Journal of Artificial Intelligence Research 80 (2024) 475-516 Submitted 10/2023; published 06/2024

General Policies, Subgoal Structure, and Planning Width

Blai Bonet BONETBLAI@GMAIL.COM
Universitat Pompeu Fabra, Spain

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

Abstract

It has been observed that many classical planning domains with atomic goals can be solved by means of a simple polynomial exploration procedure, called IW, that runs in time exponential in the problem width, which in these cases is bounded and small. Yet, while the notion of width has become part of state-of-the-art planning algorithms such as BFWS, there is no good explanation for why so many benchmark domains have bounded width when atomic goals are considered. In this work, we address this question by relating bounded width with the existence of general optimal policies that in each planning instance are represented by tuples of atoms of bounded size. We also define the notions of (explicit) serializations and serialized width that have a broader scope, as many domains have a bounded serialized width but no bounded width. Such problems are solved non-optimally in polynomial time by a variant of the Serialized IW algorithm. Finally, the language of general policies and the semantics of serializations are combined to yield a simple, meaningful, and expressive language for specifying serializations in compact form in the form of sketches, which can be used for encoding domain control knowledge by hand or for learning it from examples. Sketches express general problem decompositions in terms of subgoals, and terminating sketches of bounded width express problem decompositions that can be solved in polynomial time.

1. Introduction

Width-based search methods exploit the structure of states to enumerate the state space in ways that are different than "blind" search methods such as breadth-first and depth-first (Lipovetzky & Geffner, 2012). This is achieved by associating a non-negative integer to each state generated in the search, a so-called *novelty measure*, which is defined by the size of the smallest factor in the state that has not been seen in previously generated states. States are deemed more novel and hence preferred in the exploration search when this novelty measure is smaller. Other types of novelty measures have been used in reinforcement learning for dealing with sparse rewards and in genetic algorithms for dealing with local minima (Tang et al., 2017; Pathak et al., 2017; Ostrovski et al., 2017), but the results are mostly empirical. In classical planning, where novelty measures are part of state-of-the-art search algorithms (Lipovetzky & Geffner, 2017b, 2017a), there is a solid body of theory that relates a specific type of novelty measures with a notion of problem *width* that bounds the complexity of planning problems (Lipovetzky & Geffner, 2012).

The basic width-based planning algorithms are simple and assume a fixed number of *Boolean state features* F that in classical planning are given by the atoms in the problem. The procedure IW(1) is a breadth-first search that starts in the given initial state and prunes

all the states that do not make a feature from F true for the first time in the search. IW(k) is like IW(1) but using conjunctions of up to k features from F instead. Alternatively, IW(k) can be regarded as a breadth-first search that prunes states s with novelty measures that are greater than k, where the novelty measure of s is the size of the minimum conjunction (set) of features that is true in s but false in all the states generated before s.

For many benchmark domains, it has been shown that IW(k) for a small value of k suffices to compute plans, and indeed optimal plans, for any atomic goal (Lipovetzky & Geffner, 2012). State-of-the-art planning algorithms like BFWS (Lipovetzky & Geffner, 2017b, 2017a) make use of this property for *serializing* conjunctive goals into atomic ones, an idea that is also present in algorithms that pre-compute atomic landmarks (Hoffmann et al., 2004) and use them as counting heuristics (Richter & Westphal, 2010).

An important open question in the area is why these width-based methods are effective, and in particular, why so many domains have a small width when atomic goals are considered. Is this a property of the domains? Is it an accident of the domain encodings used? In this work, we address these and related questions. For this, we bring the notion of *general policies*; policies that solve multiple instances of a planning domain all at once (Srivastava et al., 2008; Bonet & Geffner, 2015; Hu & De Giacomo, 2011; Belle & Levesque, 2016; Segovia et al., 2016), while using the formulation of general policies expressed in terms of finite sets of rules over a fixed set of Boolean and numerical features (Bonet & Geffner, 2018).

In this paper, a class of instances is shown to have *bounded width* when there is a *general optimal policy* for the class which can be "followed" in each instance by just considering atom tuples of bounded size, without assuming knowledge of the policies or the features involved. The existing planning domains that have been shown to have bounded width can be all characterized in this way. In addition, the notion of general policies is extended to comprise serializations that split problems into subproblems. A serialization has bounded width when the resulting subproblems have bounded width and can be solved greedily for reaching the problem goal. A general policy turns out to be a serialization of width zero; namely, one in which the subproblems can be solved in a single step. Finally, the syntax of general policies is combined with the semantics of serializations to yield a simple, meaningful, and expressive language for specifying serializations succinctly in the form of *sketches*, which can be used to encode domain control knowledge by hand, or for learning it from examples (Drexler et al., 2021, 2022).

The paper is organized as follows. We review first the notions of planning, width, and general policies, and relate width with the size of the tuples of atoms that are needed to apply such policies. We then introduce serializations, the more general notion of serialized width, the relation between general policies and serialized width, and policy sketches. We finally summarize the main contributions, discuss extensions and limitations, related work, and conclusions.

2. Planning

A classical planning problem is a pair $P = \langle D, I \rangle$ where D is a first-order *domain*, such as a STRIPS domain, and I contains information about the instance (Geffner & Bonet, 2013; Ghallab et al., 2016; Haslum et al., 2019). The domain D has a set of predicate symbols

p and a set of action schemas with preconditions and effects given by atoms $p(x_1, \ldots, x_k)$, where p is a predicate symbol of arity k, and each x_i is an argument of the schema. The instance information is a tuple $I = \langle O, \mathit{Init}, G \rangle$ where O is a set of object names c_i, and *Init* and G are sets of *ground atoms* $p(c_1, \ldots, c_k)$ denoting the initial and goal situations.

A classical problem $P = \langle D, I \rangle$ encodes a state model $S(P) = \langle S, s_0, S_G, \mathit{Act}, A, f \rangle$ in compact form where the states $s \in S$ are *sets of ground atoms* from P (assumed to be the true atoms in s), s_0 is the initial state *Init*, S_G is the set of goal states s such that $G \subseteq s$, *Act* is the set of ground actions in P, $A(s)$ is the set of ground actions whose preconditions are (true) in s, and $f(a, s)$, for $a \in A(s)$, represents the state s' that follows action a in the state s. An action sequence $\sigma = a_0, a_1, \ldots, a_n$ is applicable in P if $a_i \in A(s_i)$ and $s_{i+1} = f(a_i, s_i)$, for $i = 0, \ldots, n$. The states s_i in such a sequence are said to be *reachable* in P, and states(P) denotes the set of reachable states in P. The applicable action sequence $\sigma = a_0, a_1, \ldots, a_n$ is a *plan* if $s_{n+1} \in S_G$. The *cost* of a plan is given by its **length,** and a plan is *optimal* if there is no shorter plan. If there is a plan for P, the goal G of P is said to be reachable, and P is said to be *solvable*. The *cost* of P, $\mathit{cost}(P)$, is the cost of an optimal plan for P, if such a plan exists, and infinite otherwise. A reachable state s is a *dead-end* if the goal is not reachable in the problem $P[s]$ that is like P but with s as the initial state.

We refer to subsets of ground atoms in a planning problem P as tuples of atoms, or atom tuples. An atom tuple t is reachable in P if there is a reachable state s in P that makes true all the atoms in t. The cost of a reachable state s is the minimum cost of an applicable action sequence that reaches s, the cost of a reachable tuple t is the minimum cost of a state that makes t true, and the cost of a state or tuple that is unreachable is infinite.

3. Width

While the standard definition of width is based on the consideration of sequences of atom tuples (Lipovetzky & Geffner, 2012), the formulation below is slightly more general and more convenient for our purposes.

Definition 1 (Admissible tuple set)**.** *A set T of **reachable** atom tuples in a planning problem P is **admissible** if*

1. *some tuple in T is true in the initial state of P, and*
2. *any optimal plan for a tuple t in T, that is **not** an optimal plan for P, can be extended into an optimal plan for another tuple t' in T by adding a single action.*

The definition takes the goal of the problem into account as the second clause refers to the optimal plans of P. Lipovetzky and Geffner (2012) say that a **sequence** $(t_0, t_1, \ldots, t_n)$ of atom tuples is admissible if t_0 is true at the initial state, any optimal plan for t_i can be extended with one action into an optimal plan for t_{i+1}, $0 \leq i < n$, and any optimal plan for t_n is an optimal plan for P. It is easy to show that in such a case, the set $T = \{t_0, t_1, \ldots, t_n\}$ is admissible according to the new definition of admissibility. There cases, however, where all optimal plans for a fixed tuple t in T can be extended with a single action into plans that are optimal for **either** t' or t'' but not for a unique tuple of the same size. In such cases, the width of a problem can be smaller according to the new definition.

For a tuple t and a set of tuples T, $|t|$ denotes the number of atoms in t, $|T|$ denotes the number of tuples in T, and size(T) denotes the maximum $|t|$ for a tuple t in T. The **width** of a problem P is the size of a minimum-size set of tuples T that is admissible:

Definition 2 (Width)**.** *The **width** of a STRIPS planning problem P is $w(P) \doteq \min_T$ size(T) where T ranges over the sets of tuples that are **admissible** in P. If P is solvable in zero or one step, $w(P)$ is set to 0, and if P is not solvable at all, $w(P)$ is set to ∞.*

The definition of width for classes of problems $\mathcal{Q}$ is then:

Definition 3 (Width of $\mathcal{Q}$)**.** *The **width** $w(\mathcal{Q})$ of a collection $\mathcal{Q}$ of STRIPS problems is the minimum integer k such that $w(P) \leq k$ for each problem P in $\mathcal{Q}$, or ∞ if no such k exists.*

The reason for defining the width of P as 0 or ∞ according to whether P is solvable in one step or not solvable at all, respectively, is convenience. Basically, problems are solvable in time exponential in their width, and hence, $w(P) = 0$ implies that P can be solved in constant time, if a constant branching factor is assumed. At the same time, a bounded width for a class of problems implies that all the problems in the class are solvable, something which is not ensured by setting $w(P)$ to $N + 1$, where N is the number of problem atoms (Lipovetzky & Geffner, 2012).

Example 1 (The Blocksworld domain)**.**

- $\mathcal{Q}_{Blocks}$ is the class of all Blocksworld problems over the standard domain specification with 4 operator schemas: stack/unstack and pick/putdown operators. This class of problems has **unbounded width** as shown in Example 2.

- $\mathcal{Q}_{Clear}$ is the subclass of $\mathcal{Q}_{Blocks}$ made of the problems P whose goal is the single atom $clear(x)$, for some block x, and where the gripper is initially empty.

  Let P be a problem in $\mathcal{Q}_{Clear}$, let $B_1, \ldots, B_\ell$ be the blocks above x, from top to bottom in the initial state of P, and let us consider the set of tuples $T = \{t_0, t_1, \ldots, t_{2\ell-1}\}$ where

  – $t_0 = \{clear(B_1)\}$, and

  – $t_{2i-1} = \{hold(B_i)\}$ and $t_{2i} = \{ontable(B_i)\}$ for $1 \leq i \leq \ell$.

  It is easy to check that the two conditions in Definition 1 hold for T. Hence, $w(P) \leq 1$, and since $w(P) > 0$, as the goal cannot be reached in zero or one step in general, $w(P) = 1$ and $w(\mathcal{Q}_{Clear}) = 1$.

- $\mathcal{Q}_{On}$ is the subclass of $\mathcal{Q}_{Blocks}$ made of the problems whose goal is the single atom $on(x, y)$ for two blocks x and y.

  Let us calculate the width of an arbitrary instance P in $\mathcal{Q}_{On}$ with the assumption that the blocks x and y are initially at *different* towers. Let $B_1, \ldots, B_\ell$ (resp. $D_1, \ldots, D_m$) be the blocks above x (resp. y), in order from top to bottom, in the initial state. Let us consider the set $T = \{t_0, \ldots, t_{2\ell}, t'_0, \ldots, t'_{2m}, t''_0, t''_1\}$ of tuples where

  – t_i for $0 \leq i \leq 2\ell$ same as above,

Algorithm 1: IW(T) Search

1: **Input:** Planning problem P with N ground atoms
2: **Input:** Set T of atom tuples from P
3: Initialize **perfect hash table** H for storing the tuples in T on which the operations of insertion and look up take constant time
4: Initialize FIFO queue Q on which the enqueue and dequeue operations take constant time
5: Enqueue node for the initial state s_0 of P
6: While Q is not empty:
7: Dequeue node n for state s
8: If s is a goal state, return the path to node n (Solution found)
9: If s makes true some tuple t from T that is not in H:
10: Insert all tuples from T made true by s in H
11: Enqueue a node n' for each successor s' of s
12: Return **FAILURE** (T is not admissible for P)

Figure 1: IW(T) is a breadth-first search that prunes nodes that do not satisfy a tuple in T for the first time in the search. Algorithm IW(k) is IW(T) where T is the set of conjunctions of up to k atoms in T. Conditions for the completeness and optimality of IW(T) are given in Theorem 4.

- $t'_{2i} = \{hold(D_i), clear(x)\}$ for $0 \leq i \leq m$,
- $t'_{2i-1} = \{ontable(D_i), clear(x)\}$ for $0 \leq i \leq m$,
- $t''_0 = \{hold(x), clear(y)\}$, and
- $t''_1 = \{on(x, y)\}$.

It is not difficult to check that T is admissible. Later, we will show that $w(P) > 1$ and thus $w(P) = 2$. If the two blocks x and y are in the same tower in the initial state, a different set of tuples must be considered but the width is still bounded by 2. Hence, $w(\mathcal{Q}_{On}) = 2$.

3.1 Algorithms IW(T), IW(k), and IW

If T is an admissible set of tuples for P, there is a very simple algorithm IW(T) that solves P optimally by expanding no more than $|T|$ states. The algorithm, shown in Fig. 1, carries out a forward, breadth-first search where every newly generated node that does not make a tuple in T true for the first time in the search is pruned.

Theorem 4 (Completeness of IW(T))**.** *If T is an **admissible** set of atom tuples in problem P, IW(T) finds an **optimal plan** for P. Moreover, IW(T') finds an optimal plan for P for any set T' that contains T.*

Proof. The first claim is a special case of the second. Let us assume that T' contains an admissible set T, and that T is **minimal**; i.e., an admissible set T that ceases to be admissible if some tuple is removed. Then, by Definition 1, the cost of any plan for P cannot be strictly less that the (optimal) cost of any tuple in the minimal admissible set T.

We show with induction the following invariant: *at the beginning of each iteration of IW(T'), the queue Q contains at least one node that represents either an optimal plan for some tuple in T, not yet in the hash table H, or an optimal plan for P.* Such a node in Q is called a **witness**.

For the base case, at the beginning of the first iteration, H is empty and Q only contains a node for the initial state s_0 that makes true some tuple in T.

Let us assume that the invariant holds up to iteration k; i.e., Q contains a witness n^*. Let n be the node dequeued at such iteration for state s. We will show that either IW(T') terminates with an optimal plan for P, or the invariant holds for the next iteration. We consider three cases, in order:

1. The state s is a goal state. Since the nodes are ordered by their costs, the cost of a plan for P is at most the cost of a tuple in T. By above observation, n must represents an optimal plan for P, and IW(T') terminates with an optimal plan for P.

2. All tuples in T' made true by s are already in H. Then, the node n is pruned, H is left intact, and Q is updated by only removing the node n. However, Q still contains n^* that continues to be a witness for the next iteration.

3. The node n makes true some tuple in $T' \setminus H$. We consider two subcases:

   a) The node n is a witness. By Definition 1, a node n' for a successor s' of s that represents an optimal plan for a tuple t' in T is inserted into Q. Such a tuple cannot be in H as all the tuples in H have costs strictly less that cost(t'). Therefore, n' becomes a witness for the next iteration.

   b) The node n is not a witness. The queue Q is updated by adding nodes for all the successors of s, and the hash table H is updated by adding the tuples in $\{t \in T' : s \vDash t\}$. We claim that the update on H cannot invalidate any witness in Q; e.g., n^*. Indeed, if t^* is a tuple in $T \setminus H$ reached optimally by n^*, and $s \vDash t^*$, then n also reaches t^*. On the other hand, cost(n) $\leq$ cost(n^*) as n is dequeued before n^*. Hence, n reaches t^* optimally, and n is also a witness. As this contradicts the assumption, the update on H cannot invalidate any witness.

The invariant is thus satisfied across all iterations. Therefore, IW(T') cannot return **FAILURE**, and must end with an optimal plan for P. □

To obtain simple expressions that bound the time and space used by IW(T), and other algorithms, we make the following assumptions on the time/space required by the basic operations on states and tuples. For some domains, these bounds can be attained by suitable modifications of the algorithms, like clever choice of data structures, and preprocessing that can be amortized. For other domains, the time/space complexities given in this paper may need to be adjusted for deviations from these assumptions.

Assumption (Simple time and space analyses)**.** *For any planning problem P and reachable state s in P, the generation of the set Succ(s) of its successor states takes* ***time proportional to*** *|Succ(s)|****,*** *and checking whether s is a goal state or makes true a given atom tuple t takes* ***constant time.*** *Likewise, each such state can be stored in a* ***constant amount***

of memory. *These complexities are* ***independent*** *of the numbers of ground atoms and actions in P, and the size of the tuple t.*

Under this assumption, the time and space requirements of IW(T) can be expressed as:

Theorem 5 (Complexity of IW(T))**.** *Let P be a planning problem with branching factor bounded by b, and let T be a set of atom tuples. Then,*

1. *IW(T) expands and generates at most $|T|$ and $b|T|$ nodes, respectively, thus running in $\mathcal{O}(bT^2)$ time and $\mathcal{O}(bT)$ space (where the T inside the $\mathcal{O}$-notation refers to $|T|$).*
2. *If P has N ground atoms, the number of atoms that "flip" value across a transition in P is* ***known and bounded by a constant*** *(as in STRIPS domains), and size$(T) \leq k$, then a running time of $\mathcal{O}(bTN^{k-1})$ can be obtained.*

Proof. Recall that a node is generated if it is inserted into the queue (cf. lines 5 and 11), and it is expanded if its successor nodes are generated (cf. line 11). A node is expanded if it makes true some tuple in T for the first time in the search (cf. line 9). Thus, IW(T) expands and generates up to $|T|$ and $b|T|$ nodes, respectively.

The construction of the hash table in line 3 requires time and space linear in the number $|T|$ of keys to be stored.

The test in line 8 requires checking whether some tuple t in T that belongs to the state s is not in the hash table. This can be done in $\mathcal{O}(T)$ time by iterating over each tuple in T and checking whether it belongs to s and the hash table. Hence, assuming constant time and space for the generation of successor states and the check of tuple satisfiability by states, IW(T) runs in $\mathcal{O}(bT^2)$ time and $\mathcal{O}(bT)$ space.

For obtaining the bound described in 2. an implementation that keeps track of the set of atoms Δ that change value when a successor state s' of s is generated is needed. Then, a tuple t true in s' is novel iff it contains some atom in Δ as the tuples true in s' that do not contain such atoms are also true in s. If all the tuples in T are of size at most k, and the size of Δ is bound by a constant (omitted in the $\mathcal{O}$-notation), the number of tuples that need to be checked in line 8 is at most $\mathcal{O}(N^{k-1})$. □

The algorithm IW(k) (Lipovetzky & Geffner, 2012) is a special case of IW(T) where T is the set T^k of all conjunctions of up to k atoms.[1] Versions of IW(T) have been used before for planning in a class of video-games (Geffner & Geffner, 2015), where IW(1) explored too few nodes and IW(2) too many. The set T was then defined to comprise a selected class of atoms pairs. The properties of IW(k) are:

Theorem 6 (Lipovetzky and Geffner, 2012)**.** *Let P be a planning problem with N atoms, branching factor bounded by b, and where the number of atoms that "flip" value across a*

1. There is a minor difference between the algorithm IW(k) of Lipovetzky and Geffner (2012) and the version that results from IW(T) when T is set to T^k. In the first case, non-novel nodes are pruned once they are generated and thus not enqueued during the search. In the second case, a node is pruned when it is selected for expansion if it is not novel and it is not a goal. This difference does not affect the formal properties of the algorithm (optimality and time/memory complexity) except in a "border case", ensuring that IW(0) is complete and optimal for problems of width zero, solvable in one step, according to the new definition.

Algorithm 2: IW Search

1: **Input:** Planning problem P with N ground atoms
2: For $k = 0, 1, \ldots, N$ do:
3: Run IW(k) on P
4: If IW(k) finds a plan for P, return the plan
5: Return "no plan exists for P" (P has no solution)

Figure 2: IW performs multiple IW(k) searches for increasing values of $k = 0, 1, \ldots, N$, where N is the number of ground atoms in N. The completeness of IW is given in Theorem 7.

transition is known and bounded. For each non-negative integer k, IW(k) expands up to N^k nodes, generates up to bN^k nodes, and runs in $\mathcal{O}(bN^{2k-1})$ time and $\mathcal{O}(bN^k)$ space. IW(k) is guaranteed to solve P optimally (shortest path) if $w(P) \leq k$.

Proof. Direct from Theorems 4 and 5. □

Finally, the algorithm IW, shown in Fig. 2, runs IW(k) for increasing values $k = 0, 1, \ldots, N$ stopping when a plan is found, or when no plan is found after $k = N$, where N is the number of atoms in P.

Theorem 7 (Completeness of IW)**.** *If $w(P) \leq k$, IW finds a plan for P,* ***not necessarily optimal,*** *in time and space bounded by $\mathcal{O}(bN^{2k-1})$ and $\mathcal{O}(bN^k)$ respectively.*

Proof. If $w(P) \leq k$, IW(k) finds an optimal plan for P, but a sub-optimal plan may be found by IW(i) for $i < k$. In either case, by Theorem 6, IW(i) runs in time and space $\mathcal{O}(bN^{2i-1})$ and $\mathcal{O}(bN^i)$, respectively, for $i = 1, \ldots, k$, which are dominated by the expressions with $i = k$. □

It is not always necessary to execute all the iterations in the loop in IW. Indeed, if the call to IW(k), for $k = i$, ends up pruning only *duplicate states*, the search is already complete and further calls of IW(k) for $k = i+1, i+2, \ldots, N$, will expand and prune exactly the same sets of nodes expanded and pruned by IW(i).

The procedure IW solves problems P of width bounded by k in polynomial time but not necessarily optimally. IW(k) solves such problems optimally:

Theorem 8. *Let $\mathcal{Q}$ be a collection of problems of* ***bounded width****. Then, any problem P in $\mathcal{Q}$ is solved in polynomial time by the IW algorithm. If $w(\mathcal{Q}) \leq k$, IW(k) optimally solves any instance in $\mathcal{Q}$ in polynomial time.*

Proof. Let $w(\mathcal{Q}) = k$. For any planning problem P in $\mathcal{Q}$, IW runs IW(i) until $i = k$ when IW(k) is guaranteed to find a plan for P. Each run of IW(i) takes polynomial time as k is fixed for any problem P in $\mathcal{Q}$. If the bound k is **known**, IW(k) can be run instead, solving each problem P in $\mathcal{Q}$ optimally. □

Example 2 (IW(k) for Blocksworld instances)**.**

- If $w(\mathcal{Q}_{Blocks}) \leq k$, every problem P in $\mathcal{Q}_{Blocks}$ would be optimally solvable in polynomial time by IW(k), Theorem 8. Since computing optimal plans for arbitrary instances

of Blocksworld is NP-hard (Chenoweth, 1991; Gupta & Nau, 1992), the width of $\mathcal{Q}_{Blocks}$ must be **unbounded,** unless P = NP.

- Any transition in a Blocksworld instance flips at most 5 atoms. Hence, since $w(\mathcal{Q}_{Clear}) = 1$ (resp. $w(\mathcal{Q}_{On}) = 2$), IW(1) (resp. IW(2)) is guaranteed to find an optimal plan for any problem P in $\mathcal{Q}_{Clear}$ (resp. $\mathcal{Q}_{On}$) with N ground atoms in $\mathcal{O}(N)$ time and space (resp. $\mathcal{O}(N^3)$ time and $\mathcal{O}(N^2)$ space).

From now on, we assume that the actions in the instances for a class $\mathcal{Q}$ flip at most a constant number of atoms, which is actually the case when the instances come from a STRIPS domain.

4. General Policies

General policies over a class $\mathcal{Q}$ of problems P are defined **semantically** first, then **syntactically.** Semantically, a policy π is regarded as a relation on state pairs (s, s') that are state transitions. A state transition is a pair of states (s, s') such that there is an action a that is applicable in s and which maps s into s'. The reason for policies to be defined as relations on state pairs and not on state-action pairs (e.g., as in RL and MDPs) is that the set of actions changes across the instance in $\mathcal{Q}$. Policies π expressed as relations on state transitions (s, s') specify the possible actions to do in s indirectly and non-deterministically (Bonet & Geffner, 2018): if (s, s') is a state transition in the relation π, *any* action a that maps s into s' is allowed by π and can thus be applied at the state s.

Definition 9 (Policies)**.** *A* ***policy*** π *for a class of problems* $\mathcal{Q}$ *is a* ***binary relation*** *on* $\cup_{P\in\mathcal{Q}} states(P)$*; i.e., on the reachable states of the problems in* $\mathcal{Q}$*, such that a state pair* (s, s') *is in* π *only if* (s, s') *is a* state transition *in* P*. Furthermore,*

1. *A state transition* (s, s') *in* P *is a* π***-transition*** *if* $(s, s') \in \pi$*, and* s *is* ***not*** *a goal state in* P*.*

2. *A* π***-trajectory*** *in* P *is a sequence* $s_0, s_1, \ldots, s_n$ *of states in* P *such that* (s_i, s_{i+1}) *is a* π*-transition,* $0 \leq i < n$*, and* s_0 *is the initial state of* P*. The state* s *is* π***-reachable*** *if there is a* π*-trajectory that ends at* s*.*

3. *A* π*-trajectory* $s_0, s_1, \ldots, s_n$ *in* P *is* ***maximal*** *if there are no* π*-transitions* (s_n, s) *in* P*, or* $s_n = s_i$ *for some* $0 \leq i < n$*. In the latter case, the trajectory is called* ***cyclic.***

4. *The policy* π ***solves*** P *if* ***every maximal*** π*-trajectory* $s_0, s_1, \ldots, s_n$ *ends in a goal state (i.e.,* s_n *is a goal state).*

5. *The policy* π *solves* P ***optimally*** *if* ***every maximal*** π*-trajectory reaches a goal state in* n *steps, where* n *is the cost of* P*.*

6. *The policy* π ***solves*** $\mathcal{Q}$ *if* π ***solves*** *each problem* P *in* $\mathcal{Q}$*, and it is* ***optimal*** *for* $\mathcal{Q}$ *if it solves each problem in* $\mathcal{Q}$ ***optimally.***

Sufficient and necessary conditions for a general policy π to solve a class of problems $\mathcal{Q}$ can be expressed with suitable notions that apply to each of the instances in $\mathcal{Q}$:

Definition 10 (Policy concepts)**.** *Let $\mathcal{Q}$ be a class of problems, and let π be a policy for $\mathcal{Q}$. Then,*

1. *π is **closed** in $\mathcal{Q}$ if there is a π-transition (s, s') from every π-reachable state s in $P \in \mathcal{Q}$ that is not a **goal** state.*
2. *π is **acyclic** in $\mathcal{Q}$ if there is no cyclic π-trajectory starting in an initial state for $P \in \mathcal{Q}$.*

Theorem 11 (Requirements for solvability)**.** *A policy π solves a class of problems $\mathcal{Q}$ iff π is closed and acyclic in $\mathcal{Q}$.*

Proof. Direct. If π is closed and acyclic, π solves every problem in $\mathcal{Q}$. Conversely, if π is either not closed or acyclic, there is at least one problem P in $\mathcal{Q}$ that π does not solve. □

Example 3 (General (semantic) policy for $\mathcal{Q}_{Clear}$)**.**

- Let π be the general policy for $\mathcal{Q}_{Clear}$ defined as the set of transitions (s, s') such that:

  a) for a block y **above** x, $s \vDash \text{clear}(y) \wedge \text{hand-empty}$ and $s' \vDash \text{hold}(y)$, or

  b) $s \vDash \text{hold}(y)$ and $s' \vDash \text{ontable}(y)$.

  To show that π solves all the instances in $\mathcal{Q}_{Clear}$ one needs to show that π is closed and acyclic. For closedness, it is easy to see that in every reachable state s in an instance P in $\mathcal{Q}$ that is not a goal state, there are successors s' for which condition (a) or (b) holds. Finally, π is acyclic because in any transition of a π-trajectory, one of the blocks that is initially above x is unstacked or placed on the table, and no block is ever stacked above x, so every π-trajectory must be finite.

4.1 Rule-based Policies

General policies are relations over the state transitions, and following Bonet and Geffner (2018), these relations can be defined in a compact way with rules of the form $C \mapsto E$, that maps a *condition* C to an *effect* E, over a fixed set Φ of Boolean and numerical features over $\mathcal{Q}$.

Definition 12 (Features)**.** *A Boolean (resp. numerical) feature for a problem P is a function that maps reachable states into Boolean values (resp. non-negative integers). A feature for a class $\mathcal{Q}$ of problems is a feature that is defined on each problem P in $\mathcal{Q}$. The value of feature ϕ at state s is denoted by $\phi(s)$.*

When providing time bounds, we will be interested in **linear features** defined as follows:

Definition 13 (Linear features)**.** *A feature ϕ for problem P with N ground atoms is **linear** if for any reachable state s in P, the value $\phi(s)$ can be computed in $\mathcal{O}(N)$ time (i.e., linear time), and if ϕ is numerical, the size of $\{\phi(s) : s \in \text{states}(P)\}$ is $\mathcal{O}(N)$. A set Φ of features is a **set of linear features** for P if each ϕ in Φ is linear for P, and it is a set of linear features for a class $\mathcal{Q}$, if Φ is a set of linear features for each problem P in $\mathcal{Q}$.*

The form of the rules that make use of the features is as follows:

Definition 14 (Rules)**.** *Let Φ be a set of features. A Φ-rule is a rule of the form $C \mapsto E$ where the **condition** C is a set (conjunction) of Boolean feature conditions and the **effect** E is a set (conjunction) of feature value changes. A Boolean feature condition is of the form p, $\neg p$, $n = 0$, and $n > 0$ for Boolean and numerical features p and n in Φ. Feature value changes are of the form p, $\neg p$, $p?$ for Boolean p, and $n\downarrow$, $n\uparrow$, and $n?$ for numerical n.*

A collection of rules defines a binary relation over the states in the problems in $\mathcal{Q}$. While we consider **rule-based policies** first that define policies, we will also use rules to define another type of binary relation, **serializations**. While policies select state transitions (s, s') from states s, serializations select state pairs (s, s') where s' is not necessarily reachable from s through a single action:

Definition 15. *A state pair (s, s') **satisfies** the condition C of a Φ-rule $C \mapsto E$ if the feature conditions in C are all true in s. The transition (s, s') **satisfies** the effect E if the values for the features in Φ change from s to s' according to E; i.e.,*

1. *if p (resp. $\neg p$) is in E, then $p(s') = 1$ (resp. $p(s') = 0$),*
2. *if $n\downarrow$ (resp. $n\uparrow$) is in E, then $n(s) > n(s')$ (resp. $n(s) < n(s')$),*
3. *if $p?$ (resp. $n?$) is in E, then any change in value is allowed, and*
4. *if p (resp. n) is **not mentioned** in E, then $p(s) = p(s')$ (resp. $n(s) = n(s')$).*

*The state pair (s, s') is **compatible** with a rule $C \mapsto E$ if s satisfies C, and the pair satisfies E. The pair is then also said to satisfy the rule itself. The state pair is compatible with a **set** R of rules if it is compatible with at least one rule in the set. The pair (s, s') is then said to be an R-pair, or to be **in** R.*

A set of rules R defines a policy in the following way:

Definition 16 (Rule-based policies)**.** *Let $\mathcal{Q}$ be a collection of problems and let Φ be a set of features for $\mathcal{Q}$. A set of rules R over Φ defines the policy π_R for $\mathcal{Q}$ in which a state pair (s, s') over an instance $P \in \mathcal{Q}$ is in π_R iff (s, s') is a state transition in P that satisfies a rule in R. The set of rules R solves $\mathcal{Q}$ iff the policy π_R solves $\mathcal{Q}$.*

Example 4 (General policy for $\mathcal{Q}_{Clear}$)**.**

- A general policy for $\mathcal{Q}_{Clear}$ can be expressed using rules over the set $\Phi = \{H, n\}$ of two features, where H is the Boolean feature that is true when the gripper holds a block, and n is the numerical feature that counts the number of blocks above x. The policy has the two rules:

$$\{\neg H, n > 0\} \mapsto \{H, n\downarrow\},$$
$$\{H\} \mapsto \{\neg H\}.$$

  The first rule says that when the gripper is empty and there are blocks above x, an action that decreases n and makes H true must be chosen, while the second rule says that when the gripper holds a block, an action that makes H false and does not affect

n must be selected. This policy is slightly more general than the one in the previous example as a block being held can be put "away" in any position except above x.

- As we saw before, $w(\mathcal{Q}_{Clear}) = 1$ and this policy solves any instance $\mathcal{Q}_{Clear}$.

Example 5 (The Grid domain)**.**

- The Grid domain involves an agent that moves in a rectangular grid of arbitrary but finite size, and its goal is to reach a specific cell in the grid. Any instance in Grid is encoded with objects for the cells in the grid, and a unary predicate $pos(c)$ that is true when the position of the agent is cell c. The class of Grid problems is denoted by $\mathcal{Q}_{Grid}$.

  The Grid_2 domain is like Grid except that positions are encoded with horizontal and vertical coordinates. For a $m \times n$ grid, there are m (resp. n) objects to encode the vertical (resp. horizontal) coordinates, and two unary predicates $hpos(h)$ and $vpos(v)$ to encode that the agent is at column h and row v in the grid. The class of Grid_2 problems is denoted by $\mathcal{Q}_{Grid_2}$.

- Observe that each reachable state in a problem P in $\mathcal{Q}_{Grid}$ (resp. $\mathcal{Q}_{Grid_2}$) makes true exactly 1 (resp. 2) atoms. Since all problems are solvable, $w(\mathcal{Q}_{Grid}) \leq 1$ and $w(\mathcal{Q}_{Grid_2}) \leq 2$. On the other hand, problems in $\mathcal{Q}_{Grid}$ cannot be solved in one step, in general, and thus $w(\mathcal{Q}_{Grid}) = 1$. We show below that $w(\mathcal{Q}_{Grid_2}) > 1$ which implies $w(\mathcal{Q}_{Grid_2}) = 2$.

- For either encoding, an optimal general policy can be expressed with a single rule over the singleton $\Phi = \{d\}$ of features where d measures the distance to the goal cell:

$$\{d > 0\} \mapsto \{d\downarrow\}\,.$$

  Clearly, any transition (s, s') compatible with the rule moves the agent one step closer to the goal, and the policy solves optimally any problem in $\mathcal{Q}_{Grid}$ or $\mathcal{Q}_{Grid_2}$. The feature d is linear in $\mathcal{Q}_{Grid}$ but quadratic in $\mathcal{Q}_{Grid_2}$.

Example 6 (The Delivery domain)**.**

- The Delivery domain D involves an agent that moves in a rectangular grid, and packages spread over the grid that can be picked up or dropped by the agent, which can hold one package at a time. A state for the domain thus specifies the position of the agent (i.e., a cell on the grid), and positions for the different packages, where the position of a package can be either a cell in the grid, or the agent's gripper. The task of the agent is to "deliver" all packages, one at a time, to a designated "target" cell.

  The position of the agent can be encoded using a single unary predicate as for the *Grid* domain, or two unary predicates as for the $Grid_2$ domain. For simplicity, we assume an encoding of positions using a single predicate.

  An instance for Delivery consists of objects for the different cells in the grid, and objects for the different packages. The atoms are as follows: $pos(c)$ (resp. $ppos(p, c)$) that is true when the agent (resp. the package p) is in cell c, $holding(p)$ that is true when the agent is holding package p, and *empty* that is true when the agent holds no package.

Under this encoding, the goal can be expressed as the conjunction $\bigwedge_i ppos(p_i, target)$ where p_i is the ith package, $target$ is the target cell, and the conjunction is over all the packages.

- The class of all Delivery problems in denoted by $\mathcal{Q}_D$, while $\mathcal{Q}_{D_1}$ denotes the class of Delivery problems with exactly one package. In Example 9, we show that the width of $\mathcal{Q}_D$ is **unbounded,** and $w(\mathcal{Q}_D) > 1$. On the other hand, if s is a state in a problem P in $\mathcal{Q}_{D_1}$, s either makes true exactly two atoms of form $pos(c)$ and $holding(p)$, for the unique package p, or makes true exactly three atoms of form $pos(c)$, $ppos(p, c)$, and $empty$. However, the atom $empty$ is determined by the atoms $ppos(p, c)$ and $holding(p)$: $holding(p) \Rightarrow \neg empty$, and $ppos(p, c) \Rightarrow empty$ for any cell c. This fact allows the construction of an admissible set T of size 2 for any problem P in $\mathcal{Q}_{D_1}$. Thus, $w(\mathcal{Q}_{D_1}) = 2$.

- A set of meaningful features for Delivery is $\Phi = \{H, p, t, u\}$ that capture whether the agent is holding a package, the distance to a nearest *undelivered package* (zero if the agent is holding a package, or no package to be delivered remains), the distance to the target cell, and the number of undelivered packages, respectively. The following set R of rules define a policy π_R:

$$\{\neg H, p > 0\} \mapsto \{p\downarrow, t?\},$$
$$\{\neg H, p = 0\} \mapsto \{H\},$$
$$\{H, t > 0\} \mapsto \{t\downarrow\},$$
$$\{H, t = 0, u > 0\} \mapsto \{\neg H, u\downarrow, p?\}.$$

  The first rule says that when holding no package and undelivered packages remain in the grid, the agent must move closer to a nearest undelivered package. The second that when the agent holds no package and is at the same cell of an undelivered package, the agent must pick the package. The third that when holding a package and the agent is not at the target cell, it must move closer to the target cell. The last rule says that when the agent holds a package and is at the target cell, it must drop the package at the cell (i.e., deliver it).

  It is not difficult to see that the policy R solves any problem P in both $\mathcal{Q}_D$ and $\mathcal{Q}_{D_1}$.

5. Envelopes

We address next the relation between general policies and problem width by introducing the notion of *envelopes.* An envelope for a binary relation μ over the states in a problem P is a set of reachable states E that obeys certain closure properties. Recall that we are in the unit-cost setting where optimal trajectories refer to trajectories of minimum-length.

Definition 17 (Envelopes)**.** *Let P be a problem, and let μ be a binary relation on states(P). A subset $E \subseteq states(P)$ is an **envelope of μ for** P, or simply a **μ-envelope** iff*

1. the initial state s_0 of P belongs to E, and

2. *if s is a non-goal state in E, there is a state s' in E such that (s, s') is a μ-transition; i.e., a state transition in P that is also in μ.*

The envelope is ***backward-closed****, abbreviated as* ***closed****, iff for each state s' in E that is not the initial state, there is a state s in E such that (s, s') is a μ-transition. E is an* ***optimal envelope*** *iff each maximal μ-trajectory contained in E that starts at a non-goal state s in E is a suffix of an* ***optimal trajectory*** *for P.*

Notice that if π is a policy that solves P, the set of states reached by π in the way of the goal form a μ-envelope for $\mu = \pi$, and similarly, the set of states in *any* non-empty set of π-trajectories also form a μ-envelope. These envelopes are actually closed, and indeed, for any state s in these envelopes there is a μ-trajectory that reaches the goal and passes through s. If the policy π is optimal for P, the envelopes based on the relation $\mu = \pi$ are optimal too.

Envelopes can be defined in ways that do not involve policies however. In particular, *cost-envelopes* are defined in terms of a binary relation $\mu = \prec_{cost}$ that appeals to cost considerations. For this, optimal state costs and the binary $\prec_{cost}$ relation are defined as follows:

Definition 18 (Optimal state costs and cost relation)**.** *The cost of a state s in P,* $\mathrm{cost}(s)$*, is the cost (length) of a min-cost plan for reaching s from the initial state s_0, and ∞ if there is no such plan. The* ***optimal cost*** *of a state s in P,* $\mathrm{cost}^*(s)$*, is* $\mathrm{cost}(s)$ *if s is a* ***non-goal state****, and the cost of P (i.e., the cost of an optimal plan for P) if s is a* ***goal state****. The* cost relation $\prec_{cost}$ *is the set of state pairs (s, s') from P such that* $\mathrm{cost}^*(s) < \mathrm{cost}^*(s') < \infty$*. If (s, s') is in $\prec_{cost}$, we write $s \prec_{cost} s'$.*

The reason that the optimal cost of goal states is defined as the cost of the problem P is to have a correspondence between goal reaching trajectories that are optimal and $\prec_{cost}$-trajectories:

Lemma 19 ($\prec_{cost}$-trajectories)**.** *Let P be a planning problem with initial state s_0, and let $\tau = s_0, s_1, \ldots, s_n$ be a* ***goal-reaching*** *state trajectory in P. Then, τ is a $\prec_{cost}$-trajectory iff τ is an optimal trajectory for P.*

Proof. ($\Rightarrow$) Let us assume that τ is a $\prec_{cost}$-trajectory. We use induction on i to show that the prefix $\tau_i \doteq s_0, s_1, \ldots, s_i$ is an optimal trajectory for s_i, $0 \leq i \leq n$. The claim is true for $i = 0$. Let us assume that it holds for $i = k$, and let us consider the trajectory τ_{k+1} leading to state s_{k+1}. By assumption, $\mathrm{cost}^*(s_k) < \mathrm{cost}^*(s_{k+1})$. On the other hand, the existence of the trajectory implies $\mathrm{cost}(s_{k+1}) \leq 1 + \mathrm{cost}(s_k)$. Therefore, $\mathrm{cost}(s_{k+1}) = 1 + \mathrm{cost}(s_k)$ and the trajectory τ_{k+1} is optimal for s_{k+1}. Finally, τ must be an optimal trajectory for P since otherwise $\mathrm{cost}^*(s_n) < \mathrm{cost}(s_n) = 1 + \mathrm{cost}(s_{n-1}) = 1 + \mathrm{cost}^*(s_{n-1})$ which implies $\mathrm{cost}^*(s_n) \leq \mathrm{cost}^*(s_{n-1})$, and thus $s_{n-1} \not\prec_{cost} s_n$.

($\Leftarrow$) Let us assume that τ is an optimal trajectory for P. By the principle of optimality, the trajectory τ_i must be optimal for s_i, $0 \leq i < n$. Therefore, $\mathrm{cost}(s_0) = 0$ and $\mathrm{cost}(s_{i+1}) = 1 + \mathrm{cost}(s_i)$, $0 \leq i < n$. Observe that s_n must be a **closest** goal state to s_0 and thus $\mathrm{cost}^*(s_n) = \mathrm{cost}(s_n)$. Hence, τ is a $\prec_{cost}$-trajectory. □

A property of cost-envelopes, i.e., $\prec_{cost}$-envelopes, is that they are *optimal*, and that optimal μ-envelopes are cost-envelopes independently of the relation μ:

Theorem 20 (Optimality of cost-envelopes)**.** *Let P be a planning problem, and let E be a subset of reachable states in P. If E is an optimal envelope for some binary relation μ on states(P), then it is a cost-envelope. Likewise, if E is a cost-envelope, then it is an optimal envelope.*

Proof. ($\Rightarrow$) Let us assume that E is an optimal envelope for some relation μ. We need to show that E satisfies the two conditions in Definition 17 for the relation $\prec_{cost}$. The first condition is direct since the initial state s_0 of P belongs to E as E is a μ-envelope. For the second condition, let s be a non-goal state in E. By the optimality of E, there is an optimal trajectory for P of form $\tau = s_0, s_1, \ldots, s_\ell, \ldots, s_n$ such that $s_\ell = s$ and $\{s_\ell, s_{\ell+1}, \ldots, s_n\} \subseteq E$. Since τ is optimal, $\text{cost}^*(s_i) = i$ for $i = 0, 1, \ldots, n$, and thus the transition $(s, s_{\ell+1})$ is a $\prec_{cost}$-transition with $s_{\ell+1} \in E$.

($\Leftarrow$) Let us assume that E is a cost-envelope, and let $\tau_1 = s_\ell, s_{\ell+1}, \ldots, s_n$ be a maximal $\prec_{cost}$-trajectory contained in E that starts at state s_ℓ. In particular,

$$\text{cost}^*(s_\ell) \;<\; \text{cost}^*(s_{\ell+1}) \;<\; \cdots \;<\; \text{cost}^*(s_n) \;<\; \infty\,.$$

Since s_ℓ is a reachable state in P, there is an optimal trajectory $\tau_2 = s_0, s_1, \ldots, s_\ell$ from s_0 to the state s_ℓ which satisfies, by its optimality,

$$\text{cost}^*(s_0) \;<\; \text{cost}^*(s_1) \;<\; \cdots \;<\; \text{cost}^*(s_\ell)\,.$$

Therefore, the combined trajectory $\tau = \tau_2, \tau_1$ is a goal-reaching $\prec_{cost}$-trajectory. By Lemma 19, τ is an optimal trajectory for P that contains τ_1 as a suffix. Since τ_1 is arbitrary, the envelope E is optimal. □

We will see that a relation between the width of a problem and optimal policies can be established by considering cost-envelopes defined by tuples of atoms.

5.1 Cost Envelopes and Problem Width

For a set T of **reachable atom tuples** over P, the set $OPT(T)$ is the set of min-cost states that reach the tuples in T:

Definition 21 (Optimal states for T)**.** *Let P be a problem, let t be a reachable atom tuple in P, and let T be a set of such tuples. $OPT(t)$ is the set of **optimal states** for t in P, i.e., the set of min-cost states where t is true, and $OPT(T) \doteq \cup\{OPT(t) : \text{tuple } t \text{ is in } T\}$.*

It turns out that $OPT(T)$ is a cost-envelope iff T is admissible:

Theorem 22 (Admissibility and cost-envelopes)**.** *Let P be a planning problem, and let T be a set of reachable atom tuples in P. Then, T is admissible for P iff $OPT(T)$ is a cost-envelope.*

Proof. ($\Rightarrow$) Let us assume that T is admissible for P, and let s_0 be the initial state of P. We need to show that $OPT(T)$ is a cost-envelope:

1. Since T is admissible, there is tuple t in T with $s_0 \models t$. Hence, s_0 belongs to $OPT(T)$.

2. Let s be a non-goal state in $OPT(T)$; i.e., there is tuple t in T with $s \models t$, and there is an optimal trajectory τ for t that ends in s. Using that T is admissible, it is easy to show that τ can be extended into an optimal trajectory $\tau, s_1, \ldots, s_n$ for P, where the trajectories $\tau, s_1, \ldots, s_i$ are optimal for tuples t_i in T; i.e., the states $s_1, s_2, \ldots, s_n$ all belong to $OPT(T)$. On the other hand, by Lemma 19, $\tau, s_1, \ldots, s_n$ is a $\prec_{cost}$-trajectory. Hence, (s, s_1) is a $\prec_{cost}$-transition.

($\Leftarrow$) Let us assume that $OPT(T)$ is a cost-envelope. We need to show that T is admissible; namely, that the two conditions in Definition 1 hold for T.

1. Since $OPT(T)$ is an envelope, it contains the initial state s_0 of P. Hence, there is a tuple $t \in T$ such that $s_0 \models t$.

2. Let τ be an optimal trajectory for a tuple $t \in T$ that ends in a **non-goal** state s. We need to show that τ can be extended with a single step into an optimal trajectory for a tuple $t' \in T$. Since $OPT(T)$ is a cost-envelope and $s \in OPT(T)$, there is a $\prec_{cost}$-trajectory s, s', τ' **entirely** contained in $OPT(T)$ that starts at s, transitions to s', and ends in a goal state. Hence, there is a tuple $t' \in T$ such that $s' \in OPT(t')$, and the joined trajectory τ, s', τ' is a goal-reaching $\prec_{cost}$-trajectory that starts at s_0. By Lemma 19, the trajectory τ, s' is optimal for s', and thus for the tuple t' in T. □

Therefore, optimal plans for P can be found by running the IW(T) algorithm, provided that *there is* a subset of tuples $T' \subseteq T$ such that $OPT(T')$ is a cost-envelope:

Theorem 23. *The algorithm IW(T) finds an optimal plan for P if $OPT(T')$ is a cost-envelope for some $T' \subseteq T$. Likewise, IW(k) finds an optimal plan for P if $OPT(T)$ is a cost-envelope for some set T of conjunctions of up to k atoms in P.*

Proof. If $OPT(T')$ is a cost-envelope, T' is admissible by Theorem 22, and IW(T) finds an optimal plan by Theorem 4. The second claim follows from the first and Theorem 6. □

The **width** of a problem P can thus be related to the min size of a tuple set T for which $OPT(T)$ is a cost-envelope for P:

Theorem 24. *Let P be a planning problem, and let T be a set of atom tuples in P. If $OPT(T)$ is a cost-envelope, $w(P) \leq size(T)$.*

Proof. By Theorem 22, T is admissible, and hence, by Definition 2, $w(P) \leq \text{size}(T)$. □

5.2 Optimal Policies and Problem Width

A sufficient condition for $OPT(T)$ to be a cost-envelope is for $OPT(T)$ to be a π-envelope of an **optimal policy** π for a class $\mathcal{Q}$ of problems that includes the problem P:

Theorem 25 (Envelopes for optimal policies)**.** *Let P be a planning problem, let π be an **optimal policy** for a class $\mathcal{Q}$ that includes P, and let E be a subset of reachable states in P. Then, E is a cost-envelope in P if E is a **closed** π-envelope in P.*

Proof. By Theorem 20, it is sufficient to show that E is an optimal envelope. However, this is direct since for any maximal π-trajectory τ_1 that is contained in E and that starts at some state s, by closedness, there is a π-trajectory τ_2 from the initial state of P to the state s. Hence, the trajectory $\tau = \tau_2, \tau_1$ is a goal-reaching π-trajectory from the initial state that must be optimal as π is an optimal policy. □

A closed π-envelope is an envelope formed by the states in a (non-empty) set of π-trajectories, starting in the initial state of the problem and ending at a goal state. The theorem implies that if $OPT(T)$ is a closed π-envelope, then $OPT(T)$ is a cost-envelope, and then from Theorem 24, that the width of P is bounded by the size of T:

Theorem 26 (Optimal policies and width)**.** *Let $\mathcal{Q}$ be a class of planning problems, and let π be an optimal policy for $\mathcal{Q}$. Then,*

1. *If P is a planning problem in $\mathcal{Q}$, and T is a set of atom tuples in P such that $OPT(T)$ is a closed π-envelope, then $w(P) \leq size(T)$.*
2. *If k is a non-negative integer such that for any problem P in $\mathcal{Q}$, there is a set T of atom tuples in P such that $size(T) \leq k$ and $OPT(T)$ is a closed π-envelope, then $w(\mathcal{Q}) \leq k$.*

Proof. The first claim follows by Theorems 24 and 25 as the former establishes that $w(P) \leq$ size(T) if $OPT(T)$ is a cost envelope, and the latter that $OPT(T)$ is a cost envelope if it a π-envelope of an optimal policy π. The second claim follows from the first since for any problem P in $\mathcal{Q}$, $w(P) \leq k$ by the first claim and the assumed T. □

This theorem is important as it sheds light on the notion of width and why many standard domains have bounded width when atomic goals are considered. Indeed, in such cases, the classes of instances $\mathcal{Q}$ admit optimal policies π that in each instance P in $\mathcal{Q}$ can be "followed" by considering a set of tuples T over P. If $OPT(T)$ is a closed-envelope of π, it is possible to reach the goal of P optimally through a π-trajectory without knowing π at all: one can then pay attention to the set of tuples in T only and just run the IW(T) algorithm. Indeed:

Theorem 27. *Under the assumptions of Theorem 26, IW(T) reaches the goal of P optimally, through a π-trajectory; i.e., there is a goal-reaching π-trajectory $\tau = s_0, s_1, \ldots, s_n$ seeded at the initial state s_0 of P such that all the states s_i are expanded by IW(T), except the goal state s_n that is selected for expansion but is not expanded.*

Proof. As shown in the proof of Theorem 26, $OPT(T)$ is a cost-envelope given the assumptions and T is an admissible set for P. Therefore, by Theorem 4, IW(T) finds an optimal path $\tau = s_0, s_1, \ldots, s_n$ for P; that is, IW(T) expands nodes n_i that represent the prefixes $\tau_i = s_0, s_1, \ldots, s_i$ for $0 \leq i < n$, and selects for expansion the node that represents the path τ. Since s_n is a goal state, IW(T) returns the path τ. □

When the conditions in Theorem 26 hold, the IW(T) algorithm reaches the goal of the problem $P \in \mathcal{Q}$, optimally through a π-trajectory, even if the policy π or the features involved in the policy are not known. We say in this case, that the set of tuples T **represents** the policy π in the problem P, as the IW(T) search delivers then a π-trajectory to the goal, and it is thus "following" the policy. Notice however that even in this case, it is not necessary

for the set of tuples T to capture all the possible π-trajectories to the goal. It suffices for T to capture one such trajectory. The goal of P then can be reached optimally through IW(T) by expanding no more than $|T|$ nodes (cf. Theorem 5).

It is also not necessary for the set of tuples T to be known for solving the problem P optimally. If the conditions in Theorem 26 hold for some set T of tuples of size k, i.e., $k = size(T)$, the algorithm IW(k) will deliver an optimal π-trajectory to the goal as well. On the other hand, if there is an upper bound k on the size of the tuples, but its value is not known, the algorithm IW would solve the problem in time and space exponential in k but not necessarily through an optimal π-trajectory, because non-optimal solutions can potentially be found by IW(k') for $k' < k$ (cf. Theorem 7).

5.3 Lower Bound on Width

Finally, the following result provides a lower bound on width:

Theorem 28 (Necessary conditions for bounded width)**.** *Let P be a planning problem, let k be a non-negative integer, and let T be the set of **all** atom tuples in P of size at most k. If for every optimal trajectory τ for P, τ has a state that is **not** in $OPT(T)$, then $w(P) > k$.*

Proof. We show the contrapositive of the claim. Namely, if $w(P) \leq k$, then there is an optimal trajectory τ for P that is entirely contained in $OPT(T)$. Thus, let P be a problem such that $w(P) \leq k$, and let T' be an admissible set for P with size(T') $\leq k$. We construct an optimal trajectory $\tau = s_0, s_1, \ldots, s_n$ inductively, using the admissible set T'. Indeed, T' contains a tuple t_0 that is made true by the initial state s_0, and thus $s_0 \in OPT(T')$. For the inductive step, assume that we have already constructed the prefix $\tau_i = s_0, s_1, \ldots, s_i$ such that $\tau_i \subseteq OPT(T')$ and τ_i is an optimal trajectory for s_i. In particular, there is a tuple t_i in T' such that $s_i \in OPT(t_i)$. By admissibility, τ_i can be extended with one step into an optimal trajectory for a tuple t' in T'; i.e., there is a state s_{i+1} and tuple t_{i+1} such that s_{i+1} in $OPT(t_{i+1})$ and $\tau_{i+1} \doteq \tau_i, s_{i+1}$ is an optimal trajectory for t_{i+1}. By definition of admissible sets, this process can be continued until τ_i becomes an optimal trajectory for P. At such moment, by construction, $\tau_i \subseteq OPT(T')$. To finish the proof, observe that $T' \subseteq T$ since size(T') $\leq k$. □

Example 7 (Width for the class $\mathcal{Q}_{On}$)**.**

- Let P be a problem in $\mathcal{Q}_{On}$ where in the initial state the blocks x and y are not clear and in different towers.
- Let T be the set of all atoms in P (i.e., all atom tuples of size 1). Any optimal trajectory for P contains a state s in which both x and y are clear, just before moving x on top of y. This state s is not in $OPT(T)$; e.g., it does not belong to $OPT(clear(x))$ because any optimal trajectory for $clear(x)$ does not move any block above y. By Theorem 28, $w(P) > 1$ and thus $w(\mathcal{Q}_{On}) > 1$. As we saw before, $w(\mathcal{Q}_{On}) \leq 2$. Hence, $w(\mathcal{Q}_{On}) = 2$.

Example 8 (Width for Grid problems, classes $\mathcal{Q}_{Grid}$ and $\mathcal{Q}_{Grid_2}$)**.**

- The states in problems for $\mathcal{Q}_{Grid}$ (resp. $\mathcal{Q}_{Grid_2}$) make true exactly one (resp. two) atom(s); i.e., the subset of atoms that identify the position of the agent.

- Let π be an optimal policy for $\mathcal{Q}_{Grid}$ (resp. $\mathcal{Q}_{Grid_2}$), let P be a problem in $\mathcal{Q}_{Grid}$, and let τ be a goal-reaching π-trajectory seeded at the initial state of P. Let us consider the set of atom tuples $T \doteq \{\text{atoms}(s) : s \in \tau\}$ where atoms(s) refers to the subset of atoms made true at the state s. It is easy to show that T is a closed π-envelope, and size$(T) = 1$ (resp. size$(T) = 2$). Therefore, by Theorem 26, $w(\mathcal{Q}_{Grid}) \leq 1$ and $w(\mathcal{Q}_{Grid_2}) \leq 2$.
- Likewise, $w(\mathcal{Q}_{Grid}) = 1$ as there are problems that require more than one step.
- Let P be a problem in $\mathcal{Q}_{Grid_2}$ in which the initial and target cells are not in the same row or the same column. Any optimal trajectory for P contains some state s where the agent is at a row and column different than those at the initial state. Such a state does not belong to $OPT(T)$, where T is the set of all atoms in P. By Theorem 28, $w(\mathcal{Q}_{Grid_2}) > 1$. Therefore, $w(\mathcal{Q}_{Grid_2}) = 2$.

Example 9 (Width for Delivery problems, classes $\mathcal{Q}_{D_1}$ and $\mathcal{Q}_D$)**.**

- Problems in $\mathcal{Q}_{D_1}$ involve the agent and a single package. We showed before that $w(\mathcal{Q}_{D_1}) = 2$.
- The class $\mathcal{Q}_D$ has **unbounded width,** however. The intuition is that any admissible set of atom tuples must "track" the position of an arbitrary number of packages, those that have been already delivered. We make this intuition formal in the following.
- Let P be a problem in $\mathcal{Q}_D$ with $k + 2$ packages, let π be an optimal policy for P, and let T be the set of **all** atom tuples in P of size at most k. Without loss of generality, we assume that the packages $p_1, p_2, \ldots, p_{k+2}$ must be delivered in such an order to guarantee optimality.
- Let s be the state along an optimal trajectory τ for P in which the packages $p_1, p_2, \ldots, p_{k+1}$ have been delivered, and the agent is at the target cell (just after delivering p_{k+1}). We now show that s does not belong to $OPT(T)$. Indeed, the tuples in T can only track the position of at most k objects in the set $\{p_1, p_2, \ldots, p_{k+1}\}$ of $k + 1$ objects. Hence, no matter which tuple t in T is considered, the state s does not belong to $OPT(t)$ as no state in $OPT(t)$ has $k + 1$ distinct packages at the target cell. Therefore, by Theorem 28, $w(P) > k$.
- Since $\mathcal{Q}_D$ contains problems with an arbitrary number of packages, $w(\mathcal{Q}_D) = \infty$.

5.4 Algorithm IW_Φ

The results above shed light on the power of the algorithms IW(T) and IW(k) for problems that belong to classes $\mathcal{Q}$ for which there is a general optimal policy π. Indeed, if the set of optimal T-states, $OPT(T)$, forms a closed π-envelope, the algorithm IW(T) is complete and optimal for P, and moreover, reaches the goal of P through π-trajectories, even if π is not known.

There is, however, a variant of IW(T) that does not use tuples of atoms or other particular details about the encoding for P, and uses instead a set Φ of features. The new

Algorithm 3: IW_Φ Search

Input: Planning problem P with N atoms
Input: Set Φ of features, and function f to compute its valuation on states in P
Initialize hash table H for storing feature valuations
Initialize FIFO queue Q on which enqueue and dequeue operations take constant time
Enqueue node n_0 for the initial state s_0 of P
While Q is not empty:
    Dequeue node n for state s
    If s is a goal state, return the path to node n (Solution found)
    If $f(s)$ is not in H:
        Insert the feature valuation $f(s)$ in H
        Enqueue a node n' for each successor s' of s
Return **FAILURE** (No suitable cost-envelope, cf. Theorem 29)

Figure 3: An IW_Φ search is like an $\text{IW}(T)$ search but instead of tracking the tuples in T, it tracks feature valuations, and prune nodes whose valuation have been already seen. Guarantees for completeness and optimality are given in Theorem 29.

algorithm, called IW_Φ and shown in Fig. 3, is like $\text{IW}(T)$ but works with feature valuations over Φ rather than with atom tuples. That is, IW_Φ does a breadth-first search that prunes the states s whose feature valuation $f(s)$ has been seen before during the search.

The question is the following. Assuming that π is an optimal rule-based policy that solves a class $\mathcal{Q}$ that includes P, and that Φ is the set of features used by the policy: does IW_Φ solve P optimally?

It turns out that without extra conditions, the answer to this question is no. One reason is that a policy π may solve a problem by using a number of feature valuations that is smaller (possibly, exponentially smaller) than the number of states required to reach the goal. In these cases, the IW_Φ search cannot get to the goal because any plan must involve sequences where the same feature valuation repeats. For example, a policy for the Gripper domain where a number of balls have to be carried from Room A to Room B, one by one, can be defined in terms of a set Φ of three Boolean features encoding whether the robot is in Room A, whether there are balls still left in Room A, and whether a ball is being held by the robot. The number of possible feature valuations is 8 but the length of the plans grows linearly with the number of balls.

We have the tools at our disposal however to provide conditions that ensure that the algorithm IW_Φ solves any problem $P \in \mathcal{Q}$ optimally if the policy π does. For a set of feature valuations F over the features in Φ, let $OPT(F)$ stand for the set of min-cost states s in P with feature valuation $f(s)$ in F. Sufficient conditions that ensure the completeness and optimality of IW_Φ can be expressed then as follows:

Theorem 29 (Completeness and optimality of IW_Φ)**.** *Let Φ be a set of features for a planning problem P, and let F be a set of feature valuations over Φ. Then,*

1. *If $OPT(F)$ is a cost-envelope, IW_Φ finds an optimal plan for P.*
2. *If π is an optimal policy for a class $\mathcal{Q}$, and $OPT(F)$ is a closed π-envelope for P in $\mathcal{Q}$, IW_Φ finds an optimal plan for P.*

In either case, if the features in Φ are linear (cf. Definition 13), IW_Φ finds a plan of length $\mathcal{O}(N^\ell)$ using $\mathcal{O}(bN^\ell)$ time, where ℓ is the number of ***numerical features*** *in Φ, N is the number of atoms in P, and b bounds the branching factor in P.*

Proof. Essentially, the proof involves a similar but more complex invariant that the one used in the proof for the completeness of IW(T) (cf. Theorem 4). We provide full details in what follows. The invariant that must be shown is: *at the start of each iteration, the queue contains a node n such that $n[\mathsf{state}]$ is in $OPT(F)$ and $n[\mathsf{cost}] = \text{cost}^*(n[\mathsf{state}])$, where $n[\mathsf{state}]$ denotes the state associated with the node n, $n[\mathsf{cost}]$ denotes the cost of n, and $\text{cost}^*(s)$ is the state function in Definition 18.* We do an induction on the number of iterations:

1. The claim is true for the first iteration as Q only contains the node n_0 for the initial state s_0 which is in $OPT(F)$, and $n_0[\mathsf{cost}]$ and $\text{cost}^*(s_0)$ are both equal to zero.

2. Suppose that the claim holds at the start of iteration k. That is, Q contains a node n such that $n[\mathsf{state}]$ is in $OPT(F)$, and $n[\mathsf{cost}] = \text{cost}^*(n[\mathsf{state}])$. We consider 4 cases:

   a) The node n is not dequeued. It then remains in Q and the invariant holds for the next iteration.

   b) The node n is dequeued and $n[\mathsf{state}]$ is a goal state. Then, IW_Φ terminates with a goal-reaching path.

   c) The node n is dequeued, $n[\mathsf{state}]$ is not a goal state, and the node is not pruned (cf. line 9 in IW_Φ). Since $n[\mathsf{state}]$ belongs to $OPT(F)$, a node n' is enqueued for a successor s' of $n[\mathsf{state}]$ such that s' is in $OPT(F)$. On the other hand, since $OPT(F)$ is a cost-envelope, there is a goal-reaching $\prec_{cost}$-trajectory $s_0, \ldots, n[\mathsf{state}], s', \ldots$ which is an **optimal trajectory** for P by Lemma 19. Therefore,

   $$n'[\mathsf{cost}] \;=\; 1 + n[\mathsf{cost}] \;=\; 1 + \text{cost}^*(n[\mathsf{state}]) \;=\; \text{cost}^*(s') \;=\; \text{cost}^*(n'[\mathsf{state}])$$

   where the third equality is by the principle of optimality. Hence, the invariant holds for the next iteration.

   d) The node n is dequeued, $n[\mathsf{state}]$ is not a goal state, and the node is pruned. Another node n' with $f(n'[\mathsf{state}]) = f(n[\mathsf{state}])$ was previously dequeued and expanded. Then,

   $$\text{cost}^*(n'[\mathsf{state}]) \;\leq\; n'[\mathsf{cost}] \;\leq\; n[\mathsf{cost}] \;=\; \text{cost}^*(n[\mathsf{state}]) \;\leq\; \text{cost}^*(n'[\mathsf{state}])$$

   where the second inequality is due to the nodes being ordered by costs in the queue, and the last inequality since $f(n[\mathsf{state}]) = f(n'[\mathsf{state}])$ and $n[\mathsf{state}] \in OPT(F)$. Therefore, equality holds throughout, $\text{cost}^*(n'[\mathsf{state}]) = \text{cost}^*(n[\mathsf{state}])$, and thus $n'[\mathsf{state}]$ is in $OPT(F)$. As in the previous case, at the iteration where the node n' is dequeued, a node n'' for a successor state of $n'[\mathsf{state}]$ with $n''[\mathsf{state}]$ in $OPT(F)$ and $n''[\mathsf{cost}] = \text{cost}^*(n''[\mathsf{state}])$ is generated and enqueued. Thus, the invariant holds for the next iteration because n'' is still in the queue since $n[\mathsf{cost}] = n'[\mathsf{cost}] < n''[\mathsf{cost}]$.

As the queue Q is non-empty at the start of each iteration, and the number of feature valuations is finite, the loop must terminate when a node for a goal state is dequeued (cf. line 8). We now show that the path found by IW_Φ is optimal. Let n^* be the last node dequeued by IW_Φ; i.e., $n^*[\mathsf{state}]$ is a goal state. There are two complementary cases:

- $n^*[\mathsf{state}]$ is in $OPT(F)$ and $n^*[\mathsf{cost}] = \mathrm{cost}^*(n^*[\mathsf{state}])$. Then, the path leading to n^* is an optimal trajectory for P by Lemma 19.
- The negation of the first case. By the claim, at the time when n^* is dequeued, the queue contains another node n such that $n[\mathsf{state}] \in OPT(F)$ and $n[\mathsf{cost}] = \mathrm{cost}^*(n[\mathsf{state}])$. On one hand, $n[\mathsf{cost}] \leq C^*$ where C^* is the cost of P. On the other hand, $n^*[\mathsf{cost}] \leq n[\mathsf{cost}]$ since n^* is dequeued before n. Combining both inequalities, $n^*[\mathsf{cost}] \leq C^*$ which means that the path found by IW_Φ of cost $n^*[\mathsf{cost}]$ is an optimal trajectory for P.

The second claim in the statement of the theorem is implied by the first since $OPT(F)$ is a cost-envelope by Theorem 25. Finally, for the complexity bounds, notice that the number of nodes expanded by IW_Φ is bounded by the number of different feature valuations, which is $\mathcal{O}(N^\ell)$ if the features in Φ are linear and the number of numerical features in Φ is ℓ. Thus, the plan length is $\mathcal{O}(N^\ell)$ while the number of generated nodes is $\mathcal{O}(bN^\ell)$, if b bounds the branching factor. On the other hand, the operations on the hash table can be done in constant time on a perfect hash. Hence, IW_Φ runs in $\mathcal{O}(bN^\ell)$ time and space. □

Notice that IW_Φ provides complexity bounds for the solvability of classes of problems, **independently of the underlying structure of states.** That is, IW_Φ does not know about atoms at all, only about feature valuations. Two encodings that are different syntactically but equivalent semantically will yield the same behaviour in IW_Φ, like two encodings of Blocksworld, one using *clear* as a primitive predicate and one using it as a derived predicate.

The set Φ of features in Theorem 29 does not need to be the set of features on which the policy π is defined. If this is so, however, and if F is the set of feature valuations reached by π, and $\mathrm{Reachable}(\pi, P)$ is the set of states reachable by using π in P, IW_Φ is optimal provided that the policy π is optimal and $OPT(F) = \mathrm{Reachable}(\pi, P)$:

Theorem 30. *Let π be a rule-based policy defined over the features in Φ which is **optimal** for P, and let F be the set of feature valuations reached by π in P. If $OPT(F) = \mathrm{Reachable}(\pi, P)$, then $OPT(F)$ is a closed π-envelope, and thus IW_Φ finds an optimal plan for P.*

Proof. By Theorem 29, it is enough to show that $OPT(F)$ is a closed π-envelope. Clearly, the initial state s_0 belongs to $OPT(F)$. If s is a state in $OPT(F)$, then 1) there is a π-trajectory for s (as s is π-reachable), 2) there is a π-transition (s, s') (as π solves P), and 3) s' belongs to $OPT(F)$ (as s' is reached by π). Closedness is direct by (1). □

If different states have different feature valuations, IW_Φ reduces to a breadth-first search, as only nodes for duplicate states are pruned. The interesting uses of the theorem however are on settings where the number of possible feature valuations is exponentially smaller than the number of states, as illustrated in the next two examples.

Example 10 (IW_Φ search on $\mathcal{Q}_{Clear}$)**.**

- Let us consider the policy π for $\mathcal{Q}_{Clear}$ previously defined in Example 4 over the set of features $\Phi = \{H, n\}$. Fix a problem P in $\mathcal{Q}_{Clear}$, and let F be the set of feature

valuations reached by π on P. The policy π is optimal and $OPT(F) = \text{Reachable}(\pi, P)$. By Theorem 30, IW_Φ finds an optimal plan for problem P. Notice that if P has N blocks, there are $2N$ different feature valuations, but an exponential number of configurations for the N blocks.

Example 11 (The Marbles domain)**.**

- The Marbles domain M involves boxes and marbles. Boxes can be on the table and marbles inside boxes. Problems are specified with atoms $\mathit{ontable}(b)$ to tell that box b is on the table, and $\mathit{in}(r, b)$ to tell that marble r is in box b. The goal is to remove all boxes from the table, where a box can be removed only if it is empty. Marbles thus must be removed from boxes one at a time, in no specific order. The collection of all problems over the Marbles domain is denoted by $\mathcal{Q}_M$, while $\mathcal{Q}_{M_1} \subseteq \mathcal{Q}_M$ denotes the class of such problems with exactly one box.

- Marbles is not a STRIPS domain as the goal must be specified with negative literals; i.e., as $\bigwedge_b \neg \mathit{ontable}(b)$, where the conjunction is over all boxes b in the problem.

- Yet there is a simple optimal policy π for $\mathcal{Q}_M$ defined over the set $\Phi = \{n, m\}$ of numerical features where n counts the number of boxes still on the table, and m counts the number of marbles in the "first box" among those still on the table, where a static ordering of the boxes is assumed. A general optimal policy π over Φ can be expressed as follows:

$$\{m > 0\} \mapsto \{m\!\downarrow\}\,,$$
$$\{m = 0, n > 0\} \mapsto \{n\!\downarrow, m?\}\,.$$

  The first rule says to remove a marble from the first box on the table when such a box is not empty, while the second rule says to take an action that decreases the number of boxes on the table. The policy π solves any problem in $\mathcal{Q}_M$ or $\mathcal{Q}_{M_1}$ optimally, as any optimal plan must execute a number of actions that is equal to the total number of marbles in all boxes plus the number of boxes.

- The policy π is optimal for $\mathcal{Q}_M$ and $OPT(F) = \text{Reachable}(\pi, P)$. By Theorem 30, IW_Φ finds an optimal plan for P in $\mathcal{O}(N^3)$ time, where N is the number of atoms in P, since the features in Φ are linear, and the branching factor in P is bounded by N. The total number of states, however, is exponential in N.

6. Serializations

The problem of subgoal structure is critical in classical planning, hierarchical planning, and reinforcement learning although in most cases the problem has not been addressed formally. We draw on the language for general policies to express decompositions into subproblems, and on the notion of width for expressing and evaluating such decompositions and the subgoal structures that result. We start with the notion of *serializations* which are defined semantically as binary relations on states that are acyclic.

Definition 31 (Serializations). *A **serialization** over a collection of problems $\mathcal{Q}$ is a binary relation '$\prec$' over the states in $\cup_{P\in\mathcal{Q}} states(P)$ that is* acyclic in $\mathcal{Q}$, *meaning that there is no set $\{s_0, s_1, \ldots, s_n\}$ of **reachable states** in P, where s_0 is the initial state, such that $s_{i+1} \prec s_i$ for $i = 0, \ldots, n-1$, and $s_j \prec s_n$ for some $0 \leq j \leq n$.*

For binary relations '$\prec$' that express serializations, the notation $s' \prec s$ expresses in infix form that the state pair (s, s') is in '$\prec$'. There is no assumption that the state pair (s, s') is a state transition; namely, that s' is one step from s, and the meaning of $s' \prec s$ is that s' is a possible subgoal state from state s. There is also no assumption that $s' \prec s$ implies that the state s' must be reachable from s, nor that $\prec$ must be non-empty. Acyclicity, on the other hand, avoids looping behavior on subproblems.

A serialization '$\prec$' over $\mathcal{Q}$ splits a problem P in $\mathcal{Q}$ into *subproblems.* For a reachable state s in P, the subproblem $P_\prec[s]$ is like P but with two changes: the initial state is s, and the goal states are the states s' such that s' is a goal state of P, or $s' \prec s$.

Definition 32 (Subproblems). *Let $\prec$ be a serialization over a class $\mathcal{Q}$, and let P be a problem in $\mathcal{Q}$. The class of subproblems induced by $\prec$ on P, denoted by $P_\prec$, is the* smallest class *that satisfies:*

1. *$P_\prec[s_0]$ is in $P_\prec$ for the initial state s_0 of P, and*
2. *$P_\prec[s']$ is in $P_\prec$ if $P_\prec[s]$ in $P_\prec$, $s' \prec s$, and s' is not a goal state. If $cost(P_\prec[s]) = 1$, however, s' must also be a successor state of s.*

In case 2 above, the subproblem $P_\prec[s]$ is said to induce the subproblem $P_\prec[s']$. For this to hold, s' must be a possible subgoal from s and not be a top goal state of P (else the subproblem $P_\prec[s']$ would not need a plan at all). It is not required for s' to be a closest subgoal state to s except when the subproblem $P_\prec[s]$ can be solved in one step.

Intuitively, a serialization is "good" if it results in subproblems that have small, bounded width that can be solved greedily on the way to the goal.

Definition 33 (Serialized width). *Let '$\prec$' be a serialization over a class of problems $\mathcal{Q}$, and let $P \in \mathcal{Q}$. Then,*

1. *The **(serialized) width** of P, denoted as $w_\prec(P)$, is the minimum non-negative integer k that bounds the width $w(P_\prec[s])$ of all the subproblems $P_\prec[s]$ in $P_\prec$.*
2. *The **(serialized) width** of $\mathcal{Q}$, denoted as $w_\prec(\mathcal{Q})$, is the minimum non-negative integer k that bounds the serialized width $w_\prec(P)$ of the problems P in $\mathcal{Q}$.*

Starting from a problem P in $\mathcal{Q}$, a serialization may lead to state s if the subproblem $P_\prec[s]$ belongs to the subclass $P_\prec$ of subproblems induced by $\prec$. Hence, if for a dead-end state s, the subproblem $P_\prec[s]$ belongs to $P_\prec$, by definition, $w(P_\prec[s]) = \infty$ and thus $w(\mathcal{Q}) = \infty$ as well.

Interestingly, serializations of zero width are policies, and vice versa, policies are serializations of zero width.

Theorem 34 (Zero-width serializations and policies). *Let $\mathcal{Q}$ be a class of problems, and let $\prec$ be a binary relation on $\cup_{P\in\mathcal{Q}} states(P)$. Then, $\prec$ is a serialization of zero width for $\mathcal{Q}$ iff $\prec$ is a policy that solves $\mathcal{Q}$.*

Proof. In this proof, we write $s \prec s'$ to denote that the pair (s, s') is in the relation $\prec$, either when $\prec$ denotes a serialization or a policy.

($\Rightarrow$) Assume that $\prec$ is a serialization of zero width for $\mathcal{Q}$. Clearly, $\prec$ is a policy as it is a binary relation on $\cup_{P \in \mathcal{Q}}$states(P). It remains to show that for any problem P in $\mathcal{Q}$, every maximal $\prec$-trajectory seeded at the initial state s_0 of P is goal reaching.

Let $\tau = s_0, s_1, \ldots, s_n, \ldots$ be one such trajectory; i.e., $s_{i+1} \prec s_i$ for $i \geq 0$. Since states(P) is a finite set and $\prec$ is acyclic in P, τ must be of finite length. Let us assume that it ends at s_n. If s_n is not a goal state, the subproblem $P_\prec[s_n]$ belongs to $P_\prec$ by Definition 32. Then, $w(P_\prec[s_n]) = 0$ since $\prec$ is of zero width. This means that there is a successor state s' of s_n such that $s' \prec s_n$. Hence, τ is not a maximal trajectory contradicting the assumption. Therefore, s_n must be a goal state.

($\Leftarrow$) Assume that $\prec$ is a policy that solves $\mathcal{Q}$. We first show that $\prec$ is acyclic in $\mathcal{Q}$, and then that its width is zero.

Let P be a problem in $\mathcal{Q}$, and let $s_0, s_1, \ldots, s_n$ be a set of reachable states in P such that $s_0 \prec s_1 \prec \cdots \prec s_n$. We need to show that there is no index $0 \leq j \leq n$ such that $s_n \prec s_j$. Let us suppose that there is such an index j. Then, the trajectory $\tau = s_0, s_1, \ldots, s_j, \ldots, s_n, s_j$ would be a maximal $\prec$-trajectory in P. As τ is not goal reaching, $\prec$ does not solve $\mathcal{Q}$ as assumed. Therefore, $s_n \prec s_j$ does not hold.

If $P_\prec[s]$ is a subproblem in the class $P_\prec$, the state s is reachable through a $\prec$-trajectory from the initial state s_0 of P; i.e., there is a $\prec$-trajectory $s_0, s_1, \ldots, s_n$ with $s_n = s$. Indeed, if $P_\prec[s]$ is in $P_\prec$, there is a *state sequence* $s_0, s_1, \ldots, s_n$ such that $s_n = s$, and the subproblem $P_\prec[s_i]$ induces $P_\prec[s_{i+1}]$ for $0 \leq i < n$. However, $\prec$ is only defined on state transitions as it is a policy. Therefore, such a state sequence is a state trajectory.

Let us compute the width of a subproblem $P_\prec[s]$ in $P_\prec$. By the claim, there is a $\prec$-trajectory $s_0, s_1, \ldots, s_n$ such that $s_n = s$, and by definition of subproblem, s_n is not a goal state. Then, since $\prec$ is a policy that solves $\mathcal{Q}$, there is a state transition (s, s') such that $s \prec s'$. Hence, $w(P_\prec[s]) = 0$, which implies $w_\prec(P) = 0$ and $w_\prec(\mathcal{Q}) = 0$. □

Example 12 (Zero-width serializations for different classes)**.**

- The rule-based policies given for the classes $\mathcal{Q}_{Clear}$, $\mathcal{Q}_{Grid}$, $\mathcal{Q}_M$, and $\mathcal{Q}_D$ represent serializations of zero width, as determined by Theorem 34.

An important property of serializations of bounded width is that they permit the decomposition of a problem into subproblems which can be solved greedily in polynomial time without the need to backtrack. This property is exploited by the algorithm SIW$_\prec$ shown in Fig. 4.

Theorem 35 (Completeness of SIW$_\prec$)**.** *Let '$\prec$' be a serialization for a class $\mathcal{Q}$ of problems. If $w_\prec(\mathcal{Q}) \leq k$, then any problem P in $\mathcal{Q}$ can be solved by SIW$_\prec$. Moreover, the IW search in line 5 of SIW$_\prec$ takes $\mathcal{O}(bN^{2k-1})$ time and $\mathcal{O}(bN^k)$ space, where N is the number of atoms, and b bounds the branching factor in P.*

Proof. Let s_0 be the initial state of a problem P in $\mathcal{Q}$, and let $\tau = s_0, s_1, \ldots, s_n$ be a **state sequence** in P where all states, except perhaps s_n, are non-goal states. We say that τ is

Algorithm 4: SIW$_\prec$ Search

1: **Input:** Testable serialization '$\prec$' for class $\mathcal{Q}$
2: **Input:** Planning problem P in class $\mathcal{Q}$
3: Initialize state s to initial state s_0 for P
4: While s is not a goal state of P:
5: Do an IW search from s to find s' that is either a goal state or $s' \prec s$
(i.e., the goal test in line 9 of IW(k) is augmented with $s' \prec s$ where s' is the state for the dequeued node n in line 7)
6: If s' is found, set $s \leftarrow s'$. Else, return **FAILURE** (Serialized width of P is ∞)
7: Return the path from s_0 to the goal state s (Solution found)

Figure 4: SIW$_\prec$ solves a problem P by using the serialization to decompose P into subproblems, each one that is solved with an IW search. Testable serialization means that there is an algorithm for testing $s' \prec s$ for any pair of states. The completeness of SIW$_\prec$ is given in Theorem 35.

a $\prec$-sequence iff for each index $0 \leq i < n$, the state s_{i+1} is reachable from the state s_i and $s_{i+1} \prec s_i$, but if the state s_{i+1} is not a successor of s_i, then s_i has no successor s' such that $s' \prec s_i$. Observe that if τ is a $\prec$-sequence, then a simple inductive argument shows that the subproblem $P_\prec[s_i]$ induces the subproblem $P_\prec[s_{i+1}]$, $0 \leq i < n$. If s_n is a goal state, the sequence is called a $\prec$-solution.

If $\tau = s_0, s_1, \ldots, s_n$ is a $\prec$-sequence for P, there is a $\prec$-solution τ' for P that extends τ. Indeed, if s_n is a goal state, τ is already a $\prec$-solution. Otherwise, the subproblem $P_\prec[s_n]$ belongs to $P_\prec$. Since $w_\prec(P) \leq k$, there is a state s_{n+1} reachable from s_n that is either a goal state, or $s_{n+1} \prec s_n$; i.e., the sequence τ, s_{n+1} is a $\prec$-sequence. Iterate until finding an extension τ' of τ that ends in a goal state, which can be done because $\prec$ is acyclic, and the number of states in P is finite.

It is easy to see that at the start of each iteration of the loop, SIW$_\prec$ has discovered a $\prec$-sequence $\tau = s_0, s_1, \ldots, s_n$ that ends at the current state $s_n = s$. Hence, since $\prec$ is acyclic, the loops eventually ends. The time and space bounds for each call of IW in line 5 of SIW$_\prec$ follow directly from Theorem 4. □

However, even if the subproblems are solved greedily and in polynomial time by IW, the total number of calls to IW, and hence the total running time of SIW$_\prec$, cannot be bounded without extra assumptions on the structure of the serialization. Indeed, there are serializations that split a problem into an exponential number of subproblems, like the Hanoi example below. However, once we move to *serializations* expressed by means of rules akin to those used to express policies, we will be able to provide conditions and bound the running time of SIW$_\prec$.

Example 13 (The Hanoi domain)**.**

- $\mathcal{Q}_{Hanoi}$ is the class of Towers of Hanoi problems involving 3 pegs, numbered from 0 to 2, and any number of disks, where the initial and goal states correspond to single towers

at different pegs, respectively. Recently, Liu et al. (2023) refer to a general strategy that solves problems of moving a single tower from peg 0 to peg 2:

> *Alternate actions between the smallest disk and a non-smallest disk. When moving the smallest disk, always move it to the left. If the smallest disk is on the first pillar, move it to the third one. When moving a non-smallest disk, take the only valid action.*

- This strategy can be expressed as a rule-based policy using three Boolean features $p_{i,j}$, $1 \leq i < j \leq 3$, that are true if the top disk at peg i is smaller than the top disk at peg j. However, in order to account for the alternation of movements, it must be assumed that the planning encoding adds an extra atom e that is true initially, and that flips with each movement. Provided then with a Boolean feature q that tracks the value of e, a general policy for Hanoi can be expressed with the following set R of rules over the features $\Phi = \{q, p_{1,2}, p_{1,3}, p_{2,3}\}$:

  % Movements of the smallest disk

  $\{q, p_{1,2}, p_{1,3}\} \mapsto \{\neg q, p_{1,2}?, \neg p_{1,3}, \neg p_{2,3}\}$ (Move the smallest from peg 1 to peg 3)

  $\{q, \neg p_{1,2}, p_{2,3}\} \mapsto \{\neg q, p_{1,2}, p_{1,3}, p_{2,3}?\}$ (Move the smallest from peg 2 to peg 1)

  $\{q, \neg p_{1,3}, \neg p_{2,3}\} \mapsto \{\neg q, \neg p_{1,2}, p_{1,3}?, p_{2,3}\}$ (Move the smallest from peg 3 to peg 2)

  % Movements of the other disk

  $\{\neg q, p_{1,2}, p_{1,3}, p_{2,3}\} \mapsto \{q, \neg p_{2,3}\}$ (Move the other from peg 2 to peg 3)

  $\{\neg q, p_{1,2}, p_{1,3}, \neg p_{2,3}\} \mapsto \{q, p_{2,3}\}$ (Move the other from peg 3 to peg 2)

  $\{\neg q, \neg p_{1,2}, p_{1,3}, p_{2,3}\} \mapsto \{q, \neg p_{1,3}\}$ (Move the other from peg 1 to peg 3)

  $\{\neg q, \neg p_{1,2}, \neg p_{1,3}, p_{2,3}\} \mapsto \{q, p_{1,3}\}$ (Move the other from peg 3 to peg 1)

  $\{\neg q, p_{1,2}, \neg p_{1,3}, \neg p_{2,3}\} \mapsto \{q, \neg p_{1,2}\}$ (Move the other from peg 1 to peg 2)

  $\{\neg q, \neg p_{1,2}, \neg p_{1,3}, \neg p_{2,3}\} \mapsto \{q, p_{1,2}\}$ (Move the other from peg 2 to peg 1)

- In problems with an odd, respectively even, number of disks, the policy moves a single tower at peg i to peg j, where $j = (i - 1 \mod 3)$, resp. $j = (i + 1 \mod 3)$. By moving the smallest disk to the right rather than to the left, the target peg changes to $j = (i+1 \mod 3)$, resp. $j = (i - 1 \mod 3)$.

- The policy π_R defined by rules in R is thus general for the class $\mathcal{Q}_{HanoiOdd}$ that contains the problems with an odd number of disks, initial situation with a single tower at peg 0, and goal situation with a single tower at peg 2. A general policy for $\mathcal{Q}_{Hanoi}$ can be obtained by considering additional Boolean features that tell the parity of the number of disks, and the pegs for the initial and final towers.

- The policy π_R defines a serialization of zero width by Theorem 34. This serialization splits a problem P in $\mathcal{Q}_{HanoiOdd}$ into an exponential number of subproblems as $2^n - 1$ steps are needed to solve a Hanoi problem with n disks. The algorithm SIW_R solves any problem P in $\mathcal{Q}_{HanoiOdd}$, but not in polynomial time.

6.1 Rule-Based Serializations: Sketches

As with policies, the binary relations that encode serializations can be compactly represented by means of rules. The syntax of the rules is exactly the syntax of policy rules, and the only difference is in the semantics of the rules where state pairs (s, s') are not limited to state transitions:

Definition 36 (Sketches)**.** *Let $\mathcal{Q}$ be a collection of problems, let Φ be a set of features for $\mathcal{Q}$, and let R be a set of rules over Φ. The rules in R define the* ***binary relation*** *$\prec_R$ over the states in $\cup_{P\in\mathcal{Q}} states(P)$ given by $s' \prec_R s$ iff the state pair (s, s') is compatible with some rule in R. If $\prec_R$ is* ***acyclic in $\mathcal{Q}$,*** *$\prec_R$ is a serialization over $\mathcal{Q}$.*

The rules that define serializations are called **sketch rules,** and sets of such rules are called **sketches**. The sketch width of $\mathcal{Q}$ given a sketch R is the serialized width of $\mathcal{Q}$ under the serialization $\prec_R$ defined by R.

Definition 37 (Sketch width)**.** *Let R be a set of rules that define a serialization $\prec_R$ over a class $\mathcal{Q}$ of problems. The* sketch width *of R over $\mathcal{Q}$, denoted by $w_R(\mathcal{Q})$, is $w_R(\mathcal{Q}) \doteq w_{\prec_R}(\mathcal{Q})$.*

A rule-based policy π that solves $\mathcal{Q}$ is a sketch R for $\mathcal{Q}$ of zero width:

Theorem 38 (Rule-based policies and sketches)**.** *Let R be a set of rules defined in terms of a set of features Φ for a class $\mathcal{Q}$ of problems. Then, R is a rule-based policy that solves $\mathcal{Q}$ iff R is a sketch of zero width for $\mathcal{Q}$.*

Proof. Straightforward using Theorem 34. □

6.2 Algorithms

If R is a sketch of bounded width over a class $\mathcal{Q}$, the problems in $\mathcal{Q}$ can be solved by the SIW_R algorithm, shown in Fig. 5, where $s' \prec_R s$ is tested by checking if some rule in R is compatible with the state pair (s, s'). However, to bound the complexity of SIW_R, a bound in the total number of subproblems that need to be solved is needed. A simple way to bound such a number is to require that the subgoal states $s_0, \ldots, s_n$ in state sequences compatible with a sketch R have different feature valuations:

Definition 39 (Feature-acyclic sketches)**.** *Let $\mathcal{Q}$ be a class of problems, and let R be a set of rules for $\mathcal{Q}$ defined on a set Φ of features that define a binary relation $\prec_R$. The relation $\prec_R$, or simply R, is said to* ***feature-acyclic*** *over $\mathcal{Q}$ if it is so for each problem P in $\mathcal{Q}$, where the latter means that there is no set $\{s_1, s_2, \ldots, s_n\}$ of reachable states in P such that $s_1 \prec_R s_2 \prec_R \cdots \prec_R s_n$, and $f(s_i) = f(s_j)$ for some $1 \leq i < j \leq n$.*

Clearly, if R is feature-acyclic over $\mathcal{Q}$, then $\prec_R$ is (state) acyclic over $\mathcal{Q}$, and hence $\prec_R$ is a serialization, and R is a sketch. The complexity bound for algorithm SIW_R follows:

Theorem 40 (Completeness of SIW_R)**.** *Let R be a* ***feature-acyclic*** *sketch for a class $\mathcal{Q}$ of problems of width bounded by k. SIW_R solves any problem P in $\mathcal{Q}$ in polynomial time (exponential only k, not in the size of P). In particular, if the features are linear, P is*

Algorithm 5: SIW_R Search

1: **Input:** Sketch R that defines relation $\prec_R$
2: **Input:** Planning problem P in collection $\mathcal{Q}$
3: Initialize state s to initial state s_0 for P
4: While s is not a goal state of P:
5: Do an IW search from s to find s' that is either a goal state or $f(s') \prec_R f(s)$ (i.e., the goal test in line 9 of IW(k) is augmented with $f(s) \prec_R f(s')$ where s' is the state for the dequeued node n in line 7)
6: If s' is found, set $s \leftarrow s'$. Else, return FAILURE (Serialized width of P is ∞)
7: Return the path from s_0 to the goal state s (Solution found)

Figure 5: SIW_R is $\text{SIW}_\prec$ with the serialization $\prec_R$ induced by the sketch R. The completeness and complexity of SIW_R is given in Theorem 40.

solved by SIW$_R$ in $\mathcal{O}(N^\ell(N^{k+1} + bN^{2k-1}))$ time and $\mathcal{O}(bN^k)$ space, producing a plan of length $\mathcal{O}(N^{\ell+k})$, where N is the number of atoms in P, b bounds the branching factor in P, and ℓ is the number of ***numerical features*** *in Φ.*

Proof. The SIW_R algorithm is the $\text{SIW}_\prec$ algorithm that uses the serialization '$\prec_R$' induced by the sketch R. Hence, by Theorem 35, SIW_R solves any problem P in $\mathcal{Q}$, and each call to IW in line 5 of SIW_R takes $\mathcal{O}(bN^{2k-1})$ time and $\mathcal{O}(bN^k)$ space.

As $\prec_R$ is feature-acyclic, the number of subproblems to solve is bounded by the maximum number of feature valuations that can appear when solving P. In the case of linear features, this number is $\mathcal{O}(N^\ell)$. For each expanded state in each call to IW, the value of the features are computed in $\mathcal{O}(|\Phi|N) = \mathcal{O}(N)$ time. Thus, the total running time of SIW_R is $\mathcal{O}(N^\ell(N^{k+1} + bN^{2k-1}))$.

For the space required by SIW_R, since the solutions to the subproblems produced by IW do not need to be stored, the space complexity of SIW_R is is the space complexity of the IW calls; namely, $\mathcal{O}(bN^k)$. The length of the overall plan, however, is bounded by the number of subproblems times their maximum possible lengths as $\mathcal{O}(N^{\ell+k})$. □

Example 14 (Sketches for the Delivery domain)**.**

Table 1 contains different sets of rules over the set $\Phi = \{H, p, t, u\}$ of features for the Delivery domain. For each such set, the table indicates whether the set is feature-acyclic, and contains the sketch width for the classes $\mathcal{Q}_{D_1}$ and $\mathcal{Q}_D$. The width is only specified for sets that are acyclic. We briefly explain the entries in the table without providing formal proofs, but all the details can be easily filled in with the results in the paper.

- R_0 is the empty sketch whose width is the same as the plain width.
- The rule $\{H\} \mapsto \{\neg H, p?, t?\}$ in R_1 does not help in initial states that do not satisfy H, and hence the width remains 2 and ∞ for $\mathcal{Q}_{D_1}$ and $\mathcal{Q}_D$, respectively.
- The rule $\{\neg H\} \mapsto \{H, p?, t?\}$ in R_2 says that a state s where $\neg H$ holds can be "improved" by finding a state s' where H holds, while possibly affecting p, t, or both. This rule splits every problem P in $\mathcal{Q}_{D_1}$ into two subproblems: achieve H first and then the goal, reducing the sketch width of $\mathcal{Q}_{D_1}$ to 1.

Rule set	Acyclic	Sketch width $\mathcal{Q}_{D_1}$	Sketch width $\mathcal{Q}_D$
$R_0 = \{\,\}$	✓	2	∞
$R_1 = \{\{H\} \mapsto \{\neg H, p?, t?\}\}$	✓	2	∞
$R_2 = \{\{\neg H\} \mapsto \{H, p?, t?\}\}$	✓	1	∞
$R_3 = R_1 \cup R_2$	✗	—	—
$R_4 = \{\{u > 0\} \mapsto \{u\!\downarrow, H?, p?, t?\}\}$	✓	2	2
$R_5 = R_2 \cup R_4$	✓	1	1
$R_6 = \{\{\neg H, p > 0\} \mapsto \{p\!\downarrow, t?\}\}$	✓	2	∞
$R_7 = \{\{H, t > 0\} \mapsto \{t\!\downarrow\}\}$	✓	2	∞
$R_8 = R_2 \cup R_4 \cup R_6 \cup R_7$	✓	0	0

Table 1: Different sketches for the Delivery domain, one rule set per line. The table shows whether each rule set is feature-acyclic and also upper bounds the width for sketch for the classes $\mathcal{Q}_{D_1}$ and $\mathcal{Q}_D$ of Delivery problems. The rule set R_3 is not a proper sketch as it is not acyclic; hence, the entries marked as '—'. For feature-acyclic sketches of bounded width, SIW_R solves any instance in the class in polynomial time.

- The rule set R_3 is not acyclic and thus not a proper sketch.
- The sketch R_4 decomposes problems using the feature u that counts the number of undelivered packages, reducing the width of $\mathcal{Q}_D$ to 2, but not affecting the width of $\mathcal{Q}_{D_1}$. The reduction occurs because each problem P in $\mathcal{Q}_D$ is split into subproblems, each one for delivering a single package, similar to the problems in $\mathcal{Q}_{D_1}$.
- R_5 combines the rules in R_2 and R_4. Each problem in $\mathcal{Q}_D$ is decomposed into subproblems, each one like a problem in $\mathcal{Q}_{D_1}$, and each problem in $\mathcal{Q}_{D_1}$ is further decomposed into two subproblems of width 1 each. The combined result is that the sketch width of $\mathcal{Q}_{D_1}$ and $\mathcal{Q}_D$ both get reduced to 1.
- The sketches R_6 and R_7 do not help to reduce the width for either class. The rule in R_6 generate subproblems of zero width until reaching a state where $\neg H$ and $p = 0$ holds, for which the remaining problem has width 2 or ∞ for either $\mathcal{Q}_{D_1}$ or $\mathcal{Q}_D$, respectively. R_5, on the other hand, does not help as the initial states do not satisfy H.
- Finally, the sketch R_8 yields a serialization of zero width, and hence a full policy, where each subproblem is solved in a single step.

6.3 Acyclicity and Termination

The notion of acyclicity appears in three places in our study. First, if a policy π is closed and acyclic in a problem P, then π solves P. Second, serializations must be acyclic, as otherwise, even if subproblems have small, bounded width, the $\text{SIW}_\prec$ procedure may get stuck in a cycle. Third, feature acyclicity has been used above to provide runtime bounds.

Interestingly, there are *structural* conditions on the set of rules R that ensure that the resulting binary relation on pairs of states (s, s') is **feature-acyclic** by virtue of the form of the rules and the features involved, *independently of the domain.*

This is the case, for example, if R only contains the rules $r_1 = \{\neg H\} \mapsto \{H, n\downarrow\}$ and $r_2 = \{H\} \mapsto \{\neg H\}$. A sequence of states $s_0, s_1, s_2, \ldots$ compatible with R cannot contain infinite state pairs (s_i, s_{i+1}) compatible with rule r_1, because such a rule requires feature n to decrease but n cannot decrease below zero and no rule allows n to increase. Then, since rule r_1 cannot be "applied" infinitely often, neither can rule r_2 which requires r_1 to restore the truth of the condition H. This analysis is *independent* of the underlying planning problem and the semantics of the features.

The notion of **termination** as captured by the Sieve algorithm for QNPs (Srivastava et al., 2011b; Bonet & Geffner, 2020) can be used to check, among other things, that a rule-based policy or sketch is feature-acyclic. Indeed, if R is a terminating set of rules over the features in Φ, as determined by Sieve, $s_0, s_1, s_2, \ldots$ is a state sequence compatible with R, and s_i and s_j, with $i < j$, are two states with identical valuation over the Boolean conditions defined by Φ (see below), then there is a numerical feature n such that its values satisfy $n(s_j) < n(s_i)$ and $n(s_k) \leq n(s_i)$ for $i \leq k \leq j$. This condition ensures that R is feature-acyclic, and thus that it is acyclic over any class $\mathcal{Q}$.

The Sieve algorithm, shown in Fig. 6, receives as input a directed and edge-labeled graph $G = \langle V, E, \ell \rangle$, where the edge labels $\ell(e)$ contain effects over Boolean and numerical features; i.e., expression of the form p, $\neg p$ and $p?$ for Boolean features p, and expressions of the form $n\downarrow$, $n\uparrow$, and $n?$ over numerical features n. Sieve iteratively computes the strongly connected components (SCCs) of a graph G', initially set to the input graph G, and removes edges from the graph until it becomes acyclic, or no more edges can be removed. The graph is **accepted** iff it becomes acyclic, otherwise is **rejected**. An edge e in a component C of G' can be removed if some feature n is decreased in e (i.e. $n\downarrow \in \ell(e)$), and is not increased in any other edge e' in the same component (i.e. $n\uparrow \notin \ell(e')$ and $n? \notin \ell(e')$).

The graph $G(R) = \langle V, E, \ell \rangle$ that is passed to Sieve as input is constructed from a set R of rules over a set Φ of features as follows. The vertices in V correspond to the $2^{|\Phi|}$ valuations v for the conditions p and $n = 0$ for the Boolean and numerical features p and n in Φ, and there is an edge (v, v') in E if the pair of valuations v and v' is compatible with some rule $C \mapsto E$ in R. A set of rules R is **terminating** iff Sieve accepts the graph $G(R)$. In our setting, this means the following:

Theorem 41 (Srivastava et al., 2011b, Bonet and Geffner, 2020)**.** *Let Φ be a set of features, and let R be a set of rules over Φ for a class $\mathcal{Q}$ of problems. If* Sieve *accepts $G(R)$, then the binary relation $\prec_R$ is feature-acyclic over $\mathcal{Q}$, and therefore, R is a sketch that defines a serialization $\prec_R$ for $\mathcal{Q}$.*

Proof. If Sieve accepts $G(R)$, $\tau = s_0, s_1, s_2, \ldots$ is a state sequence that is compatible with R, and the states s_i and s_j, $i < j$, have identical valuation over the Boolean conditions for Φ, then there must be a numerical feature n that is decremented in the path $\tau_{i,j} = s_i, s_{i+1}, \ldots, s_j$ and is not incremented in $\tau_{i,j}$ (Srivastava et al., 2011b; Bonet & Geffner, 2020). Therefore, $f(s_i) \neq f(s_j)$. This implies that there is no such state sequence τ that contains two different states s and s' such that $f(s) = f(s')$. □

Algorithm 6: Sieve

Input: Directed edge-labeled graph $G = \langle V, E, \ell \rangle$, where the labels contain feature effects
Output: Either accept or reject G
Initialize the graph $G' \leftarrow G$
Repeat
    Compute the SCCs of graph G'
    Choose SCC T and numerical feature n that is decreased but not increased in T; i.e.,
        – T contains some edge e such that $n{\downarrow} \in \ell(e)$, and
        – T contains no edge e' such that $n{\uparrow} \in \ell(e')$ or $n? \in \ell(e')$
    Remove the edges in T where n is decreased
until G' is acyclic or no such SCC exist
ACCEPT if G' is acyclic, **REJECT** otherwise

Figure 6: The Sieve algorithm takes as input a directed edge-labeled graph G that is either accepted or rejected. The graph G is the graph $G(R)$ constructed using the feature-based rules in a set R. If G is accepted, the binary relation on feature valuations induced by R is deemed as **terminating**, and thus R is feature-acyclic (cf. Theorem 41).

Example 15 (Sieve for different sketches for the Delivery domain)**.**

Let us consider the Delivery domain with the set of features $\Phi = \{H, p, t, u\}$ discussed in Example 6. We consider different sets of rules that define different input graphs for Sieve:

- First, let us consider the set R_3 of rules in Table 1 made of $\{H\} \mapsto \{\neg H, p?, t?\}$ and $\{\neg H\} \mapsto \{H, p?, t?\}$. The graph $G(R_3)$ contains many cycles as the first rule connects all nodes in which H holds to all nodes in which $\neg H$ holds, and the second rule does the opposite. As no edge is labeled with a decrement, Sieve cannot remove any edge, and thus the graph is **rejected.**

- Consider now the set R_5 in Table 1 with $\{\neg H\} \mapsto \{H, p?, t?\}$ and $\{u > 0\} \mapsto \{u{\downarrow}, H?, p?, t?\}$. In the graph $G(R_5)$, the numerical feature u is decreased in some edges, but no edge contains $u{\uparrow}$ or $u?$ in its label. Hence, Sieve removes all edges that contain $u{\downarrow}$ in its label. This renders the graph acyclic as the only edges left connect nodes where $\neg H$ holds to nodes where H holds. Sieve **accepts** the input graph $G(R_5)$, and thus R_5 can be used as an sketch for $\mathcal{Q}_D$ by Theorem 41.

- Finally, let us consider the policy π given in Example 6. Figure 7 shows (part of) the graph $G(\pi)$ that is given to Sieve. The graph contains three strongly connected components C_1, C_2, and C_3, with C_1 being the only one with cycles. In C_1, the feature u is decreased but not increased. Sieve then removes all edges whose label contains u ↓, splitting C_1 into the components $\{C_i : i = 4, 5 \ldots, 8\}$; see caption of Fig. 7. Of these components, only C_4 and C_8 contain cycles. In C_4 (resp. C_8), all edges are labeled with $\{p{\downarrow}, t?\}$ (resp. $\{t{\downarrow}\}$). Hence, Sieve removes all edges from C_4 and C_8, leaving the graph acyclic. By Theorem 41, π defines a feature-acyclic sketch for $\mathcal{Q}_D$.

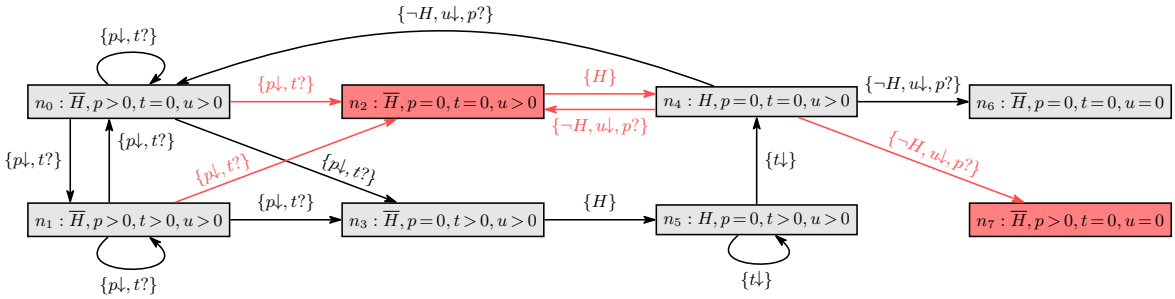

Figure 7: Relevant part of the graph $G(\pi)$ that is passed to Sieve for the policy π for the Delivery domain that contains the rules $\{\neg H, p>0\} \mapsto \{p\downarrow, t?\}$, $\{\neg H, p=0\} \mapsto \{H\}$, $\{H, t>0\} \mapsto \{t\downarrow\}$, and $\{H, t=0, u>0\} \mapsto \{\neg H, u\downarrow, p?\}$. Red nodes and edges stand for nodes and transitions in the policy graph that do not exist in instances of Delivery. $G(\pi)$ has three strongly connected components $C_1 = \{n_0, n_1, \ldots, n_5\}$, $C_2 = \{n_6\}$, and $C_3 = \{n_7\}$. In C_1, the numerical feature u is decreased but not increased. Sieve then removes all the edges whose label contains $u\downarrow$, resulting in a graph where C_1 is split into the components $C_4 = \{n_0, n_1\}$, $C_5 = \{n_2\}$, $C_6 = \{n_3\}$, $C_7 = \{n_4\}$, and $C_8 = \{n_5\}$. Of these components, only C_4 and C_8 contain cylces. C_8 contains only one edge with label $\{t\downarrow\}$ that is removed by Sieve. C_4 contains two nodes and four edges, all labeled with $\{p\downarrow, t?\}$. Sieve removes all such edges, splitting C_4 into two acyclic and singleton components, and accepts $G(\pi)$.

7. Summary of Results and Meaning

In this section, we summarize the main ideas and results of the paper. For simplicity, we do not restate the conditions of the theorems in full and focus instead on their meaning and the story that they reveal. While the work builds on an earlier paper (Bonet & Geffner, 2021), most of the results are new and convey a simpler, more meaningful narrative. Three key changes are: a slightly more general and convenient definition of admissibility based on *sets* of tuples and not sequences, the new notion of envelopes, and the definition of both policies and serializations as binary relations on states, expressed syntactically by means of rules. In addition, serializations are no longer assumed to be transitive relations. As usual, P is a planning instance from a class $\mathcal{Q}$, π is a general policy, T is a set atom tuples over P, and T^k denotes the set of conjunctions of up to k atoms. A summary of the main theorems above and their meaning follows:

- Theorems 4–7. If T is admissible, IW(T) finds an optimal plan and $w(P) \leq \text{size}(T)$. If $w(P) \leq k$, IW(k) finds an optimal plan, and IW finds a (not necessarily optimal) plan.
- Theorem 22: T admissible iff $OPT(T)$ is a cost-envelope.
- Theorem 23: IW(T) is optimal if T contains T' such that $OPT(T')$ is a cost-envelope, and IW(k) is optimal if such T' is contained in T^k.

Meaning. If T is admissible and hence a cost envelope, P is solved optimally by IW(T) which expands up to $|T|$ nodes and also by IW(T') if $T' \subseteq T$. The **width** of P, $w(P)$, is the minimum $size(T)$ of an admissible T, and IW(k) solves P optimally if $w(P) \leq k$. IW(k) is equivalent to IW(T^k).

- Theorem 25: $OPT(T)$ is a cost-envelope in P if it is a closed π-envelope of an optimal policy π for $\mathcal{Q}$, $P \in \mathcal{Q}$.
- Theorem 27 and corollaries: If $OPT(T)$ is a closed π-envelope of an optimal policy π, and $T \subseteq T^k$, IW(k) reaches the goal of P through an optimal π-trajectory.
- Theorem 28: $w(P) > k$ if every optimal plan for P contains a state outside $OPT(T^k)$.

Meaning. These results and the ones above explain why many standard planning domains have bounded width when goal atoms are considered (Lipovetzky & Geffner, 2012; Drexler et al., 2021). The reason is that such classes of problems admit general optimal policies π that can be "applied" in an instance P by just considering tuples of atoms of bounded size. If this bound is k, IW(k) finds the goal of P in polynomial time through a π-trajectory, without having to know π at all. The width of P is greater than k if no optimal plan "goes through" states that are all in $OPT(T^k)$.

- Theorem 29: IW_Φ is optimal if $OPT(F)$ is a closed π-envelope for some set F of feature valuations, and π is optimal.
- Theorem 30: $OPT(F)$ is a closed π-envelope if π is optimal and reaches all and only the states in $OPT(F)$.

Meaning. The IW_Φ procedure is like IW(T) but with the feature valuations over Φ playing the role of the atom tuples in T. The procedure is meaningful because in many tasks the number of possible feature valuations for a given Φ is exponentially smaller than the number of states (e.g., $\mathcal{Q}_{Clear}$ above). Two relevant questions are what sets of features Φ ensure that IW_Φ solves a problem (optimally) and whether the features used by a policy π that solves the problem do. The general answer to this last question is no: as shown in the example for Towers of Hanoi, one can define general policies in terms of a bounded and small set of Boolean features Φ, yet the length of any plan for Hanoi will grow exponentially with the number of disks. This simple combinatorial argument rules out Φ as a good set of features for IW_Φ in Hanoi, even when Φ supports a solution policy. The results above provide a more general argument that cuts in both ways. Namely, if no subset F of feature valuations results in $OPT(F)$ being a closed cost-envelope (as in Hanoi), then IW_Φ will not solve P optimally in general, and if F is one such set of feature valuations, then IW_Φ will solve P optimally.

The definition of policies and serialization as binary relation on states, expressed in compact form by the same type of rules, makes the relation between policies and serializations direct:

- Theorem 34: Policy π solves $\mathcal{Q}$ iff π is a serialization over $\mathcal{Q}$ of width zero (semantics).
- Theorem 38: Rule-based policy R solves $\mathcal{Q}$ iff R is a rule-based serialization (sketch) over $\mathcal{Q}$ of width zero (syntax).
- Theorem 35: A serialization of width k over $\mathcal{Q}$ implies that problems P in $\mathcal{Q}$ can be solved by solving subproblems of width bounded by k, greedily, with the SIW_R procedure. The number of subproblems to be solved, and hence the running time of SIW_R, however, is not necessarily polynomial (e.g., Hanoi).

- Theorem 40 and 41: A terminating rule-based serialization (sketch) of bounded width over $\mathcal{Q}$, implies that problems P in $\mathcal{Q}$ are solved in polynomial time by SIW_R.
- Theorem 41: Termination of rule-based serializations and policies can be checked in time exponential in the number of features by Sieve.

Meaning. A general policy is a general serialization of zero width; namely, a particular, type of serialization in which the subproblems can be solved greedily in a single step. Serializations of bounded width result in subproblems that can be solved greedily in polynomial time, while terminating rule-based serializations always result in a polynomial bounded of number of subproblems. The direct correspondence between policies and serializations is new and important, although it has not been recognized before. The reasons have been the lack of general and flexible formal accounts of serializations, compact languages for describing them, and width-like measures for bounding the complexity of subproblems. Policies have been formulated as binary relations on states and not as mapping from states to actions because actions do not generalize across instances. This choice has also helped to make the relation between policies and serializations more explicit. Finally, termination is a property of the set of rules and has nothing do with the class of problems. Introduced by Srivastava et al. (2011b), termination gives us *state acyclicity*, needed in the definition of serializations, and *feature-value acyclicity*, needed for the polynomial bound N^ℓ on the number of subproblems (provided that features are linear), where N is the number of atoms in the problem and ℓ is the number of numerical features used in the rules. If a terminating sketch has bounded width over a class $\mathcal{Q}$ of problems, the problems P in $\mathcal{Q}$ are solved greedily by SIW_R in polynomial time.

8. Extensions, Variations, and Limitations

We have presented a framework that accommodates policies and serializations, and have established relations between width and policies, on the one hand, and policies and serializations, on the other. Extensions, variations, and limitations of this framework are briefly discussed next.

Optimality and width. In the definition of an admissible set of tuples T and, hence, in the definition of width that follows, it is said that if an optimal plan σ for a tuple t in T is not an optimal plan for P, then an optimal plan for another tuple t' in T can be obtained by appending a single action to σ. A similar condition appears in the original definition of admissibility and width for sequences of tuples (Lipovetzky & Geffner, 2012). If the condition that "σ *is not an optimal plan* for P" is replaced by "σ *is not a plan* for P," the resulting definition of width (size of a min-size admissible set T) still guarantees that P is solved by $\text{IW}(k)$ if the width of P is bounded by k, but not that P is solved optimally by $\text{IW}(k)$. In the serialized setting, where optimal solutions of subproblems do not translate into optimal solution of problems, this relaxation of the definitions of admissibility (and width) makes sense, and it has been used for learning sketches of bounded width more effectively (Drexler et al., 2022).

Syntax of policy and sketch rules. The features and rules provide a convenient, compact, and general language for expressing policies and serializations, while the choice of

features, Boolean and numerical, follow the type of variables used in qualitative numerical planning problems (QNPs) for defining bounds and termination conditions (Srivastava et al., 2011b; Bonet & Geffner, 2020). In QNPs, it is critical that numerical variables change via "qualitative" increments and decrements, as reasoning with arbitrary numerical variables is undecidable (Helmert, 2002). Still, the restriction that numerical features n can *only* appear in effect expressions of the form $n\uparrow$, $n\downarrow$ or $n?$ is somewhat arbitrary, and other effect expressions like $\neg n\uparrow$, $\neg n\downarrow$, $n=0$, or $n>0$ can be accommodated with minor changes.

Non-deterministic sketches and policies. Policies and sketches are non-deterministic in the sense that many state transitions and pairs (s, s') can be compatible with a policy or sketch rule. If a policy π solves a problem P, it is because *all* π-transitions lead to the goal, and in the case of a bounded-width sketch, it is because the achievement of *any* such subgoal s' leads to the goal. This means that in a state s, one can pick any (policy or sketch) rule $C \mapsto E$ such that C is true in s, and move from s to any state s' that satisfies the rule. If the sketch has bounded width (a policy is a sketch of zero width), then at least one such rule exists. In certain cases, however, it is convenient to guarantee that there is one such state s' for *any* rule $C \mapsto E$ whose antecedent C is true at s, so that one can choose the rule to "apply" in a state without having to look ahead for the existence of such states s'. Sketches that have this additional property have been called *modular*, as the sketch rules are considered independently of each other. One can then talk about the width of a sketch rule $C \mapsto E$ as the maximum width of the problems with initial state s and goal states s' such that the state pair (s, s') satisfies the rule. Modular sketches are useful for learning *hierarchical policies*, where a sketch rule representing a class of problems of width greater than k is decomposed into sketch rules of width bounded by k, and so on iteratively, until sketch rules are obtained with width zero (Drexler et al., 2023).

Non-deterministic domains. The notion of width and the type of general policies considered are for deterministic planning domains. It is not yet clear how to extend the width notion to non-deterministic domains while preserving certain key properties like that bounded width problems can be solved in polynomial time, and that large classes of benchmark domains fall into such a class for suitable types of goals. The extension of general policies to non-deterministic domains, however, has been recently considered (Hofmann & Geffner, 2024).

9. Related Work

We review a number of related research threads.

Width, general policies, and sketches. This paper builds on prior works that introduced sketches (Bonet & Geffner, 2021), the language for expressing general policies in terms of features and rules (Bonet & Geffner, 2018), and the notion of width and IW search procedures (Lipovetzky & Geffner, 2012; Lipovetzky, 2021). Methods for learning general policies and sketches of bounded width have also been developed (Frances et al., 2021; Drexler et al., 2022), leading more recently to methods for learning hierarchical policies (Drexler et al., 2023).

General policies. The problem of learning general policies has a long history (Khardon, 1999; Martín & Geffner, 2004; Fern et al., 2006), and general policies have been formulated

in terms of first-order logic (Srivastava et al., 2011a; Illanes & McIlraith, 2019), and first-order regression (Boutilier et al., 2001; Wang et al., 2008; van Otterlo, 2012; Sanner & Boutilier, 2009). More recently, general policies for classical planning have been represented by neural nets and learned using deep learning methods (Groshev et al., 2018; Toyer et al., 2018; Bueno et al., 2019; Rivlin et al., 2020; Karia et al., 2022; Ståhlberg et al., 2023).

Hierarchical planning. While the subgoal structure of a domain is important for handcrafting effective hierarchical task networks, HTNs do not actually encode subgoal structures but general top-down solving strategies where (non-primitive) methods decompose into other methods (Erol et al., 1994; Nau et al., 1999; Georgievski & Aiello, 2015). Techniques for learning HTNs usually appeal to annotated traces that convey the intended decompositions (Hogg et al., 2008; Zhuo et al., 2009). Other methods for deriving hierarchical decompositions in planning include precondition relaxations (Sacerdoti, 1974) and causal graphs (Knoblock, 1994).

Hierarchical RL and intrinsic rewards. Hierarchical structures have also been used in reinforcement learning in the form of options (Sutton et al., 1999), hierarchies of machines (Parr & Russell, 1997) and MaxQ hierarchies (Dietterich, 2000). While this "control knowledge" is often provided by hand, a vast literature has explored techniques for learning them by considering "bottleneck states" (McGovern & Barto, 2001), "eigenpurposes" of the matrix dynamics (Machado et al., 2017), and width-based considerations (Junyent et al., 2021). Intrinsic rewards have also been introduced for improving exploration leading to exogenous rewards (Singh et al., 2010), and some authors have addressed the problem of learning intrinsic rewards. Interestingly, the title of one of the papers raises the question "What can learned intrinsic rewards capture?" (Zheng et al., 2020). The answer that follows from our setting is clean and simple: intrinsic rewards should capture the general, low-width subgoal structure of the domain. Lacking a language to talk about families of problems and subgoal structure, however, the answer to the question found in the RL literature is experimental and less crisp: learned intrinsic rewards are just supposed to speed up the convergence of (deep) RL.

Reward machines and sketches. A recent language for encoding subgoal structure in RL is based on reward machines (Icarte et al., 2018) and the closely related proposal of restraining bolts (De Giacomo et al., 2020). In these cases, the temporal structure of the (sub)goals to be achieved results in an automata which is combined with the system MDP to produce the so-called cross-product MDP. A number of RL algorithms for exploiting the known structure of the subgoal automata have been developed (Icarte et al., 2022) as well as algorithms for learning them (Toro Icarte et al., 2019). There is indeed a close relation between reward machines and sketches, as both convey subgoal structure, but there important differences too. First, reward machines usually encode the structure of explicit temporal goals (e.g., do X, then Y, and finally Z), while sketches encode the structure that is implicit in a reachability goal. Second, reward machines are defined in terms of additional propositional variables; sketches, in terms of state features that do not require cross-products. Third, sketches come with a theory of width that tells us where to split problems into subproblems and why. Finally, sketches come with a notion of termination that ensures that subgoaling cannot result in cycles.

10. Conclusions

We have established results that explain why many standard planning domains have bounded width, and have introduced a number of notions, like policy and cost envelopes that shed light on this relation and on the optimality and completeness of old and new IW-algorithms like IW(T) and IW$_\Phi$. We have also reformulated the semantic and syntactic notions of general policies and serializations slightly to make the relation between policies and serializations direct and clean: a policy is indeed a particular type of serialization that gives rise to subproblems of zero width which can be solved greedily on the way to the goal. This relation between policies and problem decompositions has not been articulated before. The paper is revised version of an earlier paper (Bonet & Geffner, 2021) that touched similar themes. The goal has been to make the results more transparent, useful, and meaningful.

Acknowledgements

The research of H. Geffner has been supported by the Alexander von Humboldt Foundation with funds from the Federal Ministry for Education and Research of Germany. The research has also received funding from the European Research Council (ERC), Grant agreement No. 885107, and Project TAILOR, Grant agreement No. 952215, under EU Horizon 2020 research and innovation programme, the Excellence Strategy of the Federal Government and the NRW Länder, and the Knut and Alice Wallenberg (KAW) Foundation under the WASP program.

References

Belle, V., & Levesque, H. J. (2016). Foundations for generalized planning in unbounded stochastic domains. In *Proc. KR*, pp. 380–389.

Bonet, B., & Geffner, H. (2018). Features, projections, and representation change for generalized planning. In *Proc. IJCAI*, pp. 4667–4673.

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

Bonet, B., & Geffner, H. (2015). Policies that generalize: Solving many planning problems with the same policy.. In *Proc. IJCAI*, pp. 2798–2804.

Bonet, B., & Geffner, H. (2021). General policies, representations, and planning width. In *Proc. AAAI*, pp. 1764–11773.

Boutilier, C., Reiter, R., & Price, B. (2001). Symbolic dynamic programming for first-order MDPs. In *Proc. IJCAI*, Vol. 1, pp. 690–700.

Bueno, T. P., de Barros, L. N., Mauá, D. D., & Sanner, S. (2019). Deep reactive policies for planning in stochastic nonlinear domains. In *AAAI*, Vol. 33, pp. 7530–7537.

Chenoweth, S. V. (1991). On the NP-hardness of blocks world. In *Proc. AAAI-91*, pp. 623–628.

De Giacomo, G., Iocchi, L., Favorito, M., & Patrizi, F. (2020). Restraining bolts for reinforcement learning agents. In *Proc. AAAI*, pp. 13659–13662.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research (JAIR)*, *13*, 227–303.

Drexler, D., Seipp, J., & Geffner, H. (2021). Expressing and exploiting the common subgoal structure of classical planning domains using sketches. In *Proc. KR*, pp. 258–268.

Drexler, D., Seipp, J., & Geffner, H. (2022). Learning sketches for decomposing planning problems into subproblems of bounded width. In *Proc. ICAPS*, pp. 62–70.

Drexler, D., Seipp, J., & Geffner, H. (2023). Learning hierarchical policies by iteratively reducing the width of sketch rules. In *Proc. KR*, pp. 208–218.

Erol, K., Hendler, J., & Nau, D. S. (1994). HTN planning: Complexity and expressivity. In *Proc. AAAI*, pp. 1123–1123.

Fern, A., Yoon, S., & Givan, R. (2006). Approximate policy iteration with a policy language bias: Solving relational Markov decision processes. *Journal of Artificial Intelligence Research (JAIR)*, *25*, 75–118.

Frances, G., Bonet, B., & Geffner, H. (2021). Learning general planning policies from small examples without supervision. In *Proc. AAAI*, pp. 11801–11808.

Geffner, H., & Bonet, B. (2013). *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers.

Geffner, T., & Geffner, H. (2015). Width-based planning for general video-game playing. In *Proc. AIIDE*, pp. 23–29.

Georgievski, I., & Aiello, M. (2015). HTN planning: Overview, comparison, and beyond. *Artificial Intelligence*, *222*, 124–156.

Ghallab, M., Nau, D., & Traverso, P. (2016). *Automated planning and acting*. Cambridge U.P.

Groshev, E., Goldstein, M., Tamar, A., Srivastava, S., & Abbeel, P. (2018). Learning generalized reactive policies using deep neural networks. In *Proc. ICAPS*, pp. 408–416.

Gupta, N., & Nau, D. (1992). On the complexity of blocks-world planning. *Artificial Intelligence*, *56*(2-3), 223–254.

Haslum, P., Lipovetzky, N., Magazzeni, D., & Muise, C. (2019). *An Introduction to the Planning Domain Definition Language*. Morgan & Claypool.

Helmert, M. (2002). Decidability and undecidability results for planning with numerical state variables.. In *Proc. AIPS*, pp. 44–53.

Hoffmann, J., Porteous, J., & Sebastia, L. (2004). Ordered landmarks in planning. *Journal of Artificial Intelligence Research (JAIR)*, *22*, 215–278.

Hofmann, T., & Geffner, H. (2024). Learning generalized policies for fully observable non-deterministic planning domains. *arXiv preprint arXiv:2404.02499*, *2024*.

Hogg, C., Munoz-Avila, H., & Kuter, U. (2008). HTN-Maker: Learning HTNs with minimal additional knowledge engineering required.. In *Proc. AAAI*, pp. 950–956.

Hu, Y., & De Giacomo, G. (2011). Generalized planning: Synthesizing plans that work for multiple environments. In *Proc. IJCAI*, pp. 918–923.

Icarte, R. T., Klassen, T., Valenzano, R., & McIlraith, S. (2018). Using reward machines for high-level task specification and decomposition in reinforcement learning. In *Proc. ICML*, pp. 2107–2116.

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 (JAIR)*, *73*, 173–208.

Illanes, L., & McIlraith, S. A. (2019). Generalized planning via abstraction: arbitrary numbers of objects. In *Proc. AAAI*, pp. 7610–7618.

Junyent, M., Gómez, V., & Jonsson, A. (2021). Hierarchical width-based planning and learning. In *Proc. ICAPS*, pp. 519–527.

Karia, R., Nayyar, R. K., & Srivastava, S. (2022). Learning generalized policy automata for relational stochastic shortest path problems. *Advances in Neural Information Processing Systems*, *35*, 30625–30637.

Khardon, R. (1999). Learning action strategies for planning domains. *Artificial Intelligence*, *113*(1–2), 125–148.

Knoblock, C. A. (1994). Automatically generating abstractions for planning. *Artificial intelligence*, *68*(2), 243–302.

Lipovetzky, N., & Geffner, H. (2012). Width and serialization of classical planning problems. In *Proc. ECAI*, pp. 540–545.

Lipovetzky, N. (2021). Planning for novelty: Width-based algorithms for common problems in control, planning and reinforcement learning. In *Proc. IJCAI*, pp. 4956–4960.

Lipovetzky, N., & Geffner, H. (2017a). Best-first width search: Exploration and exploitation in classical planning.. In *Proc. AAAI*, pp. 3590–3596.

Lipovetzky, N., & Geffner, H. (2017b). A polynomial planning algorithm that beats lama and ff. In *Proc. ICAPS*, pp. 195–199.

Liu, A., Xu, H., Van den Broeck, G., & Liang, Y. (2023). Out-of-distribution generalization by neural-symbolic joint training. In *Proc. AAAI*, pp. 12252–12259.

Machado, M. C., Bellemare, M. G., & Bowling, M. (2017). A Laplacian framework for option discovery in reinforcement learning. In *Proc. ICML*, pp. 2295–2304.

Martín, M., & Geffner, H. (2004). Learning generalized policies from planning examples using concept languages. *Applied Intelligence*, *20*(1), 9–19.

McGovern, A., & Barto, A. G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proc. ICML*, pp. 361–368.

Nau, D., Cao, Y., Lotem, A., & Munoz-Avila, H. (1999). SHOP: Simple hierarchical ordered planner. In *Proc. IJCAI*, pp. 968–973.

Ostrovski, G., Bellemare, M. G., Oord, A., & Munos, R. (2017). Count-based exploration with neural density models. In *Proc. ICML*, pp. 2721–2730.

Parr, R., & Russell, S. (1997). Reinforcement learning with hierarchies of machines. In *Proc. NeurIPS*, pp. 1043–1049.

Pathak, D., Agrawal, P., Efros, A. A., & Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp. 16–17.

Richter, S., & Westphal, M. (2010). The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research (JAIR)*, *39*(1), 127–177.

Rivlin, O., Hazan, T., & Karpas, E. (2020). Generalized planning with deep reinforcement learning. *arXiv preprint arXiv:2005.02305*, *2020*.

Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial intelligence*, *5*(2), 115–135.

Sanner, S., & Boutilier, C. (2009). Practical solution techniques for first-order MDPs. *Artificial Intelligence*, *173*(5-6), 748–788.

Segovia, J., Jiménez, S., & Jonsson, A. (2016). Generalized planning with procedural domain control knowledge. In *Proc. ICAPS*, pp. 285–293.

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

Srivastava, S., Immerman, N., & Zilberstein, S. (2008). Learning generalized plans using abstract counting. In *Proc. AAAI*, pp. 991–997.

Srivastava, S., Immerman, N., & Zilberstein, S. (2011a). A new representation and associated algorithms for generalized planning. *Artificial Intelligence*, *175*(2), 615–647.

Srivastava, S., Zilberstein, S., Immerman, N., & Geffner, H. (2011b). Qualitative numeric planning. In *Proc. AAAI*, pp. 1010–1016.

Ståhlberg, S., Bonet, B., & Geffner, H. (2023). Learning general policies with policy gradient methods. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, pp. 647–657.

Sutton, R. S., Precup, D., & Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, *112*(1-2), 181–211.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, X., Duan, Y., Schulman, J., Turck, F. D., & Abbeel, P. (2017). #Exploration: a study of count-based exploration for deep reinforcement learning. In *Proc. NeurIPS*, pp. 2753–2762.

Toro Icarte, R., Waldie, E., Klassen, T., Valenzano, R., Castro, M., & McIlraith, S. (2019). Learning reward machines for partially observable reinforcement learning. In *Proc. NeurIPS*, Vol. 32.

Toyer, S., Trevizan, F., Thiébaux, S., & Xie, L. (2018). Action schema networks: Generalised policies with deep learning. In *AAAI*, pp. 6294–6301.

van Otterlo, M. (2012). Solving relational and first-order logical markov decision processes: A survey. In Wiering, M., & van Otterlo, M. (Eds.), *Reinforcement Learning: State-of-the-Art*, pp. 253–292. Springer.

Wang, C., Joshi, S., & Khardon, R. (2008). First order decision diagrams for relational MDPs. *Journal of Artificial Intelligence Research (JAIR)*, *31*, 431–472.

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 *Proc. ICML*, pp. 11436–11446. PMLR.

Zhuo, H. H., Hu, D. H., Hogg, C., Yang, Q., & Munoz-Avila, H. (2009). Learning HTN method preconditions and action models from partial observations. In *Proc. IJCAI*, pp. 1804–1809.