW(k) and $\mathrm{SIW}_{\mathrm{R}}(k)$ are too slow to compute in practice for larger values of k. We also include two well-known, state-of-the-art planners, LAMA (Richter & Westphal, 2010) and Dual-BFWS (Lipovetzky & Geffner, 2017a), to show that the considered planning tasks are hard to solve for the strongest domain-independent planners. However, since $\mathrm{SIW}_{\mathrm{R}}(2)$ is a domain-dependent planner, we cannot directly compare it to the domain-independent approaches. The code and data are available online (Drexler, Seipp, & Geffner, 2023a).

Table 1 shows results for the five planners. In addition to the number of solved tasks and planner runtimes, which we discuss below, for the planners based on SIW, the table also holds data about the *effective* width. The effective width for a problem P and one of SIW(k), $\mathrm{SIW}_{\mathrm{R}}^{\mathrm{G}}(k)$, or $\mathrm{SIW}_{\mathrm{R}}^{\mathrm{L}}(k)$ is the smallest natural number k the algorithm needs to solve P. The effective width can be smaller than the actual width of the problem and depends on the order in which a specific implementation of the SIW-based algorithms generates successor states. For more robust and comparable results, we always randomly shuffle the applicable actions of a state before generating its successor states. Since an $\mathrm{SIW}_{\mathrm{R}}(k)$ search splits a problem into subproblems, we further distinguish between the maximum effective width (M) among all subproblems and the average effective width (A) over all subproblems. We see that the maximum effective width for $\mathrm{SIW}_{\mathrm{R}}(2)$ equals the theoretical upper bounds established in the previous section, suggesting that the bounds are indeed tight. We can see that the maximum effective width of $\mathrm{SIW}_{\mathrm{R}}^{\mathrm{L}}(2)$ in Grid (2) is larger than the maximum effective width of the same algorithm in Grid$^{\mathrm{S}}$ (1). In one subproblem in Grid, the exchange key action is applied before the drop key action, resulting in the pruning of the actual subgoal state where the key

5. The Autoscale benchmark set does not contain tasks from the Schedule domain.

		LAMA		BFWS		SIW(2)			$\mathrm{SIW}_{\mathrm{R}}^{\mathrm{G}}(2)$				$\mathrm{SIW}_{\mathrm{R}}^{\mathrm{L}}(2)$			
	Domain	S	T	S	T	S	A	M	S	T	A	M	S	T	A	M
IPC	Barman (40)	**40**	760	**40**	248	0	–	–	**40**	3	0.8	2	**40**	8	0.9	2
	Childsnack (20)	6	3	9	172	0	–	–	**20**	2	0.6	1	**20**	5	0.6	1
	Driverlog (20)	**20**	39	**20**	3	7	1.5	2	**20**	4	0.4	1	**20**	9	0.4	1
	Floortile (40)	9	202	6	865	0	–	–	**40**	1	1.2	2	**40**	1	1.2	2
	Grid (5)	**5**	3	**5**	3	2	2.0	2	**5**	2	0.8	2	**5**	1	0.9	2
	$\mathrm{Grid}^{\mathrm{S}}$ (5)	**5**	2	**5**	1	1	2.0	2	**5**	2	0.8	1	**5**	1	0.8	1
	Schedule (150)	**150**	38	**150**	103	78	1.1	2	**150**	13	0.0	0	–	–	–	–
	TPP (30)	**30**	12	**30**	1234	21	2.0	2	**30**	8	0.2	1	**30**	3	0.2	1
Autoscale	Barman (30)	25	4	5	1468	0	–	–	**30**	8	0.9	2	26	25	1.0	2
	Childsnack (30)	9	2	5	714	0	–	–	**30**	1	0.6	1	**30**	2	0.6	1
	Driverlog (30)	**30**	1194	**30**	691	1	1.9	2	**30**	461	0.6	1	**30**	417	0.6	1
	Floortile (30)	2	45	2	854	1	1.3	2	15	1	1.2	2	**17**	1	1.2	2
	Grid_* (30)	**17**	5	9	15	4	1.9	2	9	34	0.9	2	6	270	0.9	2
	$\mathrm{Grid}_*^{\mathrm{S}}$ (30)	9	4	9	37	4	1.9	2	**10**	37	0.9	2	6	1776	0.9	2
	TPP (30)	25	154	11	1406	11	2.0	2	21	64	0.3	1	**30**	4	0.2	1

Table 1: Comparison of the first iteration of LAMA, Dual-BFWS (BFWS), SIW(2), $\mathrm{SIW}_{\mathrm{R}}^{\mathrm{G}}(2)$, and $\mathrm{SIW}_{\mathrm{R}}^{\mathrm{L}}(2)$. The table shows the number of solved tasks (S), the maximum runtime (T) in seconds for tasks commonly solved by LAMA, Dual-BFWS, $\mathrm{SIW}_{\mathrm{R}}^{\mathrm{G}}(2)$, and $\mathrm{SIW}_{\mathrm{R}}^{\mathrm{L}}(2)$, and the average (A) and maximum (M) effective width over all encountered subtasks. The top and bottom parts show results for the IPC and Autoscale benchmark sets, respectively. We highlight the maximum number of solved tasks (S) per domain in boldface.

is at its target location. Inspecting the average effective width for $\mathrm{SIW}_{\mathrm{R}}(2)$, we see that the value is always closer to 1 than to 2 for the domains with sketch width 2. The sketch for the Schedule domain is a general policy where every subproblem is solved in a single step.

The original SIW(2) planner (without sketches) solves very few tasks across the board. In both the IPC and the Autoscale set, there are four domains where SIW(2) solves at most a single task. These results confirm that in many domains, the problem width is too large for plain SIW. In contrast, the sketches allow $\mathrm{SIW}_{\mathrm{R}}(2)$ to solve all IPC tasks with an exception in Schedule where $\mathrm{SIW}_{\mathrm{R}}^{\mathrm{L}}(2)$ does not support quantified preconditions that are used in the domain description.

On the harder Autoscale benchmark set, we observe that $\mathrm{SIW}_{\mathrm{R}}^{\mathrm{G}}(2)$ solves three domains completely (Barman, Childsnack, Driverlog), has a coverage similar to $\mathrm{SIW}_{\mathrm{R}}^{\mathrm{L}}$ in Floortile, and a lower coverage than LAMA in Grid_*. There are two reasons why $\mathrm{SIW}_{\mathrm{R}}^{\mathrm{G}}(2)$ solves fewer Grid_* tasks than LAMA, both related to memory usage. First, $\mathrm{SIW}_{\mathrm{R}}^{\mathrm{G}}(2)$ runs out of memory while trying to initialize the novelty table for width 2 in six tasks. Second, $\mathrm{SIW}_{\mathrm{R}}^{\mathrm{G}}(2)$, which is implemented in LAPKT, fails to ground all other remaining tasks because it uses more memory for representing the ground propositional task compared to LAMA. In TPP, $\mathrm{SIW}_{\mathrm{R}}^{\mathrm{G}}(2)$ solves all tasks where there is sufficient memory to compute the ground tasks and fails to ground all other remaining tasks. In Floortile, $\mathrm{SIW}_{\mathrm{R}}^{\mathrm{G}}(2)$ fails because the search requires too much time.

Our lifted planner $\text{SIW}_\text{R}^\text{L}$ runs out of memory in only the four most difficult Grid$_*$ and Grid$_*^\text{S}$ tasks and runs out of time in all other unsolved tasks. It significantly outperforms all other planners in TPP, solving the most difficult task in only 53 seconds. In Grid$_*$, the successor generation is much slower compared to the grounded version $\text{SIW}_\text{R}^\text{G}$. $\text{SIW}_\text{R}^\text{L}$ typically uses much less memory compared to $\text{SIW}_\text{R}^\text{G}$ because the number of reached atoms grows dynamically during search and is usually much smaller compared to the number of ground atoms.

Overall, our results show that our sketch rules capture useful information and that adding this domain-specific knowledge to a width-based planner allows it to solve whole problem domains very efficiently.

7. Related Work

We first review other approaches for expressing domain control knowledge for planning and then discuss some related work on polynomial planning algorithms and subgoal decomposition for domain-independent classical planning. The distinction between actions that are "good" or "bad" in a fixed tractable domain can often be characterized explicitly. Indeed, **general policies**, unlike sketches, can provide such a classification of all possible state transitions (s, s') over the problems in $\mathcal{Q}$. Doing so, they ensure that the goal can always be reached by following any good transition (Bonet & Geffner, 2018; Bonet et al., 2019; Francès et al., 2021). Francès, Corrêa, Geissmann, and Pommerening (2019) use the same type of description logic features (Baader et al., 2003) to define and learn general policies in terms of linear value functions. Sketch rules have the same syntax as policy rules, but instead of constraining state transitions, they define subgoals.

Logical approaches to domain control have been used to provide partial information about good and bad state transitions in terms of suitable formulas (Bacchus & Kabanza, 2000; Kvarnström & Doherty, 2000). For example, for the Schedule domain, one may have a formula in **linear temporal logic** (LTL) expressing that objects that need to be lathed and painted should not be painted in the next time step, since lathing removes the paint. This partial information about good and bad transitions can then be used by a forward-state search planner to heavily prune the state space. A key difference between these formulas and sketches is that sketch rules are not about state transitions but about subgoals, and hence they structure the search for plans in a different way, in certain cases ensuring a polynomial search.

Baier, Fritz, Bienvenu, and McIlraith (2008) combine control knowledge and preference formulas to improve search behavior and obtain plans of high quality, according to user preferences. The control knowledge is given in the **Golog** language (Levesque, Reiter, Lespérance, Lin, & Scherl, 1997) and defines subgoals such that a planner has to fill in the missing parts. Since the control knowledge is compiled directly to PDDL, they are able to leverage off-the-shelve planners. The user preferences are encoded in an LTL-like language. Like our policy sketches, their approach can be applied to any domain. However, policy sketches aim at ensuring polynomial searches in tractable domains.

Hierarchical task networks or **HTNs** are used mainly for expressing general top-down strategies for solving classes of planning problems (Erol, Hendler, & Nau, 1994; Nau, Au, Ilghami, Kuter, Murdock, Wu, & Yaman, 2003; Georgievski & Aiello, 2015). The domain knowledge is normally expressed in terms of a hierarchy of methods that have to be decomposed into primitive methods that cannot be decomposed any further. While the solution strategy expressed in HTNs does not have to be complete, it is often close to complete, as otherwise the search for decompositions easily becomes intractable. For this reason, crafting good and effective HTNs encodings is not easy. For example, the HTN formulation of the Barman domain in the 2020 Hierarchical Planning Competition (Höller, Behnke, Bercher, Biundo, Fiorino, Pellier, & Alford, 2019) contains 10 high-level tasks (like *AchieveContainsShakerIngredient*), 11 primitive tasks (like *clean-shot*) and 22 methods (like *MakeAndPourCocktail*). In contrast, the PDDL version of Barman has only 12 action schemas, and the sketch above has 4 rules over 4 linear features. Note, however, that comparing different forms of control knowledge in terms of their compactness is not well-defined.

Hierarchical Goal Networks, **HGNs**, are a hybrid approach between classical planning and HTNs (Shivashankar, Kuter, Nau, & Alford, 2012). So-called *HGN methods* are similar to actions in classical planning, but with an additional set of subgoals and a goal network that encodes a partially ordered sequence of goals. HGN methods are an alternative way to define PDDL actions, while sketches work directly on top of the PDDL planning formalism. HGNs, unlike HTNs, use a planning mechanism where ground methods are selected based on the current goals and state of the system, similar to sketches.

The need to represent the common subgoal structure of dynamic domains arises also in reinforcement learning (RL), where knowledge gained in the solution of some domain instances can be applied to speed up the learning of solutions to new instances of the same family of tasks (Finn, Abbeel, & Levine, 2017). In recent work in deep RL (DRL) these representations, in the form of general **intrinsic reward functions** (Singh, Lewis, Barto, & Sorg, 2010), are expected to be learned from suitable DRL architectures (Zheng, Oh, Hessel, Xu, Kroiss, van Hasselt, Silver, & Singh, 2020). Sketches provide a convenient high-level alternative to describe common subgoal structures, but opposed to the related work in DRL, the policy sketches above are not learned but are written by hand. We describe the challenge of automatically learning sketches briefly below.

Temporal abstraction is another method from reinforcement learning that addresses the problem of reward sparsity by decomposing tasks (Sutton, Precup, & Singh, 1999). Temporal abstractions consider a set of high-level macro actions in the form of *options*. Each option consists of a dedicated policy, reward function and termination criterion. In the options framework, an RL agent chooses one of its options based on its current state and executes the option's policy until it terminates. The policy learning happens at two levels: each option policy is learned individually on the low level and the high-level controller learns which option to select in which state. Recently, there have been several works on defining *symbolic* options, allowing the RL agent to use reasoning instead of learning for finding (partially-ordered) plans over the set of options (Illanes, Yan, Icarte, & McIlraith, 2020; Lee, Katz, Agravante, Liu, Klinger, Campbell, Sohrabi, & Tesauro, 2021; Jin, Ma, Jin, Zhuo, Chen, & Yu, 2022). These approaches are very similar in spirit to policy sketches and future research could even define options based on sketch rules.

Approaches based on temporal abstraction, such as the options framework or angelic hierarchical planning (Marthi, Russell, & Wolfe, 2008), use high-level actions to abstract away primitive actions, thereby reducing the size of the state space. Another way to simplify the state space is to use **state abstraction**, where multiple states are grouped into a single abstract state (e.g., Holte, Perez, Zimmer, & MacDonald, 1996). Policy sketches combine both types of abstraction: they use state abstraction by considering the feature valuation space and they use temporal abstraction since the subgoals are usually several steps away. In contrast to sketches, general policies only use state abstraction, but not temporal abstraction, because they operate directly on primitive actions.

Another approach for decomposing reinforcement learning tasks are **reward machines** (Icarte, Klassen, Valenzano, & McIlraith, 2022). A reward machine is a finite state machine that represents a coarse version of the underlying RL task. Each transition is labeled with a conjunction over a set of propositions. To illustrate the concept, assume that we have a reward machine with two states s and s' and a transition from s to s' labeled with conjunction c. When the reward machine is in state s and the RL agent observes a situation where c holds, the reward machine transitions into state s' and yields a reward function that is deemed useful for the agent in the new subproblem captured by s'. Even with policy sketches that only use Boolean features it is straightforward to capture the task decomposition of any reward machine. The main difference between the two approaches is that reward machines are defined solely via the transition labels and they consider their states as black boxes, whereas the rule conditions and effects for policy sketches are *observable*, i.e., amenable for reasoning by the planning algorithm that uses the sketch. Policy sketches are more general than reward machines since they can also use numerical features, allowing them to reason about quantitative change between states in addition to qualitative differences.

Hoffmann (2005) analyzes the **local search topology** of the optimal delete-relaxed heuristic h^+ and shows that enforced hill climbing using h^+ runs in polynomial time for many IPC domains. Since the FF heuristic h^{FF} (Hoffmann & Nebel, 2001) often closely approximates h^+, his findings explain the strong performance of the FF planner, which uses enforced hill climbing with h^{FF} (followed by a greedy best-first search using h^{FF}). Enforced hill climbing repeatedly runs a breadth-first search to find the next state with a lower heuristic value, which is similar to the breadth-first searches done by SIW and SIW_R. A difference is that in the former case the breadth-first search is exponential in the search depth, while in the latter case it is exponential only in the width. Usually, the width is much smaller than the search depth required to escape a heuristic plateau or local minimum.

Seipp, Pommerening, Röger, and Helmert (2016) point to shortcomings of the notion of width in planning domains with conjunctive goals, and introduce the **correlation complexity** measure that is given by the maximum size of the Boolean features needed in linear heuristic functions, called *potential heuristics*, to lead greedily to the goal. The Boolean features in that setting are conjunctions of facts in the planning problem. The authors show that many domains have a bounded and small correlation complexity, which however, unlike the notion of width, does not bound the complexity of the instances.

Subgoals have also been studied in the context of domain-independent planning. Porteous, Sebastia, and Hoffmann (2001) introduce **landmarks** as a method for decomposing problems into subproblems and use them within an incomplete hierarchical search algorithm. A way to use landmarks efficiently within a complete search algorithm was developed in the LAMA planner (Richter & Westphal, 2010) that runs a greedy best-first search with multiple queues, some ordered by goal-distance estimation heuristics like h^{FF} and others by a landmark counting heuristic.

8. Conclusions and Future Work

We have shown that the language of policy sketches as introduced by Bonet and Geffner provides a simple, elegant, and powerful way for expressing the common subgoal structure of many planning domains. The SIW_R algorithm can then solve these domains effectively, in provable polynomial time, where SIW fails either because the problems are not easily serializable in terms of the top goals or because some of the resulting subproblems have a high width. A big advantage of pure width-based algorithms like SIW and SIW_R is that unlike other planning-based methods they can be used to plan with simulators in which the structure of states is available but the structure of actions is not.[6]

While all sketches in this paper are designed by hand, we have shown in follow-up work to the original conference paper that it is possible to **learn sketches automatically** (Drexler et al., 2022). Our method for learning policy sketches is related to the method for learning general policies by Francès et al. (2021) which uses the state language (primitive PDDL predicates) to define a large pool of Boolean and numerical features via a description logic grammar (Baader et al., 2003). As shown in the appendix, all features used in the sketches above can be generated in this way. A longer-term challenge is to learn the sketches automatically when using the same inputs as DRL algorithms, where there is no state representation language. Recent works that learn first-order symbolic languages from black-box states or from states represented by images (Asai, 2019; Bonet & Geffner, 2020a; Rodriguez, Bonet, Romero, & Geffner, 2021) are important steps in that direction.

Acknowledgments

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 National Academic Infrastructure for Supercomputing in Sweden (NAISS) and the Swedish National Infrastructure for Computing (SNIC) at the National Supercomputer Centre at Linköping University partially funded by the Swedish Research Council through grant agreements no. 2022-06725 and no. 2018-05973.

Appendix A. Feature Definitions and Grammar

In this section, we describe how we represent and evaluate the features used in the sketches above. Following Francès et al. (2021), we define the features in terms of a grammar based on the predicates of each planning domain and description logics (Baader et al., 2003). Description logics are based on the notion of *concepts*, i.e., sets of objects that share some characteristic, and *roles*, i.e., relations between pairs of objects. We have published the code for constructing and evaluating such features in the form of a new C++ library (with Python bindings), called *DLPlan* (Drexler et al., 2022).

A.1 Syntax and Semantics

The following definition of syntax and semantics is based on the one given by Francès et al. (2021). As in their work, we start from a set of primitive concepts and roles: the unary and binary predicates in the planning domain. We extend their definition slightly to handle domains with predicates of higher arity by including primitive concepts and roles obtained from projections of predicates of the planning domain. Another difference to the work by Francès et al. (2021) is that we obtain the features by hand instead of generating them exhaustively until reaching a given bound in feature complexity.

6. A minor difference then is that the novelty tests in IW(k) are not exponential in $k-1$ but in k.

Consider concepts C, D and roles R, S and let the universe Δ be the set of all objects in the planning task. The set of concepts and roles for a state s are inductively defined as:

- The *primitive concept* $p[i]$ has denotation $(p[i])^s = \{c_i \mid p(c_0, \ldots, c_i, \ldots, c_{n-1}) \in s\}$ where $p(c_0, \ldots, c_i, \ldots, c_{n-1})$ is a ground atom of predicate p with arity n and objects $c_0, \ldots, c_{n-1}$.
- The *primitive role* $p[i, j]$ has denotation $(p[i, j])^s = \{(c_i, c_j) \mid p(c_0, \ldots, c_i, \ldots, c_j, \ldots, c_{n-1}) \in s\}$ where $p(c_0, \ldots, c_i, \ldots, c_j, \ldots, c_{n-1})$ is a ground atom of predicate p with arity n and objects $c_0, \ldots, c_{n-1}$.
- The *universal concept* $\top$ and the bottom concept $\bot$ are concepts with denotations $\top^s = \Delta$ and $\bot^s = \emptyset$.
- The *union* $C \sqcup D$, *intersection* $C \sqcap D$ and *negation* $\neg C$ are concepts with denotations $(C \sqcup D)^s = C^s \cup D^s$, $(C \sqcap D)^s = C^s \cap D^s$ and $(\neg C)^s = \Delta \setminus C^s$.
- The *existential abstraction* $\exists R.C$ and the *universal abstraction* $\forall R.C$ are concepts with denotations $(\exists R.C)^s = \{a \in \Delta \mid \exists b : (a, b) \in R^s \land b \in C^s\}$, $(\forall R.C)^s = \{a \in \Delta \mid \forall b : (a, b) \in R^s \rightarrow b \in C^s\}$.
- If a is a constant in the planning domain, the *nominal* a is a concept with denotation $a^s = \{a\}$.
- The *union* $R \sqcup S$, *intersection* $R \sqcap S$ and *complement* $\neg R$ are roles with denotations $(R \sqcup S)^s = R^s \cup S^s$, $(R \sqcap S)^s = R^s \cap S^s$ and $(\neg R)^s = (\Delta \times \Delta) \setminus R^s$.
- The *role-value maps* $R = S$ and $R \subseteq S$ are concepts with denotations $(R = S)^s = \{a \in \Delta \mid \forall b : (a, b) \in R^s \leftrightarrow (a, b) \in S^s\}$ and $(R \subseteq S)^s = \{a \in \Delta \mid \forall b : (a, b) \in R^s \rightarrow (a, b) \in S^s\}$.
- The *composition* $R \circ S$ is a role with denotation $(R \circ S)^s = \{(a, c) \in \Delta \times \Delta \mid (a, b) \in R^s \land (b, c) \in S^s\}$.
- The *inverse* R^{-1} is a role with denotation $(R^{-1})^s = \{(b, a) \in \Delta \times \Delta \mid (a, b) \in R^s\}$.
- The *transitive closure* R^+ and the *reflexive-transitive closure* R^* are roles with denotations $(R^+)^s = \bigcup_{n \geq 1} (R^s)^n$ and $(R^*)^s = \bigcup_{n \geq 0} (R^s)^n$, where the iterated composition is defined as $(R^s)^0 = \{(d, d) \mid d \in \Delta\}$ and $(R^s)^{n+1} = (R^s)^n \circ R^s$.
- The *restriction* $R|_C$ is a role with denotation $(R|_C)^s = R^s \sqcap (\Delta \times C^s)$.
- The *identity* $id(C)$ is a role with denotation $(id(C))^s = \{(a, a) \mid a \in C^s\}$.
- The (concept) *difference* $C \setminus D$ is a concept with denotation $(C \setminus D)^s = C^s \sqcap (\neg D)^s$.
- The (role) *difference* $R \setminus S$ is a role with denotation $(R \setminus S)^s = R^s \sqcap (\neg S)^s$.
- The *extraction* $R[i]$ with $i \in \{0, 1\}$ is a concept with denotation $(R[0])^s = \exists R^s.\top^s$ and $(R[1])^s = \exists (R^s)^{-1}.\top^s$. [7]

7. Concept difference, role difference, and extraction do not increase expressiveness but make expressing some features more convenient.

Furthermore, for each primitive concept C and primitive role R we allow for corresponding goal versions denoted by C_g and R_g that are evaluated in the goal of the planning instance instead of the state s, as described by Francès et al. (2021).

A.2 From Concepts and Roles to Features

We define Boolean and numerical features with an additional level of composition as follows. Consider concepts C, D, roles R, S, T, and X being either a role or a concept. The set of possible Boolean and numerical features for each state s are defined as:

- $\mathit{Empty}(X)$ is a Boolean feature that evaluates to true iff $|X^s| = 0$.
- $\mathit{Count}(X)$ is a numerical feature that evaluates to $|X^s|$.
- $\mathit{ConceptDist}(C, R, D)$ is a numerical feature that evaluates to the smallest $n \in \mathbb{N}_0$ s.t. there are objects $x_0, \dots, x_n$ with $x_0 \in C^s$, $x_n \in D^s$, and $(x_i, x_{i+1}) \in R^s$ for $i = 0, \dots, n-1$. If no such n exists then the feature evaluates to ∞.
- $\mathit{RoleDist}(R, S, T)$ is a numerical feature that evaluates to the smallest $n \in \mathbb{N}_0$ s.t. there are objects $x_0, \dots, x_n$, there exists some $(a, x_0) \in R^s$, $(a, x_n) \in T^s$, and $(x_i, x_{i+1}) \in S^s$ for $i = 0, \dots, n-1$. If no such n exists, the feature evaluates to ∞.
- $\mathit{SumRoleDist}(R, S, T)$ is a numerical feature that evaluates to $\sum_{r \in R^s} \mathit{RoleDist}(\{r\}, S, T)$, where the sum evaluates to ∞ if any term is ∞.

A.3 Floortile

The domain defines the following predicates: *available-color(c: color), clear(x: tile), down(x: tile, y: tile), left(x: tile, y: tile), painted(x: tile, c: color), right(x: tile, y: tile), robot-at(r: robot, x: tile), robot-has(r: robot, c: color), up(x: tile, y: tile).*

Consider the following concepts and roles:

$$\begin{aligned}
x_1 &\equiv (\mathit{painted}_g[0,1] \setminus \mathit{painted}[0,1])[0] \\
x_2 &\equiv (\mathit{left}[0] \sqcup \mathit{left}[1]) \setminus \mathit{painted}_g[0]) \\
x_3 &\equiv \mathit{up}[0,1] \sqcup \mathit{down}[0,1] \sqcup \mathit{id}(\mathit{left}[0] \sqcup \mathit{left}[1]) \\
x_4 &\equiv ((x_3|_{x_1})^{-1}|_{x_1})^{-1} \\
x_5 &\equiv ((x_3|_{x_2})^{-1}|_{x_1})^{-1}
\end{aligned}$$

Concept x_1 is the set of all unpainted tiles. Concept x_2 is the set of all normal tiles that must not be painted. Role x_3 is the set of pairs of tiles (t, t') where t is above or below of t' and the identity $t = t'$. Role x_4 is the set of pairs of tiles (t, t') where t is above or below of t' and both are unpainted. Role x_5 is the set of pairs of tiles (t, t') where t is unpainted and above or below of normal tile t'. The features $\Phi = \{v, g\}$ used in the sketch for Floortile are constructed as follows:

$$\begin{aligned}
v &= \mathit{Empty}(x_1 \setminus ((x_4)^* \circ x_5)[0]) \\
g &= \mathit{Count}(x_1)
\end{aligned}$$

A.4 TPP

The domain defines the following predicates: *at(t: truck, p: place), connected(p1: place, p2: place), loaded(g: goods, t: truck, l: level), next(l1: level, l2: level), on-sale(g: goods, m: market, l: level), ready-to-load(g: goods, m: market, l: level), stored(g: goods, l: level).*

Consider the following concepts and roles:

$$\begin{aligned} x_1 &\equiv (\mathit{stored}_g[0,1] \setminus \mathit{stored}[0,1])[0] \\ x_2 &\equiv \mathit{next}[0] \end{aligned}$$

Concept x_1 is the set of goods of which some quantity remains to be stored. Concept x_2 is the set of nonempty levels. The features $\Phi = \{b, l, n\}$ used in the sketch for TPP are constructed as follows:

$$\begin{aligned} b &\equiv \mathit{Count}(x_1 \setminus \exists \mathit{ready\text{-}to\text{-}load}[0,2].x_2) \\ l &\equiv \mathit{Count}(x_1 \setminus \exists \mathit{loaded}[0,2].x_2) \\ n &\equiv \mathit{SumRoleDist}(\mathit{stored}_g[0,1], \mathit{next}[0,1], \mathit{stored}[0,1]) \end{aligned}$$

A.5 Barman

The domain defines the following predicates: *clean(c: container), cocktail-part1(c: cocktail, i: ingredient), cocktail-part2(c: cocktail, i: ingredient), contains(c: container, b: beverage), dispenses(d: dispenser, i: ingredient), empty(c: container), handempty(h: hand), holding(h: hand, c: container), next(l1: level, l2: level), ontable(c: container), shaked(s: shaker), shaker-empty-level(s: shaker, l: level), shaker-level(s: shaker, l: level), unshaked(s: shaker), used(c: container, b: beverage).*

Consider the following concepts and roles:

$$\begin{aligned} x_1 &\equiv (\mathit{contains}_g[0,1] \setminus \mathit{contains}[0,1]) \\ x_2 &\equiv (\mathit{contains}_g[0,1] \sqcap \mathit{contains}[0,1]) \\ x_3 &\equiv \exists \mathit{cocktail\text{-}part1}[0,1].\exists \mathit{contains}[1,0].\mathit{shaker\text{-}level}[0] \\ x_4 &\equiv \exists \mathit{cocktail\text{-}part2}[0,1].\exists \mathit{contains}[1,0].\mathit{shaker\text{-}level}[0] \end{aligned}$$

Role x_1 is the set of unserved beverages paired with the corresponding shot that must be used. Role x_2 is the set of served beverages paired with the corresponding shot that was used. Concept x_3 is the set of cocktails where the first ingredient mentioned in its recipe is in the shaker. Concept x_4 is the set of cocktails where the second ingredient mentioned in its recipe is in the shaker. The features $\Phi = \{g, u, c_1, c_2\}$ used in the sketch for Barman are constructed as follows:

$$\begin{aligned} g &\equiv \mathit{Count}(x_1) \\ u &\equiv \mathit{Count}(\mathit{used}[0] \setminus x_2[0]) \\ c_1 &\equiv \neg \mathit{Empty}(x_3 \sqcap x_1[1]) \\ c_2 &\equiv \neg \mathit{Empty}(x_3 \sqcap x_4 \sqcap x_1[1]) \end{aligned}$$

A.6 Grid

The domain defines the following predicates: *arm-empty(), at(r: object, x: object), at-robot(x: object), conn(x: object, y: object), holding(k: object), key(k: object), key-shape(k: object, s: ob-*

ject), lock-shape(x: object, s: object), locked(x: object), open(x: object), place(p: object), shape(s: object).

Consider the following concepts and roles:

$$x_1 \equiv at_g[0,1] \setminus at[0,1]$$
$$x_2 \equiv \exists key\text{-}shape[0,1].\exists lock\text{-}shape[1,0].locked[0]$$

Role x_1 is the set of misplaced key paired with their respective target location. Concept x_2 is the set of keys for which a closed lock with the same shape as the key exists. The features $\Phi = \{l, k, o, t\}$ used in the sketch for Grid are constructed as follows:

$$l \equiv Count(locked[0])$$
$$k \equiv Count(x_1)$$
$$o \equiv \neg Empty(holding[0] \sqcap x_2)$$
$$t \equiv \neg Empty(holding[0] \sqcap x_1[0])$$

A.7 Childsnack

The domain defines the following predicates: *allergic-gluten(c: child), at(t: tray, p: place), at-kitchen-bread(b: bread-portion), at-kitchen-content(c: content-portion), at-kitchen-sandwich(s: sandwich), no-gluten-bread(b: bread-portion), no-gluten-content(c: content-portion), no-gluten-sandwich(s: sandwich), not-allergic-gluten(c: child), notexist(s: sandwich), ontray(s: sandwich, t: tray), served(c: child), waiting(c: child, p: place).*

Consider the following concepts and roles:

$$x_1 \equiv served_g[0] \setminus served[0]$$
$$x_2 \equiv no\text{-}gluten\text{-}sandwich[0]$$

Concept x_1 is the set of unserved children. Concept x_2 is the set of gluten-free sandwiches. The features $\Phi = \{c_g, c_r, s_g^k, s^k, s_g^t, s^t\}$ used in the sketch for Childsnack are constructed as follows:

$$c_g \equiv Count(allergic\text{-}gluten[0] \sqcap x_1)$$
$$c_r \equiv Count(not\text{-}allergic\text{-}gluten[0] \sqcap x_1)$$
$$s_g^k \equiv \neg Empty(at\text{-}kitchen\text{-}sandwich[0] \sqcap x_2)$$
$$s^k \equiv \neg Empty(at\text{-}kitchen\text{-}sandwich[0])$$
$$s_g^t \equiv \neg Empty(ontray[0] \sqcap x_2)$$
$$s^t \equiv \neg Empty(ontray[0])$$

A.8 Driverlog

The domain defines the following predicates: *at(obj: object, loc: object), driver(d: object), driving(d: object, v: object), empty(v: object), in(obj1: object, obj: object), link(x: object, y: object), location(loc: object), obj(obj: object), path(x: object, y: object), truck(truck: object).*

Consider the following concepts and roles:

$$\begin{aligned}
x_1 &\equiv (at_g[0,1] \setminus at[0,1]) \\
x_2 &\equiv (at[1,0] \sqcup \mathit{driving}[1,0])|_{at_g[0] \sqcap \mathit{driver}[0]})^{-1} \\
x_3 &\equiv (\mathit{driving}[1] \sqcup (at[1,0]|_{\mathit{driver}[0]})[0]) \\
x_4 &\equiv at[0,1] \sqcup at[1,0] \sqcup \mathit{path}[0,1]
\end{aligned}$$

Role x_1 is the set of misplaced packages, drivers and trucks, paired with their respective target location. Role x_2 is the set of misplaced drivers paired with their current location. Concept x_3 is the set of trucks with a boarded driver and the location of unboarded drivers. Role x_4 is the set of pairs of trucks and locations reachable by a single unboard, board or walk action. The features $\Phi = \{p, t, d_g, d_t, b, l\}$ used in the sketch for Driverlog are constructed as follows:

$$\begin{aligned}
p &\equiv \mathit{Count}(\mathit{obj}[0] \sqcap x_1[0]) \\
t &\equiv \mathit{Count}(\mathit{truck}[0] \sqcap x_1[0]) \\
d_g &\equiv \mathit{SumRoleDist}(x_2, x_4, at_g[0,1]) \\
d_t &\equiv \mathit{ConceptDist}(x_3, x_4, \mathit{truck}[0] \sqcap x_1[0]) \\
b &\equiv \neg \mathit{Empty}(\mathit{driving}[0]) \\
l &\equiv \neg \mathit{Empty}(\mathit{obj}[0] \sqcap \mathit{in}[0] \sqcap at_g[0])
\end{aligned}$$

A.9 Schedule

The domain defines the following predicates: *busy(machine: machine), can-orient(machine: machine, orientation: anorient), has-bit(machine: machine, width: width), has-hole(obj: part, width: width, orientation: anorient), has-paint(machine: machine, colour: colour), objscheduled(), painted(obj: part, colour: colour), scheduled(obj: part), shape(obj: part, shape: ashape), surface-condition(obj: part, surface-cond: surface), temperature(obj: part, temp: temperature-type).*

The features $\Phi = \{p_1, p_2, p_3, h, o\}$ used in the sketch for Schedule are constructed as follows:

$$\begin{aligned}
p_1 &\equiv \mathit{Count}(\mathit{shape}_g[0,1] \setminus \mathit{shape}[0,1]) \\
p_2 &\equiv \mathit{Count}(\mathit{surface\text{-}condition}_g[0,1] \setminus \mathit{surface\text{-}condition}[0,1]) \\
p_3 &\equiv \mathit{Count}(\mathit{painted}_g[0,1] \setminus \mathit{painted}[0,1]) \\
h &\equiv \mathit{Count}(\mathit{temperature}[0,1]|_{\mathit{hot}}) \\
o &\equiv \neg \mathit{Empty}(\mathit{scheduled}[0] \sqcup \mathit{busy}[0])
\end{aligned}$$

References

Asai, M. (2019). Unsupervised grounding of plannable first-order logic representation from images. In Lipovetzky, N., Onaindia, E., & Smith, D. E. (Eds.), *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)*, pp. 583–591. AAAI Press.

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., & Patel-Schneider, P. F. (Eds.). (2003). *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press.

Bacchus, F. (2001). The AIPS'00 planning competition. *AI Magazine*, *22*(3), 47–56.

Bacchus, F., & Kabanza, F. (2000). Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, *116*(1–2), 123–191.

Baier, J. A., Fritz, C., Bienvenu, M., & McIlraith, S. A. (2008). Beyond classical planning: Procedural control knowledge and preferences in state-of-the-art planners. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, pp. 1509–1512. AAAI Press.

Bonet, B., Francès, G., & Geffner, H. (2019). Learning features and abstract actions for computing generalized plans. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI 2019)*, pp. 2703–2710. AAAI Press.

Bonet, B., & Geffner, H. (2018). Features, projections, and representation change for generalized planning. In Lang, J. (Ed.), *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018)*, pp. 4667–4673. IJCAI.

Bonet, B., & Geffner, H. (2020a). Learning first-order symbolic representations for planning from the structure of the state space. In De Giacomo, G. (Ed.), *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020)*, pp. 2322–2329. IOS Press.

Bonet, B., & Geffner, H. (2020b). Qualitative numeric planning: Reductions and complexity. *Journal of Artificial Intelligence Research*, *69*, 923–961.

Bonet, B., & Geffner, H. (2021). General policies, representations, and planning width. In Leyton-Brown, K., & Mausam (Eds.), *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021)*, pp. 11764–11773. AAAI Press.

Daggelinckx, A. (2023). Generating and verifying planning sketches using temporal logic. Master's thesis, Utrecht University.

Dalmau-Moreno, M., García, N., Gómez, V., & Geffner, H. (2023). Combined task and motion planning via sketch decompositions. In *ICAPS 2023 Planning and Robotics Workshop (PlanRob)*.

Drexler, D., Francès, G., & Seipp, J. (2022). Description logics state features for planning (DLPlan). `https://doi.org/10.5281/zenodo.5826139`.

Drexler, D., Seipp, J., & Geffner, H. (2021). Expressing and exploiting the common subgoal structure of classical planning domains using sketches. In Erdem, E., Bienvenu, M., & Lakemeyer, G. (Eds.), *Proceedings of the Eighteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2021)*, pp. 258–268. IJCAI Organization.

Drexler, D., Seipp, J., & Geffner, H. (2022). Learning sketches for decomposing planning problems into subproblems of bounded width. In Thiébaux, S., & Yeoh, W. (Eds.), *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)*, pp. 62–70. AAAI Press.

Drexler, D., Seipp, J., & Geffner, H. (2023a). Code and data for the article "Expressing and Exploiting the Common Subgoal Structure Using Sketches". `https://doi.org/10.5281/zenodo.10037802`.

Drexler, D., Seipp, J., & Geffner, H. (2023b). Learning hierarchical policies by iteratively reducing the width of sketch rules. In Marquis, P., Son, T. C., & Kern-Isberner, G. (Eds.), *Proceedings* *of the Twentieth International Conference on Principles of Knowledge Representation and Reasoning (KR 2023)*. IJCAI Organization.

Erol, K., Hendler, J. A., & Nau, D. S. (1994). HTN planning: Complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI 1994)*, pp. 1123–1128. AAAI Press.

Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In Precup, D., & Teh, Y. W. (Eds.), *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*, pp. 1126–1135. JMLR.org.

Francès, G., Bonet, B., & Geffner, H. (2021). Learning general planning policies from small examples without supervision. In Leyton-Brown, K., & Mausam (Eds.), *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021)*, pp. 11801–11808. AAAI Press.

Francès, G., Corrêa, A. B., Geissmann, C., & Pommerening, F. (2019). Generalized potential heuristics for classical planning. In Kraus, S. (Ed.), *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, pp. 5554–5561. IJCAI.

Francès, G., Ramírez, M., Lipovetzky, N., & Geffner, H. (2017). Purely declarative action representations are overrated: Classical planning with simulators. In Sierra, C. (Ed.), *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017)*, pp. 4294–4301. IJCAI.

Geffner, T., & Geffner, H. (2015). Width-based planning for general video-game playing. In Jhala, A., & Sturtevant, N. (Eds.), *Proceedings of the Eleventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2015)*, pp. 23–29. AAAI Press.

Georgievski, I., & Aiello, M. (2015). HTN planning. *Artificial Intelligence*, *222*, 124–156.

Gerevini, A. E., Haslum, P., Long, D., Saetti, A., & Dimopoulos, Y. (2009). Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, *173*(5–6), 619–668.

Grundke, C. S. (2022). Compilability between generalized representations for classical planning. Master's thesis, University of Basel.

Hoffmann, J. (2005). Where 'ignoring delete lists' works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research*, *24*, 685–758.

Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, *14*, 253–302.

Höller, D., Behnke, G., Bercher, P., Biundo, S., Fiorino, H., Pellier, D., & Alford, R. (2019). HDDL - A language to describe hierarchical planning problems. In *Second ICAPS Workshop on Hierarchical Planning*.

Holte, R. C., Perez, M. B., Zimmer, R. M., & MacDonald, A. J. (1996). Hierarchical A*: Searching abstraction hierarchies efficiently. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI 1996)*, pp. 530–535. AAAI Press.

Icarte, R. T., Klassen, T. Q., Valenzano, R., & McIlraith, S. A. (2022). Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research*, *73*, 173–208.

Illanes, L., Yan, X., Icarte, R. T., & McIlraith, S. A. (2020). Symbolic plans as high-level instructions for reinforcement learning. In Beck, J. C., Karpas, E., & Sohrabi, S. (Eds.), *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)*, pp. 540–550. AAAI Press.

Jin, M., Ma, Z., Jin, K., Zhuo, H. H., Chen, C., & Yu, C. (2022). Creativity of AI: Automatic symbolic option discovery for facilitating deep reinforcement learning. In Honavar, V., & Spaan, M. (Eds.), *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI 2022)*, pp. 7042–7050. AAAI Press.

Kvarnström, J., & Doherty, P. (2000). TALplanner: A temporal logic based forward chaining planner. *Annals of Mathematics and Artificial Intelligence*, *30*, 119–169.

Lee, J., Katz, M., Agravante, D. J., Liu, M., Klinger, T., Campbell, M., Sohrabi, S., & Tesauro, G. (2021). AI planning annotation in reinforcement learning: Options and beyond. In *ICAPS 2021 Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL)*.

Levesque, H. J., Reiter, R., Lespérance, Y., Lin, F., & Scherl, R. B. (1997). GOLOG: A logic programming language for dynamic domains. *The Journal of Logic Programming*, *31*(1), 59–83.

Linares López, C., Celorrio, S. J., & Olaya, A. G. (2015). The deterministic part of the seventh international planning competition. *Artificial Intelligence*, *223*, 82–119.

Lipovetzky, N., & Geffner, H. (2012). Width and serialization of classical planning problems. In De Raedt, L., Bessiere, C., Dubois, D., Doherty, P., Frasconi, P., Heintz, F., & Lucas, P. (Eds.), *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, pp. 540–545. IOS Press.

Lipovetzky, N., & Geffner, H. (2017a). Best-first width search: Exploration and exploitation in classical planning. In Singh, S., & Markovitch, S. (Eds.), *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*, pp. 3590–3596. AAAI Press.

Lipovetzky, N., & Geffner, H. (2017b). A polynomial planning algorithm that beats LAMA and FF. In Barbulescu, L., Frank, J., Mausam, & Smith, S. F. (Eds.), *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*, pp. 195–199. AAAI Press.

Lipovetzky, N., Ramirez, M., & Geffner, H. (2015). Classical planning with simulators: Results on the Atari video games. In Yang, Q., & Wooldridge, M. (Eds.), *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pp. 1610–1616. AAAI Press.

Long, D., & Fox, M. (2003). The 3rd International Planning Competition: Results and analysis. *Journal of Artificial Intelligence Research*, *20*, 1–59.

Marthi, B., Russell, S., & Wolfe, J. (2008). Angelic hierarchical planning: Optimal and online algorithms. In Rintanen, J., Nebel, B., Beck, J. C., & Hansen, E. (Eds.), *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*. AAAI Press.

McDermott, D. (2000). The 1998 AI Planning Systems competition. *AI Magazine*, *21*(2), 35–55.

Nau, D. S., Au, T.-C., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., & Yaman, F. (2003). SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, *20*, 379–404.

Porteous, J., Sebastia, L., & Hoffmann, J. (2001). On the extraction, ordering, and usage of landmarks in planning. In Cesta, A., & Borrajo, D. (Eds.), *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, pp. 174–182. AAAI Press.

Ramirez, M., Lipovetzky, N., & Muise, C. (2015). Lightweight Automated Planning ToolKiT. `http://lapkt.org/`.

Richter, S., & Westphal, M. (2010). The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, *39*, 127–177.

Rodriguez, I. D., Bonet, B., Romero, J., & Geffner, H. (2021). Learning first-order representations for planning from black-box states: New results. In Erdem, E., Bienvenu, M., & Lakemeyer, G. (Eds.), *Proceedings of the Eighteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2021)*, pp. 539–548. IJCAI Organization.

Seipp, J., Pommerening, F., Röger, G., & Helmert, M. (2016). Correlation complexity of classical planning domains. In Kambhampati, S. (Ed.), *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, pp. 3242–3250. AAAI Press.

Seipp, J., Pommerening, F., Sievers, S., & Helmert, M. (2017). Downward Lab. `https://doi.org/10.5281/zenodo.790461`.

Shivashankar, V., Kuter, U., Nau, D., & Alford, R. (2012). A hierarchical goal-based formalism and algorithm for single-agent planning. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, pp. 981–988. International Foundation for Autonomous Agents and Multiagent Systems.

Shleyfman, A., Tuisov, A., & Domshlak, C. (2016). Blind search for Atari-like online planning revisited. In Kambhampati, S. (Ed.), *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, pp. 3251–3257. AAAI Press.

Singh, S. P., Lewis, R. L., Barto, A. G., & 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., & Geffner, H. (2011). Qualitative numeric planning. In Burgard, W., & Roth, D. (Eds.), *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2011)*, pp. 1010–1016. AAAI Press.

Ståhlberg, S. (2023). Lifted successor generation by maximum clique enumeration. In Gal, K., Nowé, A., Nalepa, G. J., Fairstein, R., & Rădulescu, R. (Eds.), *Proceedings of the 26th European Conference on Artificial Intelligence (ECAI 2023)*, pp. 2194–2201. IOS Press.

Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*, 181–211.

Torralba, Á., Seipp, J., & Sievers, S. (2021). Automatic instance generation for classical planning. In Goldman, R. P., Biundo, S., & Katz, M. (Eds.), *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*, pp. 376–384. AAAI Press.

Vallati, M., Chrpa, L., & 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). *The Knowledge Engineering Review*, *33*.

Zheng, Z., Oh, J., Hessel, M., Xu, Z., Kroiss, M., van Hasselt, H., Silver, D., & Singh, S. (2020). What can learned intrinsic rewards capture?. In *Proceedings of the 37th International Conference on Machine Learning (ICML 2020)*, pp. 11436–11446. JMLR.org.