Learning General Policies From Examples

Blai Bonet[1] , **Hector Geffner**[2]
[1]Universitat Pompeu Fabra, Spain
[2]RWTH Aachen University, Germany
bonetblai@gmail.com, hector.geffner@ml.rwth-aachen.de

Abstract

Combinatorial methods for learning general policies that solve large collections of planning problems have been recently developed. One of their strengths, in relation to deep learning approaches, is that the resulting policies can be understood and shown to be correct. A weakness is that the methods do not scale up and learn only from small training instances and feature pools that contain a few hundreds of states and features at most. In this work, we propose a new symbolic method for learning policies based on the generalization of sampled plans that ensures structural termination and hence acyclicity. The proposed learning approach is not based on SAT/ASP, as previous symbolic methods, but on a hitting set algorithm that can effectively handle problems with millions of states, and pools with hundreds of thousands of features. The formal properties of the approach are analyzed, and its scalability is tested on a number of benchmarks.

1 Introduction

The problem of learning policies that solve large collections of planning problems is an important challenge in both planning and reinforcement learning. Symbolic methods yield general policies that can be shown to be correct but are limited to small training instances and feature pools containing at most hundreds of states and features (Francès, Bonet, and Geffner 2021). Deep learning methods, on the other hand, do not require feature pools and scale up gracefully, yet the resulting policies are opaque and, in general, do not generalize equally well (Toyer et al. 2020; Ståhlberg, Bonet, and Geffner 2022).

The aim of this work is to develop a different way of learning general policies in the symbolic setting that scales up to much larger training instances and feature pools, including millions of states and hundreds of thousands of features. This scalability is also needed to address a limitation that is shared by the symbolic and deep learning approaches, and which has to do with the type of state features that can be computed. When description logic grammars or graph neural networks are used, the only logical features that can be captured are those that can be defined in first-order logic with two variables and counting quantifiers (Barceló et al. 2020; Grohe 2021). Addressing this limitation in the symbolic setting, calls for novel and richer feature grammars that result in larger feature pools, thus requiring more scalable learning algorithms.

Previous symbolic methods like (Francès, Bonet, and Geffner 2021; Drexler, Seipp, and Geffner 2022) scale up poorly because they reduce the task of learning the features and the policy to a combinatorial optimization problem that is cast and solved by weighted-SAT or ASP solvers. In these settings, relaxing the optimality criterion, or some of the constraints, yield policies that do not generalize well outside the training set. In this work, we use an scalable, heuristic algorithm for min-cost hitting set problems as the basis of a new procedure for learning general policies.

For obtaining the new formulation, two ingredients are needed. First, a classical planner that generates plans which are then generalized. Second, a new structural termination criterion that ensures that the generalization does not introduce cycles. Provided with this guarantee, the plan generalizations become fully general policies when they are also *closed* and *safe*; meaning that they do not reach states where the policy is undefined or which are dead ends, respectively. The resulting algorithm consists of an efficient, polynomial-time, core algorithm, based on min-cost hitting sets, that yields a policy that generalizes given sets of positive and negative state transitions, $\mathcal{X}^+$ and $\mathcal{X}^-$, and a wrapper algorithm that manipulates these two sets until the resulting policy is closed and safe, and hence correct. Interestingly, when the algorithm *fails* to produce a correct policy, the reasons for the failure can be understood, and sometimes, fixed.

The rest of the paper is structured as follows. We discuss related work, cover relevant background, introduce the new termination criterion, and present the resulting formulation and algorithm, assessing its properties and its performance.

2 Related Work

General policies. The problem of learning general policies has a long history (Khardon 1999; Fern, Yoon, and Givan 2006; Srivastava, Immerman, and Zilberstein 2008; Hu and De Giacomo 2011; Belle and Levesque 2016; Illanes and McIlraith 2019; Celorrio, Segovia-Aguas, and Jonsson 2019). General policies have been formulated in terms of first-order logic (Srivastava, Immerman, and Zilberstein 2011; Illanes and McIlraith 2019), first-order regression (Boutilier, Reiter, and Price 2001; Wang, Joshi, and Khardon 2008; Sanner and Boutilier 2009), and neural networks (Groshev et al. 2018; Toyer et al. 2018; Bueno et al. 2019; Rivlin, Hazan, and Karpas 2020; Karia, Nay-

yar, and Srivastava 2022; Ståhlberg, Bonet, and Geffner 2023). Our work builds on formulations where the features and policies defined on such features and rules are learned using SAT encodings (Bonet and Geffner 2018; Francès, Bonet, and Geffner 2021), and is also related to early supervised approaches that use polynomial algorithms and explicit feature pools (Martín and Geffner 2004).

QNPs, Termination, Acyclicity. While a policy that is closed, safe, and acyclic must solve a problem, enforcing acyclicity is not easy computationally, as it is not a local property. Interestingly, structural criteria and algorithms that ensure acyclicity have been developed in the setting of qualitative numerical planning problems or QNPs (Srivastava et al. 2011; Bonet and Geffner 2020), which involve Boolean and numerical variables that can be increased and decreased by random amounts. A policy *terminates* in a QNP if all "fair" trajectories are finite, where trajectories where a variable increases *finitely* often and decreases infinitely often are regarded as "unfair", and can be ignored. An algorithm called SIEVE establishes termination in time that is exponential in the number of variables in the QNP. In this work a new termination criterion is introduced that is slightly weaker than SIEVE but that is more convenient and can be built into the selection of the features.

Imitation learning and inverse reinforcement learning (IRL). The use of sampled plans for learning general policies has been used in early work (Khardon 1999; Martín and Geffner 2004; Fern, Yoon, and Givan 2006), and its a common idea in imitation learning (Ng and Russell 2000; Ho and Ermon 2016). Learning to imitate plans or behavior in a mindless manner, however, prevents good generalization. The idea in IRL is to learn the reward distribution from the examples and then solve the underlying problem where these rewards are to be optimized. The problem in IRL is in the assumptions that need to be made so that the resulting task is well-posed. In this work, we also aim to go beyond the examples (plans) and solve the more general problem that the examples illustrate. The task is not just to generalize the given plans, but to obtain a policy that is structurally terminating and which hence must converge to a goal.

3 Background

We review classical planning, generalized planning, rule-based policies, learning these policies, and termination, following (Francès, Bonet, and Geffner 2021; Bonet and Geffner 2024).

3.1 Planning and Generalized Planning

We deal with planning instances $P = \langle D, I \rangle$ where D is a **domain** specification containing object types, constants, predicate signatures, and action schemas, and I is an **instance** specification containing the objects and their types, and the description of the initial and goal situations, I and G respectively, both as sets of ground atoms.

A state trajectory in P seeded at s_0 is a state sequence $\tau = \langle s_0, s_1, s_2, \ldots \rangle$ such that for each transition (s_i, s_{i+1}), there is a ground action that maps s_i to s_{i+1}. A state s is **reachable** iff there is a trajectory seeded at the initial state that ends in s; it is a **dead end** iff it is reachable, and there is no state trajectory seeded at s that ends in a goal state; and it is **alive** iff it is reachable, and it is not a goal nor a dead-end state. The task for P is to find a trajectory seeded at the initial state that ends in a goal state, or declare that no such trajectory exist; a sequence of ground actions that map each state into the next in such a trajectory is a **plan.**

Semantically, a policy π for P is a **set** of state transitions (s, t) in P. A π-trajectory from state s_0 is a state trajectory $\tau = \langle s_0, s_1, s_2, \ldots \rangle$ such that (s_i, s_{i+1}) is in π. The states in the trajectory are said to be π-reachable in P. The policy π **solves** P iff each **maximal** π-trajectory from the initial state is finite and ends in a goal state. The policy π is **closed** if for each alive state s that is π-reachable, there is π-transition (s, s'); it is **safe** if it does not reach a dead end; and it is **acyclic** if there is no **infinite** π-trajectory seeded at the initial state. The following characterizes policies that solve P:

Theorem 1 (Solutions for P)**.** *Let P be a planning instance, and let π be a policy for P. Then, π solves P iff π is closed, safe, and acyclic in P.*

3.2 Generalized Planning

Generalized planning deals with the computation of policies that solve collections of planning instances rather than just a single planning instance. A collection of planning instances is a, finite or infinite, set $\mathcal{Q}$ of instances $P_i = \langle D, I_i \rangle$ over a common domain D. In some cases, all the instances in $\mathcal{Q}$ have the same or similar goal, like achieving a specific atom, but this is not required, nor assumed.

Semantically, a policy π for $\mathcal{Q}$ represents a subset of state transitions in each instance P in $\mathcal{Q}$. The policy π solves $\mathcal{Q}$ iff it solves each instance P in $\mathcal{Q}$. Notions about states like reachable, alive, goal, etc., and about policies like closed, safe, etc., are naturally lifted from P into $\mathcal{Q}$. A similar characterization for solutions for $\mathcal{Q}$ applies:

Theorem 2 (Solutions for $\mathcal{Q}$)**.** *Let $\mathcal{Q}$ be a collection of planning instances, and let π be a policy for $\mathcal{Q}$. Then, π solves $\mathcal{Q}$ iff π is closed, safe, and acyclic in $\mathcal{Q}$.*

3.3 Features

General policies can be represented in terms of collections of feature-based rules. Features are functions that map states into values. Boolean features take values in $\{0, 1\}$, and numerical features take values in the non-negative integers. In logical accounts, features are commonly defined in terms of concepts (unary predicates), which are **generated** from sets of **atomic** concepts and roles, using description logic grammars (Martín and Geffner 2004; Fern, Yoon, and Givan 2006; Bonet, Frances, and Geffner 2019). The **denotation** of concept C (resp. role R) in a state s is a set of objects (resp. pairs of objects) from s, denoted by $C(s)$ (resp. $R(s)$). Concept C defines a numerical feature f_C whose value at s is the cardinality $|C(s)|$ of the set $C(s)$. When the value of f_C is always in $\{0, 1\}$, the concept defines a Boolean rather than a numerical feature.

For a domain D, the atomic concepts (resp. roles) are given by the object types, constants, and unary predicates

(resp. the binary predicates) in D. A **pool of features** $\mathcal{F}$ can be generated using the domain D, and parameters that bound the maximum complexity and depth for the features in $\mathcal{F}$. The generation process is given a set $\mathcal{T}$ of transitions over instances in D to **prune redundant** features; namely, if $\mathcal{S}$ is the set of states mentioned in the transitions in $\mathcal{T}$, a feature f is redundant if there is a feature g of lesser complexity, or equal complexity but earlier in a static ordering, such that both are $\mathcal{S}$-equivalent, or $\mathcal{T}$-equivalent. Feature f is $\mathcal{S}$-equivalent to g iff $f(s) = g(s)$ for each state s in $\mathcal{S}$, and f is $\mathcal{T}$-equivalent to g if for each (s, t) in $\mathcal{T}$, both have the same Boolean valuation at s, and both change equivalently across (s, t) (i.e., $f(s) > 0$ iff $g(s) > 0$, and both increase/decrease/stay equal across (s, t)).

In the generalized planning setting, states s for an instance P are assumed to contain the description of the goal in P via **goal predicates** p_g (Martín and Geffner 2004), one for each predicate p in D with denotation $\{\bar{u} \mid G \models p(\bar{u})\}$. These predicates allow policies to work for different goals G.

Example. *Let us consider the domain for Blocksworld with 4 operators (i.e., with a gripper) and an instance P whose goal description is $G = \{clear(A)\}$. The domain description contains the predicates $clear/1$ and $on/2$. The following concepts are generated by the grammar:*

- *'$clear_g$' whose denotation is the singleton with block A,*
- *'$\exists on.\top$' whose denotation consists of the blocks that rest on another block ($\top$ is the concept that includes all objects), and*
- *'$\exists on^+.clear_g$' whose denotation consists of the blocks that are above block A.*

The second concept defines a feature that counts the number of blocks that rest on another block, while the third defines a feature that counts the number of blocks above the "target" block A. □

For a set $\mathcal{G}$ of features, a **Boolean valuation** ν is a function $\nu : \mathcal{G} \to \{0, 1\}$ that assigns a Boolean value to all the features in a state, whether Boolean or numerical, as $\nu(f) = 0$ if $f(s) = 0$, and $\nu(f) = 1$ if $f(s) > 0$. In these valuations, the exact value of numerical features is abstract away, replaced by a Boolean that is true iff the value is strictly positive.

3.4 Rule-based Policies

While semantically, policies select subset of state transitions in each instance P in $\mathcal{Q}$, syntactically, policies are represented by collection of **rules** over a given set of features $\mathcal{F}$. A policy rule $C \mapsto E$ consists of a condition C, and an effect E, where C contains expressions like p and $\neg p$ for Boolean features p, and $n = 0$ and $n > 0$ for numerical features n. The effect E in turn contains expressions like p, $\neg p$, and $p?$ for Booleans p, and $n\uparrow$, $n\downarrow$, and $n?$ for numericals n.

A set of rules defines a policy π where a state transition (s, t) is a π-transition if it is compatible with one of the rules $r = C \mapsto E$. This is true if the feature conditions in C are true in s, and the features change in the transition in a way that is compatible with E; i.e., (s, t) is compatible with r iff

- if p (resp. $\neg p$) in C, $p(s)$ (resp. $\neg p(s)$),
- if p (resp. $\neg p$) in E, $p(t)$ (resp. $\neg p(t)$),
- if $\{p, \neg p, p?\} \cap E = \emptyset$, $p(s)$ iff $p(t)$,
- if $n = 0$ (resp. $n > 0$) in C, $n(s) = 0$ (resp. $n(s) > 0$),
- if $n\uparrow$ (resp. $n\downarrow$) in E, $n(s) < n(t)$ (resp. $n(s) > n(t)$), and
- if $\{n\uparrow, n\downarrow p, n?\} \cap E = \emptyset$, $n(s) = n(t)$.

The transition (s, t) is in π if it is compatible with some rule r in π; we use notations like $(s, t) \in \pi$, and $\{(s, t)\} \subseteq \pi$.

Feature-based rules permit the representation of general policies that are not tied to a particular set of instances, as those used for learning such a policy, as the features and rules can be evaluated on any instance for the shared domain.

Example. *A general policy for Blocksworld instances with a gripper and goal descriptions of the form $G = \{clear(A)\}$, for arbitrary block A, can be expressed with only two rules $\{\neg H, n > 0\} \mapsto \{H, n\downarrow\}$ and $\{H\} \mapsto \{\neg H\}$, where H is a Boolean feature that tells whether a block is being held, and n counts the number of blocks above the target block (i.e., the block mentioned in G). The first rule says to pick block above the target when holding nothing, while the second to put the block being held somewhere* not *above the target (this last condition is achieved as the effect of the second rule requires n to be remain constant).*

Notice, however, that if the second rule is replaced by $\{H\} \mapsto \{\neg H, n?\}$, the resulting policy does not solve such instances as it can generate infinite trajectories where the same block is picked and put back above the target block, repeatedly. The policy above cannot generate infinite trajectories, independently of the interpretation of H and n; we say that it is structurally terminating. □

3.5 Learning Rule-based Policies

The new method for learning general policies builds on the SAT approach developed by Francès, Bonet, and Geffner (2021) that constructs a CNF theory $\text{Th}(\mathcal{T}, \mathcal{F})$ from a set $\mathcal{T}$ of state transitions (s, t) in a collection $\mathcal{Q}'$ of training instances, and a pool $\mathcal{F}$ of Boolean and numerical features. The theory contains propositions $good(s, t)$ and $select(f)$ that tell which transitions (s, t) in $\mathcal{T}$ and features f in $\mathcal{F}$ should be included in the resulting policy π. The policies π over the features in $\mathcal{F}$ that solve $\mathcal{Q}'$ are in *correspondence* with the models of $\text{Th}(\mathcal{T}, \mathcal{F})$. Indeed, the rules in π are determined by the true $good(s, t)$ and $select(f)$ atoms: each good transition (s, t) yields a rule $C \mapsto E$ where C captures the Boolean valuations of the selected features at state s, and E captures the changes of such features across (s, t). Such a policy rule is called the **projection** of the transition (s, t) over the set of selected features.

By constructing a policy π^* from a satisfying assignment of *minimum cost*, as determined by the complexity of the selected features, the policy π^* is expected to generalize over the entire class $\mathcal{Q}$ from which the training instances in $\mathcal{Q}'$ have been drawn. While correct generalization is not guaranteed, this can be established by manually analyzing π^*.

The key constraints in the SAT theory, expressed with the $good(s, t)$ and $select(f)$ atoms, ensure that any resulting policy π is **closed**, **safe**, and **acyclic** in $\mathcal{Q}'$. For this, the *full state space* of such instances, states and transitions, need to

be calculated and represented in $\mathrm{Th}(\mathcal{T}, \mathcal{F})$ as well as all the features in the pool, making the approach only feasible for small state spaces and feature pools.

3.6 Termination

The idea of structural termination in settings where variables can be increased and decreased "qualitatively" (i.e., by random amounts) is that certain state trajectories are not possible if the variables have minimal and maximal values in each instance, and the changes cannot be infinitesimally small (Srivastava et al. 2011; Bonet and Geffner 2020). In the generalized planning setting, it is the numerical features that change in this way. An infinite state trajectory where a numerical feature is increased *finitely* often, and decreased *infinitely* often, or vice versa, is not possible. If the infinite state trajectories generated by a policy all have this form, the policy must be *acyclic.*

This notion of termination is particularly interesting because it comes up with a sound and complete algorithm for checking the termination of a given policy π, called SIEVE (Srivastava et al. 2011) that runs in time $\mathcal{O}(2^n)$ where n is the number of features in π, and which works on the so-called *policy graph,* whose nodes are the possible Boolean valuations of the features in π. Indeed, while acyclicity is property of the policy π in a particular instance P, termination is a property of the rules in π that ensures acyclicity for *any* instance. The notion of *stratified rule-based policies* introduced below provides a novel twist to this idea which is more convenient for learning policies that are **terminating by design.** Indeed, the new structural termination criterion runs more efficiently than SIEVE and can be compiled into the procedure that selects the policy features given the good transitions, ensuring that the resulting policy is terminating.

4 The Plan

The approach for learning general policies via SAT reviewed above is simple and elegant but does not scale up. The new learning method can be thought as a simplification where:

1. The "good" state transitions are not selected by the SAT solver but incrementally, by a *planner*.
2. The acyclicity constraint, which is global and hard to enforce, is replaced by a *new termination criterion* that is enforced implicitly in the selection of the features.
3. The selection of the features is carried out by a scalable *hitting set algorithm* and not by SAT.

The three elements are all critical for the performance of the resulting learning algorithm. The second and third elements are addressed by an algorithm that we call GENEX, for *gen*eralization from *ex*amples, and that can handle sets of state transitions and features that are orders of magnitude larger that the ones handled by current approaches. The first element, on the other hand, that uses a planner to select transitions, is addressed by a simple WRAPPER algorithm around GENEX.

An essential idea of the new method is the *decoupling* of the two selections involved in the computation of general policies: the selection of the "good" transitions in the training set, and the selection of the features. In the SAT approach these two decisions are coupled, ensuring completeness: if there is a general policy over the features that solves the training instances, the SAT approach would find it. This guarantee is now gone, replaced by the decoupling that enables scalability. The experiments below evaluate this trade off.

The next sections introduce the new termination criterion, a new basic learning algorithm that yields terminating policies that generalizes given sets of "good" and "bad" transitions, and a wrapping mechanism that extends this policy to be safe and closed.

5 Termination Revisited

A new notion of termination is introduced by means of **stratified policies**. These are **rule-based policies** whose features can be layered up in such a way, that features in the first layer can be shown to be terminating without having to consider other features, while features in successive layers can be shown to be terminating given features in previous layers. A **feature f is terminating** in a policy π if it cannot keep changing values forever; namely, if the number of times that f changes value in a π-trajectory is finite. The policy π is terminating if all the features involved are terminating.

We take advantage of the assumption that the numerical features are non-negative integer valued and upper bounded in any instance. Hence, a trajectory where a feature is increased (resp. decreased) infinitely often but decreased (resp. increased) finitely often is not possible. In the definitions, Boolean features are treated as numerical features with decrements and increments referring to value changes from from 1 to 0, and from 0 to 1, respectively.

5.1 Stratified Policies

The first class of terminating features are the features that are **monotone** in the set of (policy) rules R, meaning that R does not contain rules that increase and that decrease f.

Definition 3 (Monotone Features)**.** *Let $\mathcal{F}$ be a set of features, let R be a set of policy rules over $\mathcal{F}$, and let f be a feature that appears in some rule in R. Then, f is **monotone in** R iff either there is no rule in R that increases f, or there is no rule in R that decreases f.*

Clearly, a monotone feature can only change value a finite number of times along a given trajectory, as it eventually reaches a minimum or maximum value, and stays put.

The second class of terminating features f are those that are rendered monotone by other monotone features g. Indeed, since g can change a finite number of times, f will be **monotone given** g if f is monotone over the rules that do not change the value of g and which share the same value of g in the conditions.

To capture this form of **conditional monotonicity**, we define the following rule subsets from R and a given feature g:

$$\begin{aligned}
\varrho(R, g, =) &\doteq \{r \in R \mid \{g\uparrow, g\downarrow\} \cap \mathrm{eff}(r) = \emptyset\}\,, \\
\varrho(R, g, 0) &\doteq \{r \in \varrho(R, g, =) \mid \text{`}g > 0\text{'} \notin \mathrm{cond}(r)\}\,, \\
\varrho(R, g, 1) &\doteq \{r \in \varrho(R, g, =) \mid \text{`}g = 0\text{'} \notin \mathrm{cond}(r)\}\,.
\end{aligned}$$

The intuitions for these subsets is that when a transition (s,t) is compatible with a rule r, then

- $r \in \varrho(R,g,=)$ iff g may remain unchanged across (s,t),
- $r \in \varrho(R,g,0)$ iff $r \in \varrho(R,g,=)$ and $g=0$ may hold at s,
- $r \in \varrho(R,g,1)$ iff $r \in \varrho(R,g,=)$ and $g>0$ may hold at s.

Conditional monotonicity is defined as follows:

Definition 4 (Conditional Monotonicity)**.** *Let $\mathcal{F}$ be a set of features, let R be a set of policy rules over $\mathcal{F}$, and let f and g be features that are mentioned in R. Then, f is **monotone in** R **given** g iff f is monotone in $\varrho(R,g,0)$ **and** f is monotone in $\varrho(R,g,1)$.*

A set of rules R encodes a **stratified policy** if the features in the policy can be ordered in such a way that features f with a positive rank $\kappa(f)$ are monotone given features g of lower rank.

Definition 5 (Stratified Policies)**.** *Let π be a rule-based policy over a set $\mathcal{F}$ of features. Then, π is a **stratified** iff*

1. *Each rule in π **entails the change** of some feature f, and*
2. *There is a ranking κ for the features in π such that*
   - 2a. *If $\kappa(f)=0$, then f is **monotone in** π, and*
   - 2b. *If $\kappa(f)>0$, then there is feature g such that $\kappa(g)<\kappa(f)$ and f is **monotone in** π **given** g.*

*Where a rule $C \mapsto E$ **entails the change** of feature f iff f is numerical and $E \cap \{f\uparrow, f\downarrow\} \neq \emptyset$, or f is Boolean and either $p \in C \wedge \neg p \in E$, or $\neg p \in C \wedge p \in E$.*

A stratified policy cannot generate infinite trajectories just due to its form, without regard for the class of instances $\mathcal{Q}$ where it is applied, or the interpretation of the features; i.e.,

Theorem 6 (Termination)**.** *Let π be a rule-based policy over a set of features $\mathcal{F}$. If π is stratified, π is terminating.*

Example. *We illustrate a terminating policy for the Gripper domain which involves a robot that must move balls from room A to room B, using two grippers. The rooms A and B are declared in the domain specification as **constants,** thus shared by all instances and identified with* nominal concepts. *The policy π is defined over the features: A that tells whether the robot is in room A, m that counts the number of balls being held, and n that counts the number of objects in room A. The rules are:*

$$\begin{aligned} r_1 &: \quad \{n>0\} \mapsto \{n\downarrow, m?\}, \\ r_2 &: \quad \{m>0\} \mapsto \{m\downarrow\}, \\ r_3 &: \{A, m>0\} \mapsto \{\neg A\}, \\ r_4 &: \{\neg A, m=0\} \mapsto \{A\}. \end{aligned}$$

The first rule says to decrease the number of balls in room A (i.e., to pick them up) whenever possible, perhaps affecting m; the second to drop balls somewhere while not affecting n (the only way to achieve this is to drop balls in room B); the third rule says to move from room A to B when holding a ball, and the last one to move from B to A when holding nothing. This is a general policy that moves all balls in room A to room B, and that works for any number of balls.

*The policy is stratified and thus terminating (cf. Def. 5 and Thm. 6). The feature n is monotone in π as no rule increases it, m is monotone in π **given** n as it is monotone in $\varrho(\pi,n,0)=\varrho(\pi,n,1)=\{r_2,r_3,r_4\}$, and A is monotone in π **given** m as it is monotone in $\varrho(\pi,m,0)=\{r_1,r_4\}$ and it is monotone in $\varrho(\pi,m,1)=\{r_1,r_4\}$.* □

5.2 k-Stratified Policies

Stratified policies can be generalized to cases where the monotonicity of f depends on multiple features of lower rank. For this, we need to define when a feature f is monotone given a **set** G of features, and this requires the consideration of Boolean valuations over the features. Recall that a Boolean valuation assigns a Boolean value $\nu(g)$ in $\{0,1\}$ for every feature g, whether Boolean or numerical.

Definition 7 (k-Conditional Monotonicity)**.** *Let $\mathcal{F}$ be a set of features, let R be a set of policy rules over $\mathcal{F}$, and let f and G be a feature and a set of features, respectively, mentioned in R. Then, f is **monotone in** R **given** G iff for each Boolean valuation ν for G, f is monotone in $\varrho(R,G,\nu)$, where $\varrho(R,G,\nu) \doteq \cap\{\varrho(R,g,\nu(g)) \mid g \in G\}$.*

As G increases, more "contexts" $\varrho(R,G,\nu)$ need to be considered, but each context is smaller. Thus, the chances for f being monotone given G increase as G contains more features. Also, f is monotone given G' when it is monotone given G and $G \subseteq G'$. The definition of k-stratified policies can be similarly expressed as before:

Definition 8 (k-Stratified Policies)**.** *Let π be a rule-based policy defined over the features in $\mathcal{F}$, and let k be a positive integer. Then, π is a **k-stratified policy** iff*

1. *Each rule in π **entails the change** of some feature f, and*
2. *There is ranking κ for the features in π such that*
   - 2a. *If $\kappa(f)=0$, f is **monotone in** π, and*
   - 2b. *If $\kappa(f)>0$, there are features $G=\{g_1,g_2,\ldots,g_\ell\}$, with $\ell \leq k$, such that $\max\{\kappa(g) \mid g \in G\} < \kappa(f)$ and f is **monotone in** π **given** G.*

It is not difficult to show that for any integer $k>1$, there are policies that are k-stratified but not $(k-1)$-stratified, and that there are policies that are terminating, but not k-stratified for any k. As before, k-stratified policies are terminating too:

Theorem 9 (Termination of k-Stratified Policies)**.** *Let π be a rule-based policy defined on a set of features $\mathcal{F}$, and let k be a positive integer. If π is k-stratified, π is terminating.*

Proof (sketch). Let π be a k-stratified policy, and let κ be a suitable ranking for π. Let us suppose that π is not terminating; i.e., there is an infinite trajectory of Boolean valuations over the features that is compatible with π.

Since for each transition compatible with π, there is a feature that changes across the transition, there must be a feature f in π that is increased **and** decreased infinitely often in τ. Let f be such a feature of **minimum rank.** Clearly, $\kappa(f)$ cannot be zero as such features are monotone in π. Therefore, there is a set G with at most k features and $\max\{\kappa(g) \mid g \in G\} < \kappa(f)$ such that f is monotone in π given G. This means that f is monotone in $\varrho(\pi,G,\nu)$ for all the Boolean valuations ν of the features in G. Therefore, there are at least two valuations ν_0 and ν_1 for G that appear

infinitely often in τ. Hence, there is some feature g in G whose Boolean value **flips** infinitely often in τ, which implies that g is increased **and** decreased infinitely often in τ. This contradicts the choice of f because $\kappa(g) < \kappa(f)$. □

At the same time, checking k-stratification is exponential in k and not exponential in the number of features like SIEVE. This is important because the notion of termination captured by k-stratification for a low value of k is most often powerful enough. Indeed, our learning algorithm uses $k = 1$.

Theorem 10 (Testing k-Stratification)**.** *Let π be rule-based policy defined on a set of features $\mathcal{F}$, and let k be a positive integer. Testing whether π is k-stratified can be done in time that is exponential in k, but polynomial in $|\pi|$ and $|\mathcal{F}|$, where $|\pi|$ refers to the number of rules in π.*

Proof (sketch). In order to check that π is k-stratified, one needs to "construct" a suitable ranking κ. The construction proceeds in stages, first identifying the features of rank 0, then those of rank 1, etc. First, at stage 0, the monotone features f in π are identified in linear time and assigned $\kappa(f) = 0$. Then, at stage ℓ, each feature f not yet ranked is tested whether there is a subset G of at most k features of rank less than ℓ such that f is monotone in π given G. The number of subsets to try is $\mathcal{O}(n^k)$, where n is the number of features in π, and for each such candidate, $\mathcal{O}(2^k)$ contexts must be considered. Hence, testing whether a new feature f can be assigned a rank requires time that is exponential only in k. This check must be repeated $\mathcal{O}(n)$ times as there are n features, and for at most $\mathcal{O}(n)$ stages. □

6 Basic Learning Task and GENEX

We formulate the problem of learning generalized policies from examples (plans) in two parts:

1. **Basic learning task (BLT):** Given sets $\mathcal{X}^+$ and $\mathcal{X}^-$ of "good" and "bad" state transitions, respectively, and a feature pool $\mathcal{F}$, the task is to find a **stratified** policy π over the features in $\mathcal{F}$ such that the good transitions are π-transitions, and no bad transition is a π-transition. In other words, the BLT task is about generalizing the good transitions, which may come from plans, while avoiding the bad transitions and ensuring termination.
2. **Meta learning task (MLT):** Find the sets of good and bad transitions so that the BLT returns a policy that is **closed** and **safe** over all instances P in a training set $\mathcal{Q}$.

We focus on the basic learning task in this section and on the meta learning task in the next one.

6.1 Task and Algorithm

The task is to learn a stratified policy that includes the good transitions, and excludes the bad transitions; formally,

Definition 11 (Basic Learning Task)**.** *Let $\mathcal{F}$ be a pool of features, and let $\mathcal{X}^+$ and $\mathcal{X}^-$ be sets of transitions, called "good" and "bad" transitions, respectively. Then, $BLT(\mathcal{F}, \mathcal{X}^+, \mathcal{X}^-)$ is the task of finding a **stratified policy** π over $\mathcal{F}$ such that $\mathcal{X}^+ \subseteq \pi$ and $\mathcal{X}^- \cap \pi = \emptyset$.*

The BLT is defined in this way, leaving aside **closedness** and **safeness**, because it can be cast as a **hitting set problem** that admits efficient algorithms.

In general, a hitting set problem is the tuple $\langle S, \mathcal{H}, c\rangle$ where S is a set of items, $\mathcal{H}$ is a collection of S-subsets, and $c : S \to \mathbb{N}^+$ is a cost function. The task is to find a min-cost subset $S' \subseteq S$ that "hits" every subset in $\mathcal{H}$ (i.e., $S' \cap S_i \neq \emptyset$ for $S_i \in \mathcal{H}$), where the cost of S' is $c(S') \doteq \sum_{i \in S'} c(i)$.

Definition 12 (Hitting Set Problem Induced by BLT)**.** *Let $\mathcal{F}$ be a pool of features, and let $\mathcal{X}^+$ and $\mathcal{X}^-$ be sets of good and bad transitions. The hitting set problem $H(\mathcal{F}, \mathcal{X}^+, \mathcal{X}^-)$ is the tuple $\langle \mathcal{F}, \mathcal{H}, c\rangle$ where the cost function c maps $f \in \mathcal{F}$ into its complexity, and $\mathcal{H}$ consists of the following $\mathcal{F}$-subsets:*

- *For each transition (s, t) in $\mathcal{X}^+$, $\mathcal{H}$ contains the set $\{f \in \mathcal{F} \mid f$ changes across $(s, t)\}$.*
- *For each (s, t) in $\mathcal{X}^-$ and each (s', t') in $\mathcal{X}^+$, $\mathcal{H}$ contains $\{f \in \mathcal{F} \mid f$ changes differently in (s, t) and $(s', t')\}$.*
- *For each pair (s, s') of goal and non-goal states in the transitions in $\mathcal{X}^+$, $\mathcal{H}$ contains the set $\{f \in \mathcal{F} \mid$ the Boolean valuation of f differs on $\{s, s'\}\}$.*

The last type of subsets is not needed for solving the BLT. However, policies that contain features that identify goal states tend to generalize better over new unseen instances.

From a solution $\mathcal{G}$ for $H(\mathcal{F}, \mathcal{X}^+, \mathcal{X}^-)$, one can construct a policy $\pi = \pi(\mathcal{G}, \mathcal{X}^+)$ whose rules $C \mapsto E$ are obtained by **projecting** the good transitions (s, t) in $\mathcal{X}^+$ over the features in $\mathcal{G}$, like in previous learning approaches.

GENEX, depicted in Alg. 1, is a standard **greedy algorithm** that solves $H(\mathcal{F}, \mathcal{X}^+, \mathcal{X}^-)$ by growing a hitting set $\mathcal{G}$. To guarantee completeness of the algorithm (cf. Theorem 13), the algorithm adds a set (chain) $C_f = \langle f_0, f_1, \ldots, f_{k+1} = f\rangle$ of features (of maximum score) that ends in f, and that provides complete conditional monotonicity for f: f_0 is monotone for (the transitions in) $\mathcal{X}^+$, and f_{i+1} is monotone for $\mathcal{X}^+$ given f_i, for $i = 0, 1, \ldots, k$. Stratification is guaranteed by ensuring that no choice of chains create a circular ordering among the chosen features. For this, each such chain C_f imposes an ordering constraints $f_i \prec f_{i+1}$ that are maintained in the set Ord (line 9) and that is queried for selecting features (line 7).

The greedy algorithm runs in low polynomial time in the size of $H(\mathcal{F}, \mathcal{X}^+, \mathcal{X}^-)$. More interesting is that it is sound and complete for the basic learning task:

Theorem 13 (Soundness and Completeness of GENEX)**.** *Let $\mathcal{F}$ be a pool of features, and let $\mathcal{X}^+$ and $\mathcal{X}^-$ be sets of good and bad transitions, respectively. If* GENEX *returns $\mathcal{G} \subseteq \mathcal{F}$ on input $H(\mathcal{F}, \mathcal{X}^+, \mathcal{X}^-)$, then*

1. *$\mathcal{G}$ is a hitting set for $H(\mathcal{F}, \mathcal{X}^+, \mathcal{X}^-)$, and*
2. *the policy $\pi = \pi(\mathcal{G}, \mathcal{X}^+)$ obtained by projecting the transitions in $\mathcal{X}^+$ over $\mathcal{G}$ solves BLT$(\mathcal{F}^+, \mathcal{X}^+, \mathcal{X}^-)$.*

Else, if GENEX *returns* FAILURE *on input $H(\mathcal{F}, \mathcal{X}^+, \mathcal{X}^-)$, then there is no solution π for BLT$(\mathcal{F}^+, \mathcal{X}^+, \mathcal{X}^-)$ whose features separate goal from non-goal states.*

Proof (sketch). **Soundness.** Let us assume that GENEX returns $\mathcal{G}$. Clearly, $\mathcal{G}$ is a hitting set for $H(\mathcal{F}, \mathcal{X}^+, \mathcal{X}^-)$.

Algorithm 1 GENEX for solving $H(\mathcal{F}, \mathcal{X}^+, \mathcal{X}^-)$.

Input: Hitting set problem $H = H(\mathcal{F}, \mathcal{X}^+, \mathcal{X}^-)$.

Output: FAILURE, or hitting set $\mathcal{G}$ for $H(\mathcal{F}, \mathcal{X}^+, \mathcal{X}^-)$.

1: For each f in $\mathcal{F}$, let cost$(f) \leftarrow$ complexity(f).
2: For each $f \in \mathcal{F}$, compute $C_f = \langle f_0, \ldots, f_{k+1} = f \rangle$ such that f_0 is monotone in $\mathcal{X}^+$, f_{i+1} is monotone in $\mathcal{X}^+$ given f_i, and C_f is of **minimum cost,** where the cost of C_f is $\sum_{i=0}^{k+1}$ cost(f_i).
3: Let $\mathcal{G} := \emptyset$
4: Let Ord $:= \emptyset$ be the empty ordering of chosen features. Feature f is **eligible** if Ord $\cup \{\langle g, g' \rangle \mid \langle g, g' \rangle \in C_f\}$ is acyclic. The set of eligible features is denoted by $\mathcal{E} = \mathcal{E}(\mathcal{F}, \text{Ord})$.
5: **while** $\mathcal{G}$ is not a solution for H **do**
6: Let $f^* \in \mathcal{E}$ be a feature of **max** score(f^*), where score$(f) = |\{S \in H \mid \mathcal{G} \cap S = \emptyset \wedge C_f \cap S \neq \emptyset\}|$ divided by the cost of C_f.
7: If $\mathcal{E} = \emptyset$ or score$(f^*) = 0$, **return** FAILURE
8: Let $\mathcal{G} \leftarrow \mathcal{G} \cup C_{f^*}$
9: Let Ord $\leftarrow$ Ord $\cup \{\langle g, g' \rangle \mid \langle g, g' \rangle \in C_{f^*}\}$
10: Set cost$(f) \leftarrow 0$ for $f \in C_{f^*}$.
11: Recompute chains given new cost of features in C_f
12: **end while**
13: **return** $\mathcal{G}$

Also, it is not hard to see that there is a ranking κ that renders $\pi(\mathcal{G}, \mathcal{X}^+)$ stratified. Indeed, since Ord remains acyclic, there is such ranking κ throughout the execution of the loop, at the start of each iteration.

Completeness. Let π be a stratified policy over $\mathcal{G} \subseteq \mathcal{F}$ such that $X^+ \subseteq \pi$, $\mathcal{X}^- \cap \pi = \emptyset$, and $\mathcal{G}$ separates goal from non-goal states. Moreover, let π be such a policy with a **minimum** number of rules; i.e., each rule in π is compatible with at least one transition in $\mathcal{X}^+$. One can show that during the execution of GENEX, at the beginning of each iteration, there is a feature f in $\mathcal{G}$ such that the score of C_f is non-zero. Hence, GENEX cannot terminate with failure. □

Finally, if $\mathcal{Q}$ is a finite collection of instances, then one can construct sets $\mathcal{X}^+$ and $\mathcal{X}^-$ of good and bad transitions over the instances in $\mathcal{Q}$ such that any solution π for BLT$(\mathcal{F}, \mathcal{X}^+, \mathcal{X}^-)$ solves $\mathcal{Q}$.

Theorem 14. *Let $\mathcal{Q}$ be a **finite** class of instances, let $\mathcal{F}$ be a pool of features, and let $\mathcal{X}^+$ and $\mathcal{X}^-$ be set of good and bad transitions that satisfy the following:*

1. *For each instance P in $\mathcal{Q}$, and each alive state s in P, there is a transition (s, s') in $\mathcal{X}^+$, and*
2. *For each instance P in $\mathcal{Q}$, $\mathcal{X}^-$ contains all the transitions (s, s') in P where s is alive and s' is a dead-end state.*

If π is a solution of BLT$(\mathcal{F}, \mathcal{X}^+, \mathcal{X}^-)$, then π solves $\mathcal{Q}$.

Proof. Let P be in $\mathcal{Q}$, and let $\tau = (s_0, s_1, \ldots, s, t)$ be a **maximal** π-trajectory in P. Clearly, τ is acyclic since π is stratified. Since all transitions (s, t) in P from an alive state s to a dead-end t are in $\mathcal{X}^-$, t is not a dead-end state. On the other hand, t cannot be alive as otherwise there would be a transition (t, t') in $\mathcal{X}^+$, implying that τ is not maximal. Therefore, t must be a goal state. □

7 Meta Learning Task: WRAPPER

The basic learning task BLT$(\mathcal{F}, \mathcal{X}^+, \mathcal{X}^-)$ is solved efficiently by GENEX, provided it has solution. There is however an important open question: what state transitions from the training instances to include in $\mathcal{X}^+$ and $\mathcal{X}^-$?

One answer to above question is given by sets $\mathcal{X}^+$ and $\mathcal{X}^-$ that comply with the conditions in Theorem 14. However, this is impractical as computing such sets requires the expansion of the state space for the training instances in $\mathcal{Q}$, and would result in very large sets of transitions, at least one transition per each alive state in each instance. In this section, we describe an efficient **wrapper algorithm**, simply called WRAPPER, that starting from example paths computed by a planner, identifies additional transitions that are added to $\mathcal{X}^+$ and $\mathcal{X}^-$. The algorithm, being greedy, is incomplete, yet it is able to solve a large number of benchmarks.

The idea behind WRAPPER, depicted in Alg. 2, is simple. To find a policy that solves $\mathcal{Q}$, the wrapper works with a small subset $\mathcal{Q}' \subseteq \mathcal{Q}$ (line 2), finds a solution π for $\mathcal{Q}'$ using GENEX (lines 4–7), and tests π on $\mathcal{Q}$ (line 8). If π solves $\mathcal{Q}$, it returns π. Else, $\mathcal{Q}'$ is updated, and the process repeats.

Finding a solution for $\mathcal{Q}'$ may involve multiple calls to GENEX with different sets $\mathcal{X}^+$ and $\mathcal{X}^-$ of good and bad transitions for $\mathcal{Q}'$. If $H(\mathcal{F}, \mathcal{X}^+, \mathcal{X}^-)$ has solution $\mathcal{G}$, the policy $\pi = \pi(\mathcal{G}, \mathcal{X}^+)$ is executed on the instances in $\mathcal{Q}'$. Three different outcomes may arise: 1) π solves $\mathcal{Q}'$; 2) an alive state s is found where π is not defined, hence π is not closed; or 3) a dead-end state t is reached,[1] hence π is not safe. In case 2, $\mathcal{X}^+$ is extended with a transition (s, t) obtained with the planner. In case 3, $\mathcal{X}^-$ is extended with the last π-transition (s, t) leading to the dead-end state. This is repeated until a solution for $\mathcal{Q}'$ is obtained, or GENEX fails, in which case WRAPPER also fails (line 5). This loop (lines 4–7) is called the **inner loop** of the wrapper.

The optional policy simplification in line 6 tries to remove conditions and insert unknown effects (i.e., of type $n?$) in the rules of the projected policy $\pi = \pi(\mathcal{G}, \mathcal{X}^+)$ to increase its coverage. The simplification is an iterative, greedy, process that at each iteration attempts to remove a condition or insert an unknown effect while preserving the *invariant:* π remains stratified under the same ranking κ, π has the same conditional monotonicites, and $\pi \cap \mathcal{X}^- = \emptyset$.

Finally, if the found policy, which solves $\mathcal{Q}'$, does not solve $\mathcal{Q}$, two strategies for updating $\mathcal{Q}'$ are considered. Both strategies work with a *static ordering* $P_1, P_2, P_3, \ldots$ of the instances in $\mathcal{Q}$, determined by the length of the plans computed by the planner Φ, from larger to shorter plans. In the

[1] Dead-end states are identified with the planner: state t is declared as dead-end state iff the planner cannot find a plan for it.

Algorithm 2 WRAPPER for GENEX.

Input: Training set $\mathcal{Q}$ with planning instances P, pool $\mathcal{F}$ of features, and planner Φ.

Output: FAILURE, or stratified policy π that solves $\mathcal{Q}$.

1: Use the planner Φ on each instance P in $\mathcal{Q}$ to obtain plan τ_P for P. Collect the transitions in τ_P into the sets $\mathcal{X}_P^+$ and $\mathcal{X}_P^- = \emptyset$ of good and bad transitions for P.
2: [[Outer loop, lines 2–8]] Get non-empty subset $\mathcal{Q}' \subseteq \mathcal{Q}$ of instances. If no more subsets $\mathcal{Q}'$ are available (see text), return FAILURE.
3: Let $\mathcal{X}^+ \leftarrow \cup\{\mathcal{X}_P^+ \mid P \in \mathcal{Q}'\}$; $\mathcal{X}^- \leftarrow \cup\{\mathcal{X}_P^- \mid P \in \mathcal{Q}'\}$.
4: [[Inner loop, lines 4–7]] Solve $H = H(\mathcal{F}, \mathcal{X}^+, \mathcal{X}^-)$ with GENEX (Algorithm 1).
5: **Return** FAILURE, if GENEX fails, else let $\pi = \pi(\mathcal{G}, \mathcal{X}^+)$ for the solution $\mathcal{G}$ of H.
6: [[Optional]] Simplify policy π (see text).
7: Test π on $\mathcal{Q}'$. If, for some P in $\mathcal{Q}'$, there is a π-trajectory $\langle s_0, \ldots, s, t\rangle$ such that t is identified as a dead-end state, or t is not covered by π, then: in the first case, augment $\mathcal{X}^-$ with transition (s,t) (to be avoided), else, in the second case, augment $\mathcal{X}^+$ with a transition (t,t') "recommended" by the planner Φ (to be covered). **Continue** to Step 4.
8: Test π on $\mathcal{Q}$. If π does not solve P in $\mathcal{Q}$, update the subset $\mathcal{Q}'$ to contain P (and possibly other instances), and **continue** to Step 2. Else, **return** π as π solves $\mathcal{Q}$. (See the text for strategies to update $\mathcal{Q}'$.)

first strategy, named S_1, $\mathcal{Q}'$ is always a singleton, initialized to $\{P_1\}$. If π solves $\mathcal{Q}' = \{P_k\}$ but not P_ℓ, where ℓ is minimum, $\mathcal{Q}'$ is updated to $\{P_\ell\}$ if $\ell > k$, else to $\{P_{k+1}\}$. In the **second strategy,** named S_2, $\mathcal{Q}'$ also starts as $\{P_1\}$. But, if π solves $\mathcal{Q}'$, with max instance index k, but not P_ℓ, of minimum index, $\mathcal{Q}'$ is set to $\mathcal{Q}' \cup \{P_\ell\}$ if $\ell < k$, else to $\{P_\ell\}$. The loop comprising lines 4–8 is referred to as the **outer loop.** The number of iterations of the outer loop is bounded by $|\mathcal{Q}|$ for ther first strategy, and by $|\mathcal{Q}|^2$ for the second strategy.

8 Experiments

We implemented GENEX and WRAPPER in Python with the help of the libraries DLPLAN and TARSKI (Drexler, Francès, and Seipp 2022; Francés, Ramirez, and Collaborators 2018). The planner is SIW (Lipovetzky and Geffner 2012) which is fast and effective, except for `Spanner` where BFWS (Lipovetzky and Geffner 2017) is used because SIW can only solve instances with one nut. The source code, benchmarks, and results are publicly available.[2]

The other approaches for computing general rule-based policies for generalized planning are: the approach of Francès, Bonet, and Geffner (2021) for computing general policies, and the approach of Drexler, Seipp, and Geffner (2022) for computing sketches of bounded width. Both approaches are based on SAT/ASP, and require the full expansion of the state-space for the training instances, and thus they cannot handle large training instances or feature pools. Indeed, the reported experiments for the first approach, only consider 9 domains, with pools of up to 2,000 features, and instances with up to 6,000 non-equivalent states. General policies are obtained by a careful choice of the training instances. The second approach reports experiments on 9 domains, from which policies are obtained for 6 domains. The pools involved have at most a thousand features, and tens of states. The domains solved by either approach are solved by the new approach, but not vice versa: there are domains solved by the new approach that cannot be solved by any of the previous approaches.

We carried out experiments on 34 standard planning domains of various sorts that pose different type of challenges for generalized planning. Some of these domains have been considered before, and others are new (we don't have space to describe the domains). In all cases, we pre-computed a pool of features using DLPLAN with a complexity bound of 15 and max-depth bound of 5, except for Logistics and 8-puzzle domains where the bound was increased to 20. The WRAPPER starts with strategy S_1, switching to strategy S_2 when the first fails, as explained below.

8.1 Results

We divide the results into 5 categories, depending on the number of considered plans, and the addition of good and bad transitions, in order to obtain a policy π that solves $\mathcal{Q}$:

C1. 5 domains that require a single call to GENEX (i.e., only the first plan is considered, and no additional transitions are needed).
C2. 4 domains that require just the first plan, but additional good or bad transitions are needed.
C3. 5 domains that require just one plan, but the first plan considered does not yield a solution.
C4. 6 domains that require the second strategy S_2.
C5. 14 domains where no general policy is found with the reason (only 3 timeouts, and 11 for lack of expressivity, explained below).

Notice that there are 14 domains where general policies are found by generalizing a single plan, cf. categories C1–C3.

Table 1 shows the results for categories C1–C4, 20 domains, where $|\mathcal{Q}|$ is the number of instances in the training set $\mathcal{Q}$, $|\mathcal{S}|$ is the number of states considered (seen) during training, $|\mathcal{F}|$ is the size of the feature pool, 'Strat.' is the wrapper strategy, and 'Outer' and 'Inner' are the total number of iterations for outer and inner loop of WRAPPER, respectively.

The table also contains details for the last outer iteration: $|\mathcal{Q}'|$ is the number of instances from $\mathcal{Q}$ used in the last outer iteration, 'Inner*' is the number of inner iterations for the last outer iteration, $|\mathcal{X}^+|$ and $|\mathcal{X}^-|$ is the last number of good and bad transitions, $|H|$ is the size of the hitting set problem, $|\mathcal{G}|$ is the size of hitting set, and $|\pi|$ is the number of rules in the projected policy $\pi = \pi(\mathcal{G}, \mathcal{X}^+)$.

[2] http://github.com/bonetblai/learner-policies-from-examples.

The last four columns in Table 1 show the accumulated times in seconds spent by preprocessing to support GENEX, GENEX itself, verification, and total wall time. The verification is costly when the number of instances in $\mathcal{Q}$ is large, or the policy visits a large number of states.

The table is vertically divided into four parts corresponding to the categories C1–C4, respectively. The 5 domains in the top are solved fast, in a few seconds, except `Visitall` that requires 18.5min, from which about 16.5min are spent in the verification over the 460 instances in $\mathcal{Q}$. In the second part, there are domains with dead-end states, like `Spanner-1nut`, which are used to augment the set $\mathcal{X}^-$ of bad transitions. Interestingly, just few bad transitions are needed in domains with dead-end state: 1 for both versions of `Spanner`, 2 for `Barman-1cocktail-1shot`, and 4 for `Sokoban-1stone-7x7`. The domains in the third part require just one example to find a general policy, but such an example is not the first one considered as determined by a static ordering of the instances. In `Blocks4ops-on`, for example, the second example suffices without the need to consider additional good or bad transitions (i.e., Inner$^* = 1$). Finally, the domains at the bottom of the table required the strategy S_2 that simultaneously considers transitions from more than one example path. Strategy S_2 is used after S_1 considers all the $\mathcal{Q}'$ subsets (i.e., each singleton $\mathcal{Q}'$ yields a policy that solve $\mathcal{Q}'$ but not $\mathcal{Q}$). Regarding times, for the majority of the cases, a general policy is found in a few minutes (13 domains finished in less than 10 minutes). `Sokoban-1stone-7x7` takes 7 hours and 50 minutes for 309 calls to GENEX in order to learn a policy that is able to solve 8 instances of Sokoban with 1 stone on a 7×7 grid. The policy has 50 rules and 15 features. It is a policy that is highly over fit to the training set.

For the remaining 14 domains, WRAPPER was not able to find a general policy. Table 2 summarizes the results, with columns similar to Table 1, except for a new column entitled 'Reason' that explains the failure of WRAPPER: 'Edge' if there is a transition (s, t) in $\mathcal{X}^+$ for which the pool contains no feature f that changes across (s, t), and 'Timeout' (when the process is killed and no timing data is available). The 'Edge' failure is due to *lack of expressivity in the pool.* We believe that for most cases, it would not be enough to just increase the complexity bounds that are used to generate the pool of features. Rather, it is simply that the features needed to express a general policy fall outside the class of features that are captured by the grammar; i.e., features definable with 2-variable logic and counting quantifiers.

8.2 Further Testing of Policies

The learned policies solve the entire class $\mathcal{Q}$, even though, in many cases, the number of instances seen during learning is a fraction of $\mathcal{Q}$. In this section, we go further and test the policies on instances that are significantly larger than the ones used for training. For example, for `Blocks4ops` the training set contains 5 instances with 10 blocks each, but we evaluate the resulting policy on instances with 20–45 blocks.

Table 3 shows statistics for this extended test. For each depicted domain, the table contains the number of instances, the percentage of coverage (percentage of solved instances), and the maximum and average **effective width**.

The **width of a sketch** π on state s for instance P is the minimum integer k needed for the search algorithm IW(k) to find a state t from s such that the pair (s, t) is compatible with π (Drexler, Seipp, and Geffner 2022). If the sketch π is a policy, such state t is a successor of s, and IW(0) finds it. Else, if π is not a proper policy, it is regarded as a sketch and paired with IW(k) for some $k > 0$; $k = 2$ in this test. The max (resp. average) **effective width** for instance P is the max (resp. average) width of π for the states encountered when using π, and the max (resp. average) effective width for the class $\mathcal{Q}$ is the maximum of the max (resp. average) effective width of π over instances P in $\mathcal{Q}$.

Values for coverage and effective width that deviate from 100% and 0, respectively, are highlighted in Table 3, as they exhibit flaws of the policy π when used on large instances. Nonetheless, in all cases, except `Childsnack` with a coverage of 41.7%, the learned policy solves the large instances in the test set. Likewise, except for `8puzzle`, `Blocks4ops`, and `Childsnack`, an effective width of 0 tells us that the learned policy is indeed a policy that selects transitions that lead to goal states. The case for the two version of the 8-puzzle is interesting. The policies are learned using only instances for the 8-puzzle, but the test set includes instances for the $(n^2 - 1)$-puzzle, $n = 2, 3, \ldots, 6$. The effective width is bigger than 0 **only** on some instances for n in $\{4, 5, 6\}$.

9 Conclusions

We have introduced a novel formulation and algorithms for learning generalized policies from examples computed by a planner. In many cases, we have shown that the method yields general policies by generalizing a single plan. In other cases, a few plans need to be considered.

Two key contributions in relation to existent methods are that the new method scales up to much larger pools of features and training instances, and this enables the solution of domains that could not be addressed before. At the same time, this is the first method in which the resulting general policies are *acyclic by design.* This is achieved through the introduction of a new powerful structural termination criterion that is built-in in the selection of the features.

The new learning algorithm is made up of a core algorithm, GENEX, that is implemented by a fast and efficient hitting-set algorithm, and the WRAPPER around it, that feeds GENEX with transitions in $\mathcal{X}^+$ to be included in the policy, and others in $\mathcal{X}^-$ to be excluded. A shortcoming of WRAPPER is that it is not complete: there could be positive and negative transitions that yield a general policy over the target class of problems, but the wrapper may fail to find them. An improved, complete and efficient, wrapper around the complete and efficient GENEX is left as a challenge for future work.

							Last (outer) iteration for WRAPPER							Time in seconds			
Domain	$\|\mathcal{Q}\|$	$\|\mathcal{S}\|$	$\|\mathcal{F}\|$	Strat.	Outer	Inner	$\|\mathcal{Q}'\|$	Inner*	$\|\mathcal{X}^+\|$	$\|\mathcal{X}^-\|$	$\|H\|$	$\|\mathcal{G}\|$	$\|\pi\|$	Prep.	GENEX	Verif.	Total
Blocks4ops-clear	53	172	13,579	S_1	1	1	1	1	7	0	14	2	2	0.63	0.01	0.18	10.40
Delivery-1pkg	169	981	5,765	S_1	1	1	1	1	9	0	18	4	4	0.27	0.02	3.21	24.61
Gripper	5	60	6,154	S_1	1	1	1	1	19	0	38	3	4	0.46	0.21	3.95	8.22
Reward	207	1,119	249,122	S_1	1	1	1	1	17	0	34	2	2	26.16	33.73	6.97	152.51
Visitall	460	4,288	146,085	S_1	1	1	1	1	20	0	40	2	3	15.68	1.41	991.55	1,110.15
Childsnack	10	85	2,900	S_1	1	2	1	2	15	0	30	6	8	0.50	0.08	32.17	35.64
Spanner-1nut	90	540	8,001	S_1	1	2	1	2	6	1	19	3	3	0.52	0.08	0.58	11.75
Logistics-1truck	67	363	262,509	S_1	1	17	1	17	32	0	64	5	8	905.52	803.52	2.71	1,926.61
Barman-1cocktail-1shot	90	1,350	69,040	S_1	1	21	1	21	32	2	133	11	22	243.14	539.91	216.19	1,049.92
Blocks4ops-on	94	518	183,322	S_1	2	2	1	1	10	0	20	4	7	24.24	1.90	1.14	108.27
Spanner	270	2,160	13,679	S_1	2	3	1	2	10	1	31	3	3	1.96	0.39	176.08	210.42
Delivery	397	3,635	13,904	S_1	4	21	1	3	23	0	46	4	5	42.18	6.69	23.03	135.30
Ferry	180	1,416	8,547	S_1	5	13	1	6	17	0	34	4	5	8.69	3.95	3.66	39.36
Miconic	360	3,504	107,785	S_1	9	30	1	6	20	0	40	4	5	253.81	73.78	50.87	502.87
8puzzle-1tile-fixed	18	140	11,230	S_2	7	38	4	3	40	0	80	8	14	44.77	25.52	1.89	81.54
8puzzle-1tile	16	122	12,417	S_2	8	52	3	6	40	0	80	8	18	69.14	35.62	2.20	117.80
Blocks4ops	5	129	100,897	S_2	10	66	3	5	81	0	162	7	24	5,349.49	7,159.42	19.62	12,863.98
Sokoban-1stone-7x7	8	67	115,214	S_2	12	309	5	15	94	4	671	15	50	14,248.40	12,712.36	896.13	28,196.51
Logistics-1pkg	24	173	225,518	S_2	15	138	4	8	56	0	112	7	16	6,936.58	5,913.87	416.71	16,136.66
Zenotravel-1plane	73	779	24,959	S_2	28	266	7	3	122	0	244	7	18	2,039.20	277.37	2,905.90	5,406.45

Table 1: Results for the 20 domains in categories C1–C4 for which WRAPPER yields a general policy. As it can be seen, WRAPPER can handle hundreds of thousands of features, and instances with millions of reachable states (e.g., all instances in Blocks4ops have 10 blocks).

							Last (outer) iteration for WRAPPER						Time in seconds			
Domain	$\|\mathcal{Q}\|$	$\|\mathcal{S}\|$	$\|\mathcal{F}\|$	Strat.	Outer	Inner	$\|\mathcal{Q}'\|$	Inner*	$\|\mathcal{X}^+\|$	$\|\mathcal{X}^-\|$	$\|H\|$	Reason	Prep.	GENEX	Verif.	Total
Rovers	608	6,238	190,064	S_1	1	1	1	1	25	0	50	Edge	17.26	0.53	0.00	225.75
Tidybot-opt11-strips	8	211	59,402	S_1	1	1	1	1	40	0	80	Edge	7.70	0.21	0.00	119.06
Tpp	11	608	14,128	S_1	1	1	1	1	201	0	402	Edge	9.32	0.21	0.00	237.19
8puzzle-2tiles	16	207	4,458	S_1	1	2	1	2	20	0	40	Edge	0.85	0.14	0.18	6.30
Hiking	180	1,215	16,723	S_1	1	2	1	2	8	0	16	Edge	1.41	0.06	4.02	190.05
Depot	18	851	255,079	S_1	1	17	1	17	119	0	238	Edge	4,771.50	379.59	415.29	15,511.73
Freecell	65	2,842	146,428	S_1	1	35	1	35	133	19	2,964	Timeout	-1.00	-1.00	-1.00	-1.00
Barman-1cocktail	270	3,998	69,040	S_1	1	94	1	94	105	6	868	Edge	3,056.59	5,474.44	36.43	8,769.37
Tetris-opt14-strips	16	393	37,496	S_1	1	142	1	142	180	1	550	Edge	5,109.83	3,458.80	7,157.74	16,500.19
Satellite	950	6,716	171,475	S_1	2	209	1	157	168	0	334	Timeout	-1.00	-1.00	-1.00	-1.00
Driverlog	381	3,197	172,818	S_1	6	152	1	16	31	0	62	Edge	5,768.13	1,822.26	52.39	8,088.94
Zenotravel-1person	80	500	42,271	S_1	12	385	1	77	80	3	507	Edge	2,262.94	350.48	1,359.22	4,701.44
Logistics	282	1,855	51,418	S_2	11	72	4	5	99	0	198	Edge	987.57	169.34	15.41	1,310.91
Blocks3ops	392	2,216	236,954	S_2	17	102	7	8	106	0	206	Timeout	-1.00	-1.00	-1.00	-1.00

Table 2: Results for the 14 domains in category C5 for which WRAPPER is unable to find a general policy. The column 'Reason' explains the failure: 'Edge' means that there is good transition (s, t) for which no feature in the pool changes across (s, t), and 'Timeout' means the solver did not finish after 12 hours.

			Effective width	
Domain	#inst.	%Coverage	max.	avg.
8puzzle-1tile-fixed	100	100.0%	2.00	0.50
8puzzle-1tile	100	100.0%	2.00	0.32
Blocks4ops-clear	30	100.0%	0.00	0.00
Blocks4ops-on	30	100.0%	0.00	0.00
Blocks4ops	30	100.0%	2.00	0.05
Childsnack	120	41.7%	1.00	0.20
Delivery	30	100.0%	0.00	0.00
Ferry	30	100.0%	0.00	0.00
Gripper	30	100.0%	0.00	0.00
Logistics-1pkg	420	100.0%	0.00	0.00
Logistics-1truck	55	100.0%	0.00	0.00
Miconic	31	100.0%	0.00	0.00
Reward	30	100.0%	0.00	0.00
Zenotravel-1plane	180	100.0%	0.00	0.00

Table 3: Coverage and effective width of some learned policies on large instances. Values that reveal flaws of the policies when applied on large instances are highlighted. The few number of highlighted cells shows that the learned policies are robust on instances that are significantly larger than the ones in the training sets.

Acknowledgments

The research of H. Geffner has been supported by the Alexander von Humboldt Foundation with funds from the Federal Ministry for Education and research, by the European Research Council (ERC), Grant agreement No. 885107, and by the Excellence Strategy of the Federal Government and the NRW Länder, Germany.

References

Barceló, P.; Kostylev, E. V.; Monet, M.; Pérez, J.; Reutter, J.; and Silva, J. P. 2020. The logical expressiveness of graph neural networks. In *Proc. ICLR*.

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

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

Bonet, B., and Geffner, H. 2020. Qualitative numeric plan-

ning: Reductions and complexity. *Journal of Artificial Intelligence Research (JAIR)* 69:923–961.

Bonet, B., and Geffner, H. 2024. General policies, subgoal structure, and planning width. *Journal of Artificial Intelligence Research (JAIR)* 80:475–516.

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

Boutilier, C.; Reiter, R.; and Price, B. 2001. Symbolic dynamic programming for first-order MDPs. In *Proc. IJCAI*, 690–700.

Bueno, T. P.; de Barros, L. N.; Mauá, D. D.; and Sanner, S. 2019. Deep reactive policies for planning in stochastic nonlinear domains. In *Proc. AAAI*, 7530–7537.

Celorrio, S. J.; Segovia-Aguas, J.; and Jonsson, A. 2019. A review of generalized planning. *Knowl. Eng. Rev.* 34.

Drexler, D.; Francès, G.; and Seipp, J. 2022. DLPlan. 10.5281/zenodo.5826139.

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

Fern, A.; Yoon, S.; and 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.

Francès, G.; Bonet, B.; and Geffner, H. 2021. Learning general planning policies from small examples without supervision. In *Proc. AAAI*, 11801–11808.

Francés, G.; Ramirez, M.; and Collaborators. 2018. Tarski: An AI planning modeling framework. https://github.com/aig-upf/tarski.

Grohe, M. 2021. The logic of graph neural networks. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 1–17.

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

Ho, J., and Ermon, S. 2016. Generative adversarial imitation learning. In *Proc. NIPS*.

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

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

Karia, R.; Nayyar, R. K.; and 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* 113:125–148.

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

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

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

Ng, A., and Russell, S. 2000. Algorithms for inverse reinforcement learning. In *Proc. ICML*, 663–670.

Rivlin, O.; Hazan, T.; and Karpas, E. 2020. Generalized planning with deep reinforcement learning. In *Proc. ICAPS Workshop on Planning and Reinforcement Learning*.

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

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

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

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

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022. Learning generalized policies without supervision using GNNs. In *Proc. KR*, 474–483.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2023. Learning general policies with policy gradient methods. In *Proc. KR*, 647–657.

Toyer, S.; Trevizan, F.; Thiébaux, S.; and Xie, L. 2018. Action schema networks: Generalised policies with deep learning. In *Proc. AAAI*, 6294–6301.

Toyer, S.; Thiébaux, S.; Trevizan, F.; and Xie, L. 2020. Asnets: Deep learning for generalised planning. *Journal of Artificial Intelligence Research (JAIR)* 68:1–68.

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