General Policies, Representations, and Planning Width*

Blai Bonet,[1] **Hector Geffner**[2]
[1] Universitat Pompeu Fabra, Barcelona, Spain
[2] ICREA & Universitat Pompeu Fabra, Barcelona, Spain
bonetblai@gmail.com, hector.geffner@upf.edu

Abstract

It has been observed that in many of the benchmark planning domains, atomic goals can be reached with a simple polynomial exploration procedure, called IW, that runs in time exponential in the problem width. Such problems have indeed a bounded width: a width that does not grow with the number of problem variables and is often no greater than two. Yet, while the notion of width has become part of the state-of-the-art planning algorithms like BFWS, there is still no good explanation for why so many benchmark domains have bounded width. In this work, we address this question by relating bounded width and serialized width to ideas of generalized planning, where general policies aim to solve multiple instances of a planning problem all at once. We show that bounded width is a property of planning domains that admit optimal general policies in terms of features that are explicitly or implicitly represented in the domain encoding. The results are extended to the larger class of domains with bounded serialized width where the general policies do not have to be optimal. The study leads also to a new simple, meaningful, and expressive language for specifying domain serializations in the form of **policy sketches** which can be used for encoding domain control knowledge by hand or for learning it from traces. The use of sketches and the meaning of the theoretical results are all illustrated through a number of examples.

Introduction

Pure width-based search methods exploit the structure of states to enumerate the state space in ways that are different from standard methods like breadth-first, depth-first, or random search (Lipovetzky and Geffner 2012). For this, width-based methods appeal to a notion of novelty to establish a preference for first visiting states that are most novel. Novelty-based methods have also been used in the context of genetic algorithms where a greedy focus on the function to optimize (fitness) often leads to bad local optima (Lehman and Stanley 2011a,b), and in reinforcement learning to guide exploration in large spaces where reward is sparse (Tang et al. 2017; Pathak et al. 2017; Ostrovski et al. 2017).

In classical planning, i.e., planning in factored spaces for achieving a given goal from a known initial state (Geffner and Bonet 2013; Ghallab, Nau, and Traverso 2016), the notion of novelty is now part of state-of-the-art search algorithms like BFWS (Lipovetzky and Geffner 2017b,a) and has been applied successfully in purely exploratory settings where no compact model of the actions or goals is assumed to be known a priori (Francès et al. 2017; Bandres, Bonet, and Geffner 2018). A key open question in the area is why these width-based methods are effective at all, and in particular, why so many domains have a small width when atomic goals are considered (Lipovetzky and Geffner 2012). Is this a property of the domains? Is it an accident of the manual representations 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, Immerman, and Zilberstein 2008; Bonet, Palacios, and Geffner 2009; Hu and De Giacomo 2011; Belle and Levesque 2016; Segovia, Jiménez, and Jonsson 2016), in some cases by appealing to a fixed set Φ of general state features that we restrict to be linear. A number of correspondences are then obtained connecting the notions of *width, the size of general policies as measured by the number of features used, and the more general and useful notion of serialized width*. The study also yields a new meaningful and expressive language for specifying serializations, called **policy sketches**, which can be used for encoding domain control knowledge by hand or for learning it from traces.

The paper is organized as follows. We review first the notions of width, representations, and general policies, and relate width with the size of such policies. Then we introduce serializations, the more general notion of serialized width, the relation between general policies and serialized width, and policy sketches. A longer version of this paper with all the proofs is available (Bonet and Geffner 2020a).

Width

IW(1) is a simple search procedure that operates on a rooted directed graph where the nodes represent states, and states assign values v to a given set F of features f (Lipovetzky and Geffner 2012). IW(1) performs a breadth-first search starting at the root but pruning the states that do not make an atom $f{=}v$ true for the first time in the search. For classical planning problems expressed in languages such as (grounded) STRIPS, the features f are the problem variables

*A longer version of this paper with all the proofs is available (Bonet and Geffner 2020a).

which can take the values true or false. In other settings, like the Atari games as supported in ALE (Bellemare et al. 2013), the features and their values are defined in other ways (Lipovetzky, Ramirez, and Geffner 2015; Bandres, Bonet, and Geffner 2018). The procedure IW(k) for $k > 1$ is IW(1) but with a feature set given by F^k.

A finding reported by Lipovetzky and Geffner (2012) and exploited since in width-based algorithms is that the procedure IW(k) with $k = 2$ suffices to solve a wide variety of planning problems. Indeed, Lipovetzky and Geffner consider the 37,921 instances that result from all the domains used in planning competitions until 2012, where each instance with a goal made up of a conjunction of k atoms is split into k instances, each one with a single atomic goal. They report that IW(2) solves more than 88% of such instances, and moreover, 100% of the instances from 26 of the 37 domains considered.

Underlying the IW algorithms is the notion of *problem width*, which borrows from similar notions developed for constraint satisfaction problems and Bayesian networks (Freuder 1982; Pearl 1988; Dechter 2013). For planning, the notion introduced by Lipovetzky and Geffner takes this form:[1]

Definition 1 (Based on Lipovetzky and Geffner, 2012)**.** *The **width** $w(P)$ of problem P is the minimum k for which there is a **sequence** $t_0, t_1, \ldots, t_m$ **of atom tuples** t_i, each with at most k atoms, such that:*

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

The width is $w(P) = 0$ iff the initial state of P is a goal state. For convenience, we set $w(P)$ to 0 if the goal of P is reachable in a single step, and to $w(P) = N + 1$ if P has no solution where N is the number of atoms in P.

Chains of tuples $\theta = (t_0, t_1, \ldots, t_m)$ that comply with conditions 1–3 are called **admissible**, and the size of the chain is the size $|t_i|$ of the largest tuple in the chain. The width $w(P)$ is thus the minimum size of an admissible chain for P. The main properties of the IW(k) algorithm can then be expressed as follows:[2]

Theorem 2 (Lipovetzky and Geffner, 2012)**.** *IW(k) expands up to N^k nodes, generates up to bN^k nodes, and runs in time and space $O(bN^{2k-1})$ and $O(bN^k)$, respectively, where N is the number of atoms and b bounds the branching factor in problem P. IW(k) is guaranteed to solve P optimally (shortest path) if $w(P) \leq k$.*

When the width of problem P is not known, the **IW algorithm** can be run instead which calls IW(k) iteratively for $k = 0, 1, \ldots, N$ until the problem is solved, or shown to have no solution when IW(N) finds no plan. While these algorithms are not aimed at being practical, some state-of-the-art planners, make use of these ideas in a slightly different way (Lipovetzky and Geffner 2017b,a).

[1]For convenience, we set the width of problems that can be solved in one step to zero.

[2]It is assumed that the number of atoms affected by an action is bounded by a constant. When this is not the case, the time bound in Theorem 2 becomes $O(bN^{2k})$.

Representations

The width of a planning problem is tied to the representation language and the encoding of the problem in the language. In order to deal with a variety of possible languages and encodings, and since width-based methods rely on the *structure of states* but not on the *structure of actions* (i.e., action preconditions and effects), we consider *first-order languages* for describing states but not actions. In addition, the state language is extended with *features* f whose values $f(s)$ in a state s are determined by the state. The features provide additional expressive power and ways for bridging different state representation languages, although they are logically redundant as their value is determined by the truth value of the atoms in the state. The features extend the notion of *derived predicates* as defined in PDDL, as they do not have to be boolean, and they do not have to be defined in the language of first-order logic or logic programs (Thiébaux, Hoffmann, and Nebel 2005), but can be defined via procedures. Domains, instances, and states are defined as follows:

Definition 3 (Domains, problems, and states)**.** *A **domain** is a pair $D = (R, F)$ where R is a set of **primitive predicate symbols** with their corresponding arities, and F is a set of **features** defined in terms of the primitive predicates with their corresponding range of feature values. A **problem** P over domain $D = (R, F)$ is a tuple $P = (D, O, I, G)$ where O is a set of unique object names c (objects), and I and G are sets of ground atoms that denote the **initial** and **goal** states of P. A ground atom $r(c_1, \ldots, c_{a(r)})$ is made of a predicate $r \in R$ and an object tuple in $O^{a(r)}$ for the arity $a(r)$ of r. A **state** s over problem $P = (D, O, I, G)$ is a collection of ground atoms. The state s is a **goal** if $G \subseteq s$. A state s denotes a unique valuation for the ground atoms in P; $s \vDash r(c_1, \ldots, c_k)$ iff $r(c_1, \ldots, c_k)$ is in s.*

This is all standard except for the two details mentioned before: there are no action schemas, and there are state features. For the former, it is implicitly assumed that in each problem P, there is a function that maps states s into the set of possible transitions (s, s'). This implies, for example, that the states may contain *static atoms,* like adjacency relations, whose truth value are not affected by any action. For the features, we make the assumption that they are **linear**, in the sense that they can be computed efficiently and span a linear number of values only. More precisely:

Linear features assumption. The features f in F are either boolean or numerical, ranging in the latter case over the non-negative integers. The value of feature f in a state s for problem P, $f(s)$, can be computed in time bounded by $O(bN)$ where N is the number of atoms and b bounds the branching factor in P. Numerical features can take up to N values. ∎

This assumption rules out features like $V^*(s)$ that stands for the optimal cost (distance) from s to a goal which may

take a number of values that is not linear in the number of problem atoms, and whose computation may take exponential time. In many cases, the features can be defined in the language of first-order logic but this is not a requirement.

Example. Three state languages for Blocksworld are:

1. $\mathcal{L}^1_{BW}$ with the binary predicate (symbol) on^2 and the unary $ontable^1$ (superindex indicates arity),
2. $\mathcal{L}^2_{BW}$ with predicates on^2, $ontable^1$, $hold^1$, and $clear^1$,
3. $\mathcal{L}^3_{BW}$ with predicates on^2 and $hold^1$, and boolean features $ontable^1$ and $clear^1$. ∎

Example. Four languages for a domain Boxes, where boxes b containing marbles ma must be removed from a table, and for this, marble must be removed one by one first:

1. $\mathcal{L}^1_B$ with predicates $ontable^1(b)$ and $in^2(ma, b)$,
2. $\mathcal{L}^2_B$ with predicates $ontable^1$, in^2, and $empty^1(b)$,
3. $\mathcal{L}^3_B$ with predicates $ontable^1$ and in^2, and features $n(b)$ that count the number of marbles in b,
4. $\mathcal{L}^4_B$ with predicates $ontable^1$ and in^2, and features m and n counting the number of marbles in a box with the least number of marbles, and the number of boxes left. ∎

By abstracting away the details of the domain dynamics and the ability to introduce features, it is simple to move from one state representation to another.

The notion of width and the IW algorithms generalize to state languages containing features in a direct fashion. In both cases, the set of atoms considered is extended to contain the possible feature values $f{=}v$ where f is a feature and v is one of its possible values. Features are logically redundant but can have drastic effect on the problem width.

The width for class $\mathcal{Q}$ of problems P over some domain D is k, written as $w(\mathcal{Q}) = k$, if $w(P) = k$ for some $P \in \mathcal{Q}$ and $w(P') \leq k$ for every other problem P' in $\mathcal{Q}$.

Example. The width for the class of problems $\mathcal{Q}_{clear}$ where block x has to be cleared has width 1 in the state languages $\mathcal{L}^i_{BW}$, $i = 1, 2, 3$, while the class $\mathcal{Q}_{on}$ has width 2 for the three languages. On the other hand, for the class $\mathcal{Q}_{B_1}$ of instances from Boxes with a single box, the width is not bounded as it grows with the number of marbles when encoded in the languages $\mathcal{L}^1_B$ and $\mathcal{L}^2_B$, but it is 1 when encoded in the languages $\mathcal{L}^3_B$ or $\mathcal{L}^4_B$. Likewise, for the class $\mathcal{Q}_B$ of instances from Boxes with arbitrary number of boxes, the encoding in the language $\mathcal{L}^3_B$ has width that is not bounded as it grows with the number of boxes, but remains bounded and equal to 2 in $\mathcal{L}^4_B$. ∎

Generalized policies

We want to show that bounded width is a property of domains that admit a certain class of general policies. Different language for expressing general policies have been developed, some of which can deal with relational domains where different instances involve different (ground) actions. Most closely to this work, general policies have been defined in terms of qualitative numerical planning problems (QNPs) (Srivastava et al. 2011; Bonet and Geffner 2018, 2020b). We build on this idea but avoid the introduction of QNPs by defining policies directly as mappings from *boolean feature conditions* into *feature value changes*.

A **boolean feature condition** for a set of features Φ is a condition of the form p or $\neg p$ for a boolean feature p in Φ, or $n = 0$ or $n > 0$ for a numerical feature n in Φ. Similarly, a **feature value change** for Φ is an expression of the form p, $\neg p$, or $p?$ for a boolean feature p in Φ, and $n{\downarrow}$, $n{\uparrow}$, or $n?$ for a numerical feature n in Φ. General policies are given by a set of rules $C \mapsto E$ where C and E stands for boolean feature conditions and feature changes respectively.

Definition 4 (Policies)**.** *A **general policy** π_Φ for a domain D over a set of features Φ is a set of rules of the form $C \mapsto E$, where C is a set of boolean feature conditions and E is a set of feature value changes. The condition $n > 0$ is assumed in rules with effects $n{\downarrow}$ or $n?$.*

The policy π_Φ prescribes the possible actions a to be done in a state s over a problem P *indirectly,* as the set of state transitions (s, s') that the actions in P make possible and which are *compatible* with the policy:

Definition 5. *A transition (s, s') **satisfies** the effect E when:*

1. *if p (resp. $\neg p$) in E, $p(s') = 1$ (resp. $p(s') = 0$),*
2. *if $n{\downarrow}$ (resp. $n{\uparrow}$) in E, $n(s) > n(s')$ (resp. $n(s) < n(s')$),*
3. *if p (resp. n) is not mentioned at all in E, $p(s) = p(s')$ (resp. $n(s) = n(s')$).*

*The transition (s, s') is **compatible** with policy π_Φ (or is a π_Φ-transition) if there is a policy rule $C \mapsto E$ such that s makes true C and (s, s') satisfies E.*

Policy rules provide a description of how the value of the features must change along the state trajectories that are compatible with the policy.

Example. A policy for solving the class $\mathcal{Q}_{clear}$ can be expressed in terms of the features $\Phi = \{H, n\}$, where H is true if a block is being held, and n counts the number of blocks above x. The policy can be expressed with two rules:

$$\{\neg H, n{>}0\} \mapsto \{H, n{\downarrow}\} \;\; ; \;\; \{H, n{>}0\} \mapsto \{\neg H\}\,. \quad (1)$$

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 and there are blocks above x, an action that makes H false and does not affect n must be selected. ∎

The conditions under which a general policy π_Φ solves an instance P and class $\mathcal{Q}$ are:

Definition 6 (Trajectories and solutions)**.** *A **state trajectory** $s_0, \ldots, s_n$ for problem P is **compatible** with policy π_Φ over features Φ (or is π_Φ-trajectory), iff s_0 is the initial state of P, no state s_i is goal, $0 \leq i < n$, and each pair (s_i, s_{i+1}) is a possible state transition in P that is compatible with π_Φ. It is **maximal** if either s_n is a goal state, no transition (s_n, s_{n+1}) in P is compatible with π_Φ, or the trajectory is infinite (i.e., $n = \infty$). A policy π_Φ **solves** a problem P if all maximal state trajectories $s_0, \ldots, s_n$ compatible with π_Φ are goal reaching (i.e., s_n is a goal state in P). π_Φ solves a collection $\mathcal{Q}$ of problems if it solves each problem in $\mathcal{Q}$.*

It is easy to show that the policy captured by the rules in (1) solves $\mathcal{Q}_{clear}$. The verification and synthesis of general policies of this type have been addressed by Bonet and

Geffner (2020b) in the context of qualitative numerical planning, and by Bonet, Frances, and Geffner (2019) where the set of features Φ and QNP model are learned from traces.

Generalized Policies and Width

The first results establish a relation between the width of classes of problems $\mathcal{Q}$ and the size $|\Phi|$ of certain optimal policies that solve $\mathcal{Q}$. A policy π_Φ over the features Φ is is said to be Markovian when the features provide a suitable abstraction of the states; i.e., when the possible next feature valuations are not just a function of the current state, but of the feature valuation in the state. More precisely, if we say that a state s is *optimal for a feature valuation* f in a problem P when $f = f(s)$ and there is no state s' with the same feature valuation $f = f(s')$ that is reachable in P in less number of steps than s, the Markovian property is defined as follows:

Definition 7 (Markovian)**.** *A policy π_Φ is **Markovian** for a problem P iff the existence of a transition (s, s') compatible with π_Φ with feature valuations $f = f(s)$ and $f' = f(s')$, implies the existence of transitions (s_1, s'_1) with the same feature valuations $f = f(s_1)$ and $f' = f(s'_1)$, in all states s_1 that are optimal for f in P. The policy is **Markovian** for a class of problems $\mathcal{Q}$ if it is so for each P in $\mathcal{Q}$.*

The states s_1 on which the Markovian condition is required are not necessarily among those which are reachable with the policy.

The features must also distinguish goals from non-goals:

Definition 8 (Separation)**.** *The features Φ **separate goals from non-goals** in $\mathcal{Q}$ iff there is a set of boolean feature valuations κ such that for any problem P in $\mathcal{Q}$ and any* reachable *state s in P, s is a goal state iff $f(s)$ is in κ. The valuations in κ are called **goal valuations**.*

The **boolean feature valuations** determined by state s refer to the truth valuations of the expressions p and $n = 0$ for the boolean and numerical features p and n in Φ, respectively. While the number of feature valuations is not bounded as the size of the instances in $\mathcal{Q}$ is not bounded in general, the number of boolean feature valuations is always $2^{|\Phi|}$.

The last notion required is the notion of optimality:

Definition 9 (Optimal policies)**.** *A policy π_Φ that solves a class of problems $\mathcal{Q}$ is **optimal** if any plan ρ induced by π_Φ over a problem P in $\mathcal{Q}$ is optimal for P.*

If the policy π_Φ solves a problem P, the plans induced by the policy are the action sequences ρ that yield the goal-reaching trajectories that are compatible with π_Φ. These plans are optimal for P if there are no shorter plans for P.

Under these conditions, the width of the instances in $\mathcal{Q}$ can be bounded by the number of features $|\Phi|$ in the policy π_Φ, provided that the features are represented explicitly in the instances:[3]

Theorem 10. *Let π_Φ be a **Markovian** policy that solves a class of problems $\mathcal{Q}$ **optimally,** where the features Φ **separate goals**. If the features in Φ are **explicitly represented** in the instances P in $\mathcal{Q}$, $w(P) \leq |\Phi|$.*

[3]Proofs in (Bonet and Geffner 2020a).

Indeed, if a policy π_Φ solves $\mathcal{Q}$ optimally under the given conditions, an *admissible chain* $t_0, t_1, \ldots, t_n$ of size $k = |\Phi|$ can be formed for solving each problem P in $\mathcal{Q}$ optimally, where t_i is the valuation of the features in Φ at the i-th state of any state trajectory that results from applying the policy.

If we let IW_Φ refer to the variant of IW that replaces tuples of atoms by feature valuations and thus deems a state s novel in the search (unpruned) when $f(s)$ has not been seen before, one can show that:

Theorem 11. *Let π_Φ be a **Markovian** policy that **solves** a class of problems $\mathcal{Q}$ **optimally** with features Φ that **separate the goals**. The procedure IW_Φ solves any instance P in $\mathcal{Q}$ **optimally** in time $O(N^{|\Phi|})$ where N is the number of atoms in P.*

Example. A general policy for Boxes is given by the rules: $\{m > 0\} \mapsto \{m\downarrow\}$ and $\{m = 0, n > 0\} \mapsto \{n\downarrow, m?\}$ where m and n are two features that count the number of marbles in a box with a least number of marbles, and the number of boxes left on the table. Since the policy complies with the conditions in Theorem 10 and the two features are represented explicitly in the language $\mathcal{L}^4_B$, it follows that the width of instances of Boxes in such encoding is 2. ∎

Example. Similarly, the policy π_Φ with features $\Phi = \{H, n\}$ for $\mathcal{Q}_{clear}$ given by the two rules in (1) is Markovian, solves $\mathcal{Q}_{clear}$ optimally, and the features separate goals. Yet there are no atoms representing the counter n explicitly. Still, Theorem 11 implies that IW_Φ solves the instances in $\mathcal{Q}_{clear}$ optimally in quadratic time. ∎

Theorem 10 relates the number of features in a general policy π_Φ that solves $\mathcal{Q}$ with the width of the class $\mathcal{Q}$ provided that the features are part of the problem encodings. This, however, is not strictly necessary:

Theorem 12. *Let π_Φ be an **optimal** and **Markovian** policy that solves a class $\mathcal{Q}$ of problems over some domain D for which Φ **separates the goals**. The width of the problems P is bounded by k if for any sequence of feature valuations $\{f_i\}_i$ generated by the policy π_Φ in the way to the goal, there is a sequence of sets of atoms $\{t_i\}_i$ in P of size at most k such that the optimal plans for t_i and the optimal plans for f_i coincide.*

Example. Consider an instance P in $\mathcal{Q}_{clear}$ where $B_1, \ldots, B_m$ are the blocks above x initially, from top to bottom, $m > 0$. The feature valuations in the way to the goal following the Markovian policy π_Φ are $f_i = \{H, n = m - i\}$, $i = 1, \ldots, m$, and $g_i = \{\neg H, n = m - i\}$, $i = 0, \ldots, m - 1$. The policy is optimal, Markovian, and the features separate the goals. The tuples of atoms in P that capture these valuations as expressed in Theorem 12 are $t_{f_i} = \{hold(B_i)\}$ and $t_{g_i} = \{ontable(B_i)\}$ for $i > 0$, and $t_{g_0} = \{clear(B_1)\}$. Since these tuples have size 1, the widths $w(P)$ and $w(\mathcal{Q}_{clear})$ are both equal to 1. ∎

Admissible Chains and Projected Policies

Theorem 12 relates the width of $\mathcal{Q}$ to the size of the atom tuples t_i that capture the values of the features f_i in all optimal trajectories compatible with an optimal policy for $\mathcal{Q}$. It

is often sufficient, however, if the tuples t_i capture the value of the features f_i in *some* of those trajectories only. Let us start with the notion of *feasible chains:*

Definition 13 (Feasible chain)**.** *Let $\theta = (t_0, t_1, \ldots, t_n)$ be a chain of tuples of atoms from P. The chain θ is **feasible** in problem P if t_0 is true in the initial state, the optimal plans for t_n have length n, and they are all optimal for P.*

An **admissible** chain, as used in the definition of width, is a feasible chain that satisfies an extra condition; namely, that every optimal plan ρ for the tuple t_i in the chain can be extended with a single action into an optimal plan for the tuple t_{i+1}, $i = 0, 1, \ldots, n-1$. In order to account for this key property, we map feasible chains into features and policies as follows:

Definition 14. *Let $\theta = (t_0, t_1, \ldots, t_n)$ be a chain of atom tuples from P, and let $\tilde{t}_i(s)$ denote the **boolean state feature** that is true in s when t_i is true in s and t_j is false for all $i < j \leq n$, $i = 1, \ldots, n$. The chain defines a **policy** π_θ over P with rules $\{\tilde{t}_i\} \mapsto \{\tilde{t}_{i+1}, \neg\tilde{t}_i\}$, $i = 0, \ldots, n-1$.*

The first result gives necessary and sufficient conditions for a chain θ to be admissible.

Theorem 15. *Let $\theta = (t_0, t_1, \ldots, t_n)$ be a chain of tuples for a problem P. θ is **admissible** in P if and only if θ is **feasible** and the policy π_θ solves P **optimally** and is **Markovian**.*

This result connects admissible chains with policies that are optimal and Markovian, but does not connect admissible chains with general policies. This is done next. We first define when a policy π_1 can be regarded as a **projection** of another policy π_2:

Definition 16 (Projection)**.** *Let π_Φ be a policy over a class $\mathcal{Q}$ and let $\pi_{\Phi'}$ be a policy for a problem P in $\mathcal{Q}$. The policy $\pi_{\Phi'}$ is a **projection** of π_Φ in P if every **maximal state trajectory** compatible with $\pi_{\Phi'}$ in P is a **maximal state trajectory** in P compatible with π_Φ.*

Notice that it is not enough for the state trajectories $s_0, \ldots, s_i$ compatible with $\pi_{\Phi'}$ to be state trajectories compatible with π_Φ; it is also required that if s_i is a final state in the first trajectory that it is also a final state in the second one. This rules out the possibility that there is a continuation of the first trajectory that is compatible with π_Φ but not with $\pi_{\Phi'}$. A result of this is that if π_Φ is optimal for $\mathcal{Q}$, the projected policy $\pi_{\Phi'}$ must be optimal for P. From this, the main theorem of this section follows:

Theorem 17. *Let π_Φ be an **optimal** policy for a class $\mathcal{Q}$ of problems. If for any problem P in $\mathcal{Q}$, there is a **feasible** chain θ of size at most k such that π_θ is a **projection** of π_Φ in P that is **Markovian**, then $w(\mathcal{Q}) \leq k$.*

While the proof of this theorem is a direct consequence of Theorem 15, the result is important as *it renders explicit the logic underlying all proofs of bounded width for meaningful classes of problems $\mathcal{Q}$ that we are aware of.* In all such cases, the proofs have been constructed by finding feasible chains with tuples t_i that are a function of the instance $P \in \mathcal{Q}$, and whose role is to capture suitable projections of **some general optimal policy** π_Φ.

Example. In the Delivery (D) domain an agent moves in a grid to pick up packages and deliver them to a target cell, one by one; Delivery-1 (D_1) is the version with one package. A general policy π_Φ for D and D_1 is given in terms of four rules that capture: move to the (nearest) package, pick it up, move to the target cell, and drop the package, in a cycle, until no more packages are left. The rules can be expressed in terms of the features $\Phi = \{H, p, t, n\}$ that express holding, distance to the nearest package (zero if agent is holding a package or no package to be delivered remains), distance to the target cell, and the number of undelivered packages respectively. Hence, $n = 0$ identifies the goal states for the problems in the classes $\mathcal{Q}_D$ and $\mathcal{Q}_{D_1}$ for the problems D and D_1 respectively. The rules that define the policy π_Φ are $\{\neg H, p > 0\} \mapsto \{p\downarrow, t?\}$, $\{\neg H, p = 0\} \mapsto \{H\}$, $\{H, t > 0\} \mapsto \{t\downarrow\}$, and $\{H, n > 0, t = 0\} \mapsto \{\neg H, n\downarrow, p?\}$.[4]

Let us consider a problem P in the class $\mathcal{Q}_{D_1}$. If the encoding of P contains atoms like $at(cell_i)$, $atp(pkg_i, cell_j)$, $hold(pkg_i)$, and $empty$, it can be shown that $\mathcal{Q}_{D_1}$ has width 2. Indeed, without loss of generality, let us assume that in P, the package is initially at $cell_i$ and has to be delivered at $cell_t$, and the agent is initially at $cell_0$, and let $\theta = (t_0, t_1, \ldots, t_n)$ be the chain made of tuples of the form $\{at(cell_k)\}$ for the cells on a shortest path from $cell_0$ to $cell_i$, followed by tuples of the form $\{at(cell_k), hold(pkg_j)\}$, with $cell_k$ now ranging over a shortest path from $cell_i$ up to $cell_t$, and a last tuple t_n of the form $\{at(pkg_j, cell_t)\}$. It is easy to show that the chain θ is feasible, and π_θ is the projection of the general policy π_Φ that in D_1 is both optimal and Markovian (although not in D). Then, by Theorems 15 and 17, it follows that the chain θ is admissible, and $w(\mathcal{Q}_{D_1}) \leq 2$. ∎

Serialized Width

Theorem 17 suggests that bounded width is often the result of general policies that project on suitable tuples of atoms. We extend this relation now to the larger class of problems that have bounded *serialized width* where problems are decomposed in subproblems and the general policies do not have to be optimal or Markovian. A *serialization* is a *strict partial order* (an irreflexive and transitive binary relation) over the feature valuations (Φ-tuples) over a whole class of problem $\mathcal{Q}$:

Definition 18 (Serializations)**.** *Let $\mathcal{Q}$ be a class of problems, let Φ be a set of goal-separating features for $\mathcal{Q}$, and let $\prec$ be a **strict partial order** over Φ-tuples. The pair $(\Phi, \prec)$ is a **serialization** over $\mathcal{Q}$ if 1) the ordering $\prec$ is well-founded; i.e. there is no infinite descending chain $f_1 \succ f_2 \succ f_3 \succ \cdots$ where $f_i \succ f_{i+1}$ stands for $f_{i+1} \prec f_i$, and 2) the goal*

[4] A different formulation involves packages that need to be delivered to target cells that depend on the package. The set of features is the same $\Phi = \{H, p, t, n\}$ except that t is defined as the distance to the *current target cell* (zero if agent holds nothing or there are no more packages to deliver). A policy for this formulation has the rules $\{\neg H, p > 0\} \mapsto \{p\downarrow\}$, $\{\neg H, p = 0\} \mapsto \{H, t\uparrow\}$, $\{H, t > 0\} \mapsto \{t\downarrow\}$, and $\{H, n > 0, t = 0\} \mapsto \{\neg H, n\downarrow, p?\}$.

feature valuations f are $\prec$-minimal; i.e., no $f' \prec f$ for any feature valuation f'.

A serialization decomposes each problem P in $\mathcal{Q}$ into subproblems that define the width of the serialization:

Definition 19 (Subproblems)**.** *Let $(\Phi, \prec)$ be a serialization. The subproblem $P[s, \prec]$ is the problem of finding a state s' reachable from s such that s' is goal in P or $f(s') \prec f(s)$; i.e., the goal states in $P[s, \prec]$ are the goal states in P and the states s' such that $f(s') \prec f(s)$. The collection $P[\prec]$ of subproblems for problem P is the smallest subset of problems $P[s, \prec]$ that comply with:*

1. *if the initial state s_0 in P is not goal, $P[s_0, \prec] \in P[\prec]$,*
2. *$P[s', \prec] \in P[\prec]$ if $P[s, \prec] \in P[\prec]$ and s' is a non-goal state in P that is at **shortest distance** from s such that $f(s') \prec f(s)$, and no goal state of P is strictly closer from s than s'*

Definition 20 (Serialized width)**.** *Let $(\Phi, \prec)$ be a serialization for a collection $\mathcal{Q}$ of problems. The **serialized width** of problem P relative to $(\Phi, \prec)$ is k, written as $w_\Phi(P) = k$ with the ordering "$\prec$" left implicit, if there is a subproblem $P[s, \prec]$ in $P[\prec]$ that has width k, and every other subproblem in $P[\prec]$ has width at most k. The **serialized width** for $\mathcal{Q}$ is k, written as $w_\Phi(\mathcal{Q}) = k$, if $w_\Phi(P) = k$ for some problem $P \in \mathcal{Q}$, and $w_\Phi(P) \leq k$ every other problem $P' \in \mathcal{Q}$.*

If a class of problems $\mathcal{Q}$ has bounded serialized width and the ordering $f \prec f'$ can be tested in polynomial time, each problem P in $\mathcal{Q}$ can be solved in polynomial time using a variant SIW$_\Phi$ of the SIW algorithm: starting at the state $s = s_0$, SIW$_\Phi$ performs an IW search from s to find a state s' that is a goal state or that renders the precedence constraint $f(s') \prec f(s)$ true. If s' is not a goal state, s is set to s', $s := s'$, and the loop repeats until a goal state is reached. The result below follows from the observations by Lipovetzky and Geffner (2012) once the goal counter is replaced by a partial order, and the notion of serialized width is suitably formalized:

Theorem 21. *Let $\mathcal{Q}$ be a collection of problems and let $(\Phi, \prec)$ be a serialization for $\mathcal{Q}$ with width $w_\Phi(\mathcal{Q}) \leq k$. Any problem P in $\mathcal{Q}$ is solved by SIW$_\Phi$ in time and space bounded by $O(bN^{|\Phi|+2k-1}\Lambda)$ and $O(bN^k + N^{|\Phi|+k})$ respectively, where b bounds the branching factor in P, N is the number of atoms in P, and Λ bounds the time to test the order $\prec$ for the Φ-tuples that arise from the states in P.*

The class $\mathcal{Q}_{VA}$ of instances for the problem VisitAll, where all cells in a grid must be visited, has serialized width $w_\Phi(\mathcal{Q}_{VA}) = 1$ for $\Phi = \{\#g\}$ where $\#g$ counts the number of unvisited cells (unachieved goals), with the obvious ordering ('$\prec$' set to '$<$'). The class $\mathcal{Q}_{BW}$ of Blocksworld instances cannot be serialized into subproblems of bounded width with $\Phi' = \{\#g\}$ because achieved goals may have to be undone, yet with $\Phi = \{\#m\}$ where $\#m$ counts the number of misplaced blocks (i.e., not on their targets or above one such block), $w_\Phi(\mathcal{Q}_{BW}) = 2$. The result above implies that SIW$_\Phi$ solves these problems in polynomial time.

From General Policies to Serializations

We show next how serializations can be inferred from general policies, a result that paves the way to introduce a convenient language for defining serializations. For this, however, we focus on the policies π_Φ that solve a class of problems $\mathcal{Q}$ **structurally**; i.e., by virtue of the effects of the actions on the features as expressed in the policy rules in π_Φ. We use ideas developed in the context of QNPs (Srivastava et al. 2011; Bonet and Geffner 2020b).

The key idea is the notion of **terminating policy graph.** Recall that every feature valuation f defines a projected boolean valuation $b(f)$ over the expressions p and $n = 0$ for the boolean and numerical features p and n in Φ, where p and $n = 0$ are true in $b(f)$ iff p and $n = 0$ are true in f respectively. The policy graph for policy π_Φ is:

Definition 22 (Policy graph)**.** *The policy graph $G(\pi_\Phi)$ for policy π_Φ has nodes b, one for each of the $2^{|\Phi|}$ boolean feature valuations over Φ, and edges $b \to b'$ labeled with E if b is not a goal valuation and (b, b') is compatible with a rule $C \to E$ in the policy.*

The edge $b \to b'$ is compatible with a rule $C \to E$ if C is true in b, and (b, b') is compatible with E; namely, if p (resp. $\neg p$) is in E, p (resp. $\neg p$) must be true in b', and if $n\!\uparrow$ is in E, $n = 0$ must be false in E. Effects $p?$, $n?$, and $n\!\downarrow$ in E put no constraints on b'. In particular, in the latter case, $n = 0$ can be either true or false in b', meaning that after a decrement, a numerical feature may remain positive or have value 0. Notice that the policy graph associated with a policy π_Φ does not take the class of problems $\mathcal{Q}$ into account. Indeed, the policy graph may have an edge $b \to b'$ even if there is no state transition (s, s') in an instance in $\mathcal{Q}$ where this transition between these boolean valuations arises. The policy graph and the notion of **termination** below are purely **structural** as they depend on the form of the policy rules only:

Definition 23 (Termination)**.** *A policy π_Φ and a policy graph $G(\pi_\Phi)$ are **terminating** if for every cycle in the graph $G(\pi_\Phi)$, i.e., any sequence of edges $b_i \to b_{i+1}$ with labels E_i that start and end in the same node, there is a numerical feature n in Φ that is decreased along some edge and increased in none. That is, $n\!\downarrow \in E_k$ for some k-th edge in the cycle, and $n\!\uparrow \notin E_j$ and $n? \notin E_j$ for all others edges in the cycle.*

The termination condition can be checked in time that is polynomial in the size of the policy graph by a procedure called SIEVE that looks at the strongly connected components in the graph (Srivastava et al. 2011; Bonet and Geffner 2020b).[5] A key property of terminating policies is that they give rise to state trajectories that are finite.

Theorem 24. *Let π_Φ be a terminating policy with features Φ over $\mathcal{Q}$ that separate the goals. Then, π_Φ cannot give rise to infinite state trajectories in instances P in $\mathcal{Q}$.*

[5] See Bonet and Geffner (2020b) for more details on the formal properties of termination in QNPs. Our setting is slightly different as our numerical features are linear and cannot grow without bound as in QNPs, but we do not use this property to define termination.

Since terminating policies induce state trajectories with a final state, a sufficient condition for a terminating policy to solve a class $\mathcal{Q}$ is to be closed:

Definition 25 (Closed policy)**.** *A policy π_Φ is **closed** over a class of problems $\mathcal{Q}$ if for any non-goal state s in a problem P in $\mathcal{Q}$ that is π_Φ-reachable, there is a transition (s, s') that is compatible with π_Φ.*

Theorem 26. *If π_Φ is closed over $\mathcal{Q}$, the features in Φ separate goals in $\mathcal{Q}$, and π_Φ is terminating, π_Φ **solves** $\mathcal{Q}$.*

We are interested in the conditions under which general policies determine serializations, and in particular, serializations with bounded width. For the first part, we do not need the policy to be closed or solve $\mathcal{Q}$, but just to be terminating:

Theorem 27. *A terminating policy π_Φ with features Φ over $\mathcal{Q}$ that separate goals determines a serialization $(\Phi, \prec)$ of $\mathcal{Q}$ where '$\prec$' is the minimal strict partial relation (i.e., the transitive closure) that satisfies $f' \prec f$ for a non-goal valuation f, if (f, f') is compatible with a policy rule in π_Φ.*

Here a pair of feature valuations (f, f') is compatible with a rule $C \rightarrow E$, if C is true in f and the change in values from f to f' is compatible with E. Notice that if a state transition (s, s') is compatible with $C \rightarrow E$, the pair of feature valuations $(f(s), f(s'))$ is compatible with the rule, but the existence of such a state transition is not a requirement for the pair (f, f') to be compatible with the policy rule; they can be arbitrary feature valuations over Φ.

There are simple syntactic criteria that ensure that a policy is terminating. For example, a policy that uses numerical features only is terminating if there is an ordering on the features such that every rule $C \mapsto E$ that increases a feature n_i decreases a feature n_j later in the ordering. Such policies are called **regular** (Bonet and Geffner 2020b).

Example. The policy π_Φ for Boxes with rules $\{m > 0\} \mapsto \{m\downarrow\}$ and $\{m = 0, n > 0\} \mapsto \{n\downarrow, m?\}$ and goal $n = 0$ is **closed** and **regular**. It is closed since for any state s where $n > 0$, there is a transition (s, s') that is compatible with π_Φ: if $m = 0$, s' results from an action that puts an empty box away, while if $m > 0$, s' results from an action that puts a marble away from a box with a least number of marbles. Theorem 26 implies that π_Φ solves $\mathcal{Q}_B$. ∎

If a terminating policy for the class $\mathcal{Q}$ defines a serialization over $\mathcal{Q}$, a terminating policy that solves $\mathcal{Q}$ should define a serialization over $\mathcal{Q}$ with zero width. Two conditions are needed for this though. The first is the notion of **goal connectedness:**

Definition 28 (Goal connected)**.** *A policy π_Φ for $\mathcal{Q}$ with goal separating features and its policy graph are said to be **goal connected** when all nodes $b(f(s_0))$ associated with the initial states s_0 of instances P in $\mathcal{Q}$ are connected only to nodes b that are connected with goal nodes.*

Clearly, a policy π_Φ for $\mathcal{Q}$ is not closed if its policy graph is not goal-connected, but goal-connectedness does not imply that the policy is closed. For this, we need a condition that goes beyond the structure of the policy graph:[6]

Definition 29 (Sound policy)**.** *A policy π_Φ over $\mathcal{Q}$ is **sound** if for any reachable non-goal state s in an instance P in $\mathcal{Q}$ where the policy rule $C \mapsto E$ is applicable (i.e., where C holds), there is a transition (s, s') in P that is compatible with π_Φ.*

Soundness and goal connectedness imply that a policy is closed, and both are critical for establishing the conditions under which the serialization induced by a terminating policy has zero width:

Theorem 30. *If π_Φ is sound and goal-connected in $\mathcal{Q}$, then π_Φ is closed in $\mathcal{Q}$.*

From Theorem 26, it follows that a terminating policy π_Φ that is sound and goal-connected in $\mathcal{Q}$, solves $\mathcal{Q}$. In such a case, we say that the policy π_Φ solves the class of problems $\mathcal{Q}$ **structurally**, as two of the conditions, termination and goal-connectedness, can be tested on the policy graph. Soundness, on the other hand, is the condition that ensures that only sink nodes in the policy graph over any instance $P \in \mathcal{Q}$ are the goal nodes.

As expected, if a terminating policy π_Φ solves $\mathcal{Q}$, the width of $\mathcal{Q}$ under the serialization determined by the policy (Theorem 27) is zero, provided however, that certain structural conditions hold:

Theorem 31. *Let π_Φ be a policy that solves $\mathcal{Q}$ **structurally** and let $(\Phi, \prec)$ be the serialization over $\mathcal{Q}$ determined by π_Φ. If π_Φ is sound and goal connected, $w_\Phi(\mathcal{Q}) = 0$.*

Example. The policy for Boxes in the previous example is closed, goal connected, and sound. By Theorem 31, it determines a serialization $(\Phi, \prec)$ of width zero, where $f(s) = [n(s), m(s)] \prec f(s') = [n(s'), m(s')]$ iff $n(s) < n(s')$ or $n(s) = n(s')$ and $m(s) < m(s')$. ∎

Example. In Delivery, we use the features $\Phi = \{H, p, t, n\}$ and the policy π_Φ defined by the rules $\{\neg H, p > 0\} \mapsto \{p\downarrow, t?\}$, $\{\neg H, p = 0\} \mapsto \{H\}$, $\{H, t > 0\} \mapsto \{t\downarrow\}$, and $\{H, n > 0, t = 0\} \mapsto \{\neg H, n\downarrow, p?\}$. It is easy to check that π_Φ is sound for the classes $\mathcal{Q}_D$ and $\mathcal{Q}_{D_1}$ since for any reachable non-goal state s in a problem P, there is a transition (s, s') that is compatible with π_Φ. On the other hand, the policy graph for π_Φ, depicted in Fig. 1, is clearly terminating and goal connected. Since Φ separates goals for the classes $\mathcal{Q}_D$ and $\mathcal{Q}_{D_1}$, by Theorem 26, π_Φ solves both classes structurally, and the induced serialization $(\Phi, \prec)$ has width zero for the classes $\mathcal{Q}_D$ and $\mathcal{Q}_{D_1}$ by Theorem 31. ∎

Sketches: A Language for Serializations

We make use of the results above for introducing a convenient **language** for specifying serializations. The language can be used either to encode serializations by hand for extending the scope of width-based algorithms such as SIW$_\Phi$, or for **learning serializations** from traces.

A **policy sketch** or simply **sketch** R_Φ, for a class of problems $\mathcal{Q}$, is a set of policy rules over the features Φ that distinguish the goals of $\mathcal{Q}$. The sketch R_Φ can be a full fledged policy over $\mathcal{Q}$, part of it, or just set of policy rules $C \rightarrow E$,

[6]The notion of soundness is similar to action soundness in QNPs when QNPs are used to abstract classes of problems (Bonet and Geffner 2018; Bonet, Frances, and Geffner 2019).

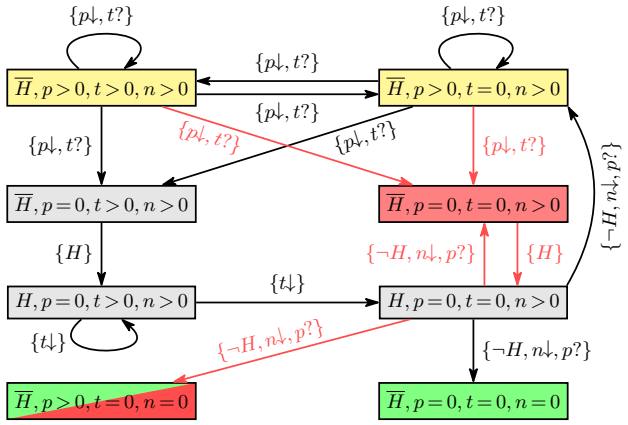

Figure 1: Policy graph for Delivery for the policy defined by the rules $\{\neg H, p>0\} \mapsto \{p\downarrow, t?\}$, $\{\neg H, p=0\} \mapsto \{H\}$, $\{H, t>0\} \mapsto \{t\downarrow\}$, and $\{H, n>0, t=0\} \mapsto \{\neg H, n\downarrow, p?\}$. Yellow and green nodes denote initial and goal nodes respectively. Red nodes and edges stand for nodes and transitions in the policy graph that do not arise in the instances. The graph is terminating and goal connected, and the policy is closed and sound for the classes $\mathcal{Q}_D$ and $\mathcal{Q}_{D_1}$.

including the empty set. By interpreting R_Φ as a policy, we can transfer previous results that involve policy graphs and termination. We call the rules in a sketch R_Φ, **sketch rules** because their **semantics** is different from the semantics of policy rules.

Definition 32 (Sketch)**.** *A sketch for $\mathcal{Q}$ is a set R_Φ of sketch rules $C \to E$ over features Φ that **separate goals** in $\mathcal{Q}$. The sketch R_Φ is **well-formed** if the set of rules R_Φ interpreted as a policy is **terminating**.*

Notice that the definition of terminating policies does not require the policy to be closed or even to solve $\mathcal{Q}$. Theorem 27 directly yields:

Theorem 33. *A well-formed sketch R_Φ for $\mathcal{Q}$ defines a serialization $(\Phi, \prec)$ over $\mathcal{Q}$ where '$\prec$' if the smallest strict partial order that satisfies $f' \prec f$ if the pair of feature valuations (f, f') is compatible with a sketch rule in R_Φ.*

The distinction between policy and sketch rules is **semantical**, not syntactical. A policy π_Φ defines a filter on state transitions and a serialization (Theorem 27). A sketch R_Φ, on the other hand, defines just a serialization. Namely, each sketch rule $C \mapsto E$ defines the **subproblem** of going from a state s where C holds in $f(s)$ to a state s' such that the pair of feature valuations $(f(s), f(s'))$ is compatible with the rule. It is not a requirement that s' should be reachable in one step from s. Sketches thus provide a language for decomposing a problem into subproblems and thus for reducing its width, which goes well beyond the language of goal counters and variations, as the language for sketches includes the language of general policies.

Example. The serialization given by the single feature $\#g$ that counts the number of unachieved (top) goals is captured with the sketch that only contains the rule $\{\#g>0\} \mapsto \{\#g\downarrow\}$ when there are no other features, and the rule $\{\#g>0\} \mapsto \{\#g\downarrow, p?, n?\}$ when p and n are other features. The rules say that it is "good" to decrease the goal counter independently of the effects on other features. ∎

Our last results are about the width of the serializations defined by sketches, and the modifications in the SIW_Φ algorithm to work with sketches:

Definition 34 (Sketch width)**.** *Let R_Φ be a well-formed sketch for a class of problems $\mathcal{Q}$ such that Φ separates the goals, and let s be a reachable state in some instance P of $\mathcal{Q}$. The width of the sketch R_Φ **at state** s **of problem** P, $w_R(P[s])$, is the width of the subproblem $P[s]$ that is like P but with initial state s and goal states s' such that s' is a goal state of P, or the pair $(f(s), f(s'))$ is compatible with a sketch rule $C \mapsto E$. The **width of the sketch** R_Φ, $w_R(\mathcal{Q})$, is the maximum width $w_R(P[s])$ for any reachable state s in any problem P in $\mathcal{Q}$.*

Theorem 35. *Let R_Φ be a well-formed sketch for a class $\mathcal{Q}$ of problems, and let $(\Phi, \prec)$ be the serialization determined by R_Φ from Theorem 33. The width $w_\Phi(\mathcal{Q})$ of the serialization is bounded by the width $w_R(\mathcal{Q})$ of the sketch.*

If a well-formed sketch R_Φ has bounded width for a class of problems $\mathcal{Q}$, then the problems in $\mathcal{Q}$ can be solved in polynomial time by an algorithm SIW_R that is like SIW_Φ, with the difference that the precedence test $f \prec f'$ among pairs of feature valuations f and f' is replaced by the test of whether the feature valuation pair (f, f') is compatible with a rule in R_Φ. In other words, SIW_R start at the state $s := s_0$, where s_0 is the initial state of P, and then performs an IW search from s to find a state s' that is a goal state or such the pair $(f(s), f(s'))$ is compatible with a sketch rule in R_Φ. Then if s' is not a goal state, s is set to s', $s := s'$, and the loop repeats until a goal state is reached. The precedence test in R_Φ can be done in constant time unlike the general test $f' \prec f$ in SIW_Φ.[7] The runtime properties of SIW_R are thus similar to those of SIW_Φ, as captured in Theorem 21, with precedence tests that can be done in constant time:

Theorem 36. *Let R_Φ be a well-formed sketch for a class $\mathcal{Q}$ of problems. If the sketch width $w_R(\mathcal{Q})$ is bounded by k, SIW_R solves any problem P in $\mathcal{Q}$ in $O(bN^{|\Phi|+2k-1})$ time and $O(bN^k + N^{|\Phi|+k})$ space, where b and N bound the branching factor and number of atoms in P respectively.*

Example. Different and interesting sketches are given in Table 1 for the two classes of problems for Delivery: the

[7] For testing if $f' \prec f$ is true, one needs to check if there is a sequence $\{f_i\}_{i=0}^n$ of feature valuations such that $f = f_0$, $f' = f_n$, and each pair (f_i, f_{i+1}) is compatible with a sketch rule, $i = 0, 1, \ldots, n-1$. Actually, this test can be done in constant time too, provided that the binary relation '$\prec$' for each instance P is precompiled in a boolean hash table with N^k rows and N^k columns where N is the number of atoms in P, N^k is the number of feature valuations in P, and k is the number of features. Unlike the procedure SIW_R, this precompilation however is not practical in general. The efficiency of SIW_R comes at a price: by testing "progress" with the sketch rules directly and not with the serializations that results from such rules, SIW_R is not using the serialization fully, as it ignores the transitive closure of the precedence relations.

Policy sketch	$\mathcal{Q}_{D_1}$	$\mathcal{Q}_D$
$\sigma_0 = \{\}$	2	unb
$\sigma_1 = \{\{H\} \mapsto \{\neg H, p?, t?\}\}$	2	unb
$\sigma_2 = \{\{\neg H\} \mapsto \{H, p?, t?\}\}$	1	unb
$\sigma_3 = \sigma_1 \cup \sigma_2$	—	—
$\sigma_4 = \{\{n > 0\} \mapsto \{n\downarrow, H?, p?, t?\}\}$	2	2
$\sigma_5 = \sigma_2 \cup \sigma_4$	1	1
$\sigma_6 = \{\{\neg H, p > 0\} \mapsto \{p\downarrow, t?\}\}$	2	unb
$\sigma_7 = \{\{H, t > 0\} \mapsto \{t\downarrow, p?\}\}$	2	unb
$\sigma_8 = \sigma_2 \cup \sigma_4 \cup \sigma_6 \cup \sigma_7$	0	0

Table 1: Upper bounds on the width of different sketches for the classes $\mathcal{Q}_{D_1}$ and $\mathcal{Q}_D$ of Delivery problems. The entries unb and '—' mean, respectively, unbounded width and ill-defined sketch. For sketches of bounded width, SIW_R solves any instance in the class in polynomial time.

class $\mathcal{Q}_D$ of problems with an arbitrary number of packages and the class $\mathcal{Q}_{D_1}$ of problems with a single package. In the table, the entries in the columns $\mathcal{Q}_{D_1}$ and $\mathcal{Q}_D$ upper bound the width of the different sketches in the table for the two classes of Delivery problems. The entries unb and '—' stand respectively for unbounded width and ill-defined (non-terminating) sketch. The features used are: (boolean) H for holding a package, p is distance to nearest package (zero if holding a package or no package to be delivered remains, t is distance to current target cell (zero if holding nothing), and n is number of packages still to be delivered.

We briefly explain the entries in the table without providing formal proofs (such proofs can be obtained with Theorem 17). σ_0 is the empty sketch whose width is the same as the plain width, 2 for D_1 and unbounded for D, as no problem P is decomposed into subproblems. The rule $\{H\} \mapsto \{\neg H, p?, t?\}$ in σ_1 does not help in initial states that do not satisfy H, and hence the width of D_1 remains 2. For σ_2, the rule $\{\neg H\} \mapsto \{H, p?, t?\}$ 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. Hence, any problem P in D_1 is split in two subproblems: achieve H first and then the goal, reducing the sketch width of D_1 to 1 but not the sketch width of D. The sketch σ_3 is not well-formed as it is not terminating, and indeed, the resulting ordering is not a strict partial order. The sketch σ_4 decomposes the problems using the feature n that counts the number of undelivered packages, reducing width of D to 2 but not affecting that of D_1. The sketch σ_5 combines the rules in σ_2 and σ_4, decomposing D into two subproblems, the first of which corresponds to D_1, which is further decomposed into two subproblems. The sketch width of both D and D_1 becomes then 1: the first subproblem is to collect the nearest package, the second one to drop it at the target cell, and the same sequence of subproblems gets repeated. The sketches σ_6 and σ_7 simply some of the subproblems but do not reduce the widths of either D_1 or D. Finally, the sketch σ_8 yields a serialization of width zero, and hence a full policy. where each subproblem is solved in a single step. ∎

Thm	Notes
2	Performance and guarantees of IW(k).
10	Optimal, Markov policies bound width if features encoded.
11	Markovian policies guarantee optimal solutions with IW_Φ.
12	Bounded width also when tuples capture the features in all optimal trajectories.
17	Bounded width when tuples capture features in a projection.
21	Performance and guarantees of SIW_Φ.
24	Conditions for termination of policies.
26	Closed and terminating policies define structural solutions.
27	Terminating policies define serializations.
31	Structural solutions that are sound and closed define serializations of zero width.
33	Well-formed sketches define serializations.
35	Sketch width bounds width of induced serialization.
36	Performance and guarantees of SIW_R given sketches.

Table 2: Summary of main formal results.

Conclusions

We have established a number of connections between the notions of width, as developed and used in classical planning, and the notion of generalized plans, which are summarized in Table 2. The results suggest a deep connection between these two notions and that bounded width for infinite collections of problems $\mathcal{Q}$ is often the result of simple general policies that solve $\mathcal{Q}$ optimally in terms of features that are partially represented in the problem encodings. When this is not the case, we have shed light on the representations that deliver such properties, and hence, polynomial-time searches. We have also formalized and generalized the notion of serialized width by appealing to an explicit and abstract notion of serializations, and established connections between generalized policies and serialized width by borrowing notions from QNPs. From this connection, we introduced policy sketches which make use of the language of policy rules but with a different semantics for providing a convenient and powerful language for specifying subproblems and serializations that can be exploited by algorithms such as SIW. The language can be used for encoding domain-specific knowledge by hand, or as a target language for learning domain serializations automatically from traces. These are interesting challenges that we would like to address in the future.

Acknowledgments

The research is partially funded by an ERC Advanced Grant (No 885107), by grant TIN-2015-67959-P from MINECO, Spain, and by the Knut and Alice Wallenberg (KAW) Foundation through the WASP program. H. Geffner is a also Wallenberg Guest Professor at Linköping University, Sweden.

References

Bandres, W.; Bonet, B.; and Geffner, H. 2018. Planning with Pixels in (Almost) Real Time. In *Proc. AAAI*, 6102–6109.

Belle, V.; and Levesque, H. J. 2016. Foundations for Generalized Planning in Unbounded Stochastic Domains. In *Proc. KR*, 380–389.

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47: 253–279.

Bonet, B.; Frances, G.; and Geffner, H. 2019. Learning features and abstract actions for computing generalized plans. In *Proc. AAAI*, 2703–2710.

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

Bonet, B.; and Geffner, H. 2020a. General Policies, Serializations, and Planning Width. *arXiv:2012.08033*.

Bonet, B.; and Geffner, H. 2020b. Qualitative Numeric Planning: Reductions and Complexity. *Journal of Artificial Intelligence Research (JAIR)* 69: 923–961.

Bonet, B.; Palacios, H.; and Geffner, H. 2009. Automatic Derivation of Memoryless Policies and Finite-State Controllers Using Classical Planners. In *Proc. ICAPS*, 34–41.

Dechter, R. 2013. *Reasoning with probabilistic and deterministic graphical models: Exact algorithms*. Morgan & Claypool Publishers.

Francès, G.; Ramírez, M.; Lipovetzky, N.; and Geffner, H. 2017. Purely Declarative Action Representations are Overrated: Classical Planning with Simulators. In *Proc. IJCAI*, 4294–4301.

Freuder, E. C. 1982. A sufficient condition for backtrack-free search. *Journal of the ACM* 29(1): 24–32.

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

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

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

Lehman, J.; and Stanley, K. O. 2011a. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation* 19(2): 189–223.

Lehman, J.; and Stanley, K. O. 2011b. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proc. 13th annual conference on Genetic and evolutionary computation*, 211–218.

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

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

Lipovetzky, N.; and Geffner, H. 2017b. A polynomial planning algorithm that beats LAMA and FF. In *Proc. ICAPS*, 195–199.

Lipovetzky, N.; Ramirez, M.; and Geffner, H. 2015. Classical Planning with Simulators: Results on the Atari Video Games. In *Proc. IJCAI*, 1610–1616.

Ostrovski, G.; Bellemare, M. G.; Oord, A.; and Munos, R. 2017. Count-Based Exploration with Neural Density Models. In *Proc. ICML*, 2721–2730.

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

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann.

Segovia, J.; Jiménez, S.; and Jonsson, A. 2016. Generalized planning with procedural domain control knowledge. In *Proc. ICAPS*, 285–293.

Srivastava, S.; Immerman, N.; and Zilberstein, S. 2008. Learning generalized plans using abstract counting. In *Proc. AAAI*, 991–997.

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

Tang, H.; Houthooft, R.; Foote, D.; Stooke, A.; Chen, O. X.; Duan, Y.; Schulman, J.; DeTurck, F.; and Abbeel, P. 2017. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in neural information processing systems*, 2753–2762.

Thiébaux, S.; Hoffmann, J.; and Nebel, B. 2005. In defense of PDDL axioms. *Artif. Intell.* 168(1-2): 38–69.