Learning General Policies from Small Examples Without Supervision

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

Abstract

Generalized planning is concerned with the computation of general policies that solve multiple instances of a planning domain all at once. It has been recently shown that these policies can be computed in two steps: first, a suitable abstraction in the form of a qualitative numerical planning problem (QNP) is learned from sample plans, then the general policies are obtained from the learned QNP using a planner. In this work, we introduce an alternative approach for computing more expressive general policies which does not require sample plans or a QNP planner. The new formulation is very simple and can be cast in terms that are more standard in machine learning: a large but finite pool of features is defined from the predicates in the planning examples using a general grammar, and a small subset of features is sought for separating "good" from "bad" state transitions, and goals from non-goals. The problems of finding such a "separating surface" while labeling the transitions as "good" or "bad" are jointly addressed as a single combinatorial optimization problem expressed as a Weighted Max-SAT problem. The advantage of looking for the simplest policy in the given feature space that solves the given examples, possibly non-optimally, is that many domains have no general, compact policies that are optimal. The approach yields general policies for a number of benchmark domains.

Introduction

Generalized planning is concerned with the computation of general policies or plans that solve multiple instances of a given 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). For example, a general plan for clearing a block x in **any** instance of Blocksworld involves a loop where the topmost block above x is picked up and placed on the table until no such block remains. A general plan for solving any Blocksworld instance is also possible, like one where misplaced blocks and those above them are moved to the table, and then to their targets in order. The key question in generalized planning is how to represent and compute such general plans from the domain representation.

In one of the most general formulations, general policies are obtained from an abstract planning model expressed as a qualitative numerical planning problem or QNP (Srivastava et al. 2011). A QNP is a standard STRIPS planning model extended with non-negative numerical variables that can be decreased or increased "qualitatively"; i.e., by uncertain positive amounts, short of making the variables negative. Unlike standard planning with numerical variables (Helmert 2002), QNP planning is decidable, and QNPs can be compiled in polynomial time into fully observable non-deterministic (FOND) problems (Bonet and Geffner 2020)

The main advantage of the formulation of generalized planning based on QNPs is that it applies to standard relational domains where the pool of (ground) actions change from instance to instance. On the other hand, while the planning domain is assumed to be given, the QNP abstraction is not, and hence it has to be written by hand or learned. This is the approach of Bonet, Francès, and Geffner (2019) where generalized plans are obtained by learning the QNP abstraction from the domain representation and sample plans, and then solving the abstraction with a QNP planner.

In this work, we build on this thread but introduce an alternative approach for computing general policies that is simpler, yet more powerful. The learning problem is cast as a **self-supervised classification problem** where (1) a pool of features is automatically generated from a general grammar applied to the domain predicates, and (2) a small subset of features is sought for separating "good" from "bad" state transitions, and goals from non-goals. The problems of finding the "separating surface" while labeling the transitions as "good" or "bad" are addressed jointly as a single combinatorial optimization task solved with a Weighted Max-SAT solver. The approach yields general policies for a number of benchmark domains.

The paper is organized as follows. We first review related work and classical planning, and introduce a new language for expressing general policies motivated by the work on QNPs. We then present the learning task, the computational approach for solving it, and the experimental results.

Related Work

The computation of general plans from domain encodings and sample plans has been addressed in a number of works (Khardon 1999; Martín and Geffner 2004; Fern, Yoon, and Givan 2006; Silver et al. 2020). Generalized planning has also been formulated as a problem in first-order logic (Srivastava, Immerman, and Zilberstein 2011; Illanes and McIlraith 2019), and general plans over finite horizons have been derived using first-order regression (Boutilier, Reiter, and Price 2001; Wang, Joshi, and Khardon 2008; van Otterlo, M. 2012; Sanner and Boutilier 2009). More recently, general policies for planning have been learned from PDDL domains and sample plans using deep learning (Toyer et al. 2018; Bueno et al. 2019; Garg, Bajpai, and Mausam 2020). Deep reinforcement learning methods (Mnih et al. 2015) have also been used to generate general policies from images without assuming prior symbolic knowledge (Groshev et al. 2018; Chevalier-Boisvert et al. 2019), in certain cases accounting for objects and relations through the use of suitable architectures (Garnelo and Shanahan 2019). Our work is closest to the works of Bonet, Francès, and Geffner (2019) and Francès et al. (2019). The first provides a model-based approach to generalized planning where an abstract QNP model is learned from the domain representation and sample instances and plans, which is then solved by a QNP planner (Bonet and Geffner 2020). The second learns a generalized value function in an unsupervised manner, under the assumption that this function is linear. Model-based approaches have an advantage over inductive approaches that learn generalized plans; like logical approaches, they guarantee that the resulting policies (conclusions) are correct provided that the model (set of premises) is correct. The approach developed in this work does not make use of QNPs or planners but inherits these formal properties.

Planning

A (classical) planning instance is a pair $P = \langle D, I\rangle$ where D is a first-order planning **domain** and I is an **instance**. The domain D contains a set of predicate symbols and a set of action schemas with preconditions and effects given by atoms $p(x_1, \ldots, x_k)$, where p is a k-ary predicate symbol, and each x_i is a variable representing one of the arguments of the action schema. The instance is a tuple $I = \langle O, Init, Goal\rangle$, where O is a (finite) set of object names c_i, and $Init$ and $Goal$ are sets of ground atoms $p(c_1, \ldots, c_k)$, where p is a k-ary predicate symbol. This is indeed the structure of planning problems as expressed in PDDL (Haslum et al. 2019).

The states associated with a problem P are the possible sets of ground atoms, and the state graph $G(P)$ associated with P has the states of P as nodes, an initial state s_0 that corresponds to the set of atoms in $Init$, and a set of goal states s_G with all states that include the atoms in $Goal$. In addition, the graph has a directed edge (s, s') for each state transition that is possible in P, i.e. where there is a ground action a whose preconditions hold in s and whose effects transform s into s'. A state trajectory $s_0, \ldots, s_n$ is possible in P if every transition (s_i, s_{i+1}) is possible in P, and it is goal-reaching if s_n is a goal state. An action sequence $a_0, \ldots, a_{n-1}$ that gives rise to a goal-reaching trajectory, i.e., where transition (s_i, s_{i+1}) is enabled by ground action a_i, is called a plan or solution for P.

Generalized Planning

A key question in generalized planning is how to represent general plans or policies when the different instances to be solved have different sets of objects and ground actions. One solution is to work with general features (functions) that have well defined values over any state of any possible domain instance, and think of general policies π as mappings from feature valuations into *abstract actions* that denote changes in the feature values (Bonet and Geffner 2018). In this work, we build on this intuition but avoid the introduction of abstract actions (Bonet and Geffner 2021).

Policy Language and Semantics

The **features** considered are boolean and numerical. The first are denoted by letters like p, and their (true or false) value in a state s is denoted as $p(s)$. Numerical features n take non-negative integer values, and their value in a state is denoted as $n(s)$. The complete set of features is denoted as Φ and a joint valuation over all the features in Φ in a state s is denoted as $\phi(s)$, while an arbitrary valuation as ϕ. The expression $[\![\phi]\!]$ denotes the boolean counterpart of ϕ; i.e., $[\![\phi]\!]$ gives a truth value to all the atoms $p(s)$ and $n(s) = 0$ for features p and n in Φ, without providing the exact value of the numerical features n if $n(s) \neq 0$. The number of possible **boolean feature valuations** $[\![\phi]\!]$ is equal to $2^{|\Phi|}$, which is a fixed number, as the set of features Φ does not change across instances.

The possible **effects** E on the features in Φ are p and $\neg p$ for boolean features p in E, and $n\downarrow$ and $n\uparrow$ for numerical features n in E. If $\Phi = \{p, q, n, m, r\}$ and $E = \{p, \neg q, n\uparrow, m\downarrow\}$, the meaning of the effects in E is that p must become true, q must become false, n must increase its value, and m must decrease it. The features in Φ that are not mentioned in E, like r, keep their values. A set of effects E can be thought of as a set of constraints on possible state transitions:

Definition 1. *Let Φ be a set of features over a domain D, let (s, s') be a state transition over an instance P of D, and let E be a set of effects over the features in Φ. Then the transition (s, s') is **compatible with** or **satisfies** E when 1) if p ($\neg p$) in E, then $p(s') = true$ (resp. $p(s') = false$), 2) if $n\downarrow$ ($n\uparrow$) in E, then $n(s) > n(s')$ (resp. $n(s) < n(s')$), and 3) if p and n are not mentioned in E, then $p(s) = p(s')$, and $n(s) = n(s')$ respectively.*

The form of the general policies considered in this work can then be defined as follows:

Definition 2. *A **general policy** π_Φ is given by a set of **rules** $C \mapsto E$ where C is a set (conjunction) of p and n literals for p and n in Φ, and E is an effect expression.*

The p and n-literals are p, $\neg p$, $n{=}0$, and $\neg(n{=}0)$, abbreviated as $n{>}0$. For a reachable state s, the policy π_Φ is a filter on the state transitions (s, s') in P:

Definition 3. *A general policy π_Φ **denotes** a mapping from state transitions (s, s') over instances $P \in \mathcal{Q}$ into boolean values. A transition (s, s') is **compatible** with π_Φ if for some policy rule $C \mapsto E$, C is true in $\phi(s)$ and (s, s') satisfies E_i.*

As an illustration of these definitions, we consider a policy for achieving the goals $clear(x)$ and an empty gripper in any Blocksworld instance with a block x.

Example. Consider the policy π_Φ given by the following two rules for features $\Phi{=}\{H, n\}$, where H is true if a block is being held, and n tracks the number of blocks above x:

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

The first rule says that when the gripper is empty and there are blocks above x, then any action that decreases n and makes H true should be selected. The second one says that when the gripper is not empty and there are blocks above x, any action that makes H false and does not affect the count n should be selected (this rules out placing the block being held above x, as this would increase n). ☐

The conditions under which a general policy solves a class of problems are the following:

Definition 4. *A state trajectory $s_0, \ldots, s_n$ is* ***compatible*** *with policy π_Φ in an instance P if s_0 is the initial state of P and each pair (s_i, s_{i+1}) is a possible state transition in P compatible with π_Φ. The trajectory is* ***maximal*** *if s_n is a goal state, there are no state transitions (s_n, s) in P compatible with π_Φ, or the trajectory is infinite and does not include a goal state.*

Definition 5. *A general policy π_Φ* ***solves*** *a class $\mathcal{Q}$ of instances over domain D if in each instance $P \in \mathcal{Q}$, all maximal state trajectories compatible with π_Φ reach a goal state.*

The policy expressed by the rules in (1) can be shown to solve the class $\mathcal{Q}_{clear}$ of all Blocksworld instances.

Non-deterministic Policy Rules

The general policies π_Φ introduced above determine the actions a to be taken in a state s *indirectly*, as the actions a that result in state transitions (s, s') that are compatible with a policy rule $C \mapsto E$. If there is a single rule body C that is true in s, for the transition (s, s') to be compatible with π_Φ, (s, s') must satisfy the effect E. Yet, it is possible that the bodies C_i of many rules $C_i \mapsto E_i$ are true in s, and then for (s, s') to be compatible with π_Φ it suffices if (s, s') satisfies one of the effects E_i.

For convenience, we abbreviate sets of rules $C_i \mapsto E_i$, $i = 1, \ldots, m$, that have the same body $C_i = C$, as $C \mapsto E_1 \mid \cdots \mid E_m$, and refer to the latter as a **non-deterministic rule**. The non-determinism is on the effects on the features: one effect E_i may increment a feature n, and another effect E_j may decrease it, or leave it unchanged (if n is not mentioned in E_j). Policies π_Φ where all pairs of rules $C \mapsto E$ and $C' \mapsto E'$ have bodies C and C' that are jointly inconsistent are said to be **deterministic**. Previous formulations that cast general policies as mappings from feature conditions into abstract (QNP) actions yield policies that are deterministic in this way (Bonet and Geffner 2018; Bonet, Francès, and Geffner 2019). Non-deterministic policies, however, are strictly more expressive.

Example. Consider a domain **Delivery** where a truck has to pick up m packages spread on a grid, while taking them, one by one, to a single target cell t. If we consider the collection of instances with one package only, call them **Delivery-1**, a general policy π_Φ for them can be expressed using the set of features $\Phi = \{n_p, n_t, C, D\}$, where n_p represents the distance from the agent to the package (0 when in the same cell or when holding the package), n_t represents the distance from the agent to the target cell, and C and D represent that the package is carried and delivered respectively. One may be tempted to write the policy π_Φ by means of the four deterministic rules:

$$r_1 : \{\neg C, n_p{>}0\} \mapsto \{n_p\downarrow\} \; ; \; r_2 : \{\neg C, n_p{=}0\} \mapsto \{C\}$$
$$r_3 : \{C, n_t{>}0\} \mapsto \{n_t\downarrow\} \; ; \; r_4 : \{C, n_t{=}0\} \mapsto \{\neg C, D\} \, .$$

The rules say "if away from the package, get closer", "if don't have the package but in the same cell, pick it up", "if carrying the package and away from target, get closer to target", and "if carrying the package in target cell, drop the package". This policy, however, does not solve **Delivery-1**. The reason is that transitions (s, s') where the agent gets closer to the package satisfy the conditions $\neg C$ and $n_p > 0$ of rule r_1 but may fail to satisfy its head $\{n_p\downarrow\}$. This is because the actions that decrease the distance n_p to the package may affect the distance n_t of the agent to the target, contradicting r_1, which says that n_t does not change. To solve **Delivery-1** with the same features, rule r_1 must be changed to the non-deterministic rule:

$$r_1' : \{\neg C, n_p > 0\} \;\mapsto\; \{n_p\downarrow, n_t\downarrow\} \mid \{n_p\downarrow, n_t\uparrow\} \mid \{n_p\downarrow\},$$

which says indeed that "when away from the package, move closer to the package for any possible effect on the distance n_t to the target, which may decrease, increase, or stay the same." We often abbreviate rules like r_1' as $\{\neg C, n_p > 0\} \mapsto \{n_p\downarrow, n_t?\}$, where $n_t?$ expresses "any effect on n_t." ☐

Learning General Policies: Formulation

We turn now to the key challenge: **learning** the features Φ and general policies π_Φ from **samples** $P_1, \ldots, P_k$ of a target class of problems $\mathcal{Q}$, given the domain D. The learning task is formulated as follows. From the predicates used in D and a fixed grammar, we generate a **large pool $\mathcal{F}$ of boolean and numerical features** f, like in (Bonet, Francès, and Geffner 2019), each of which is associated with a measure $w(f)$ of syntactic complexity. We then *search for the simplest set of features $\Phi \subseteq \mathcal{F}$ such that a policy π_Φ defined on Φ solves all sample instances $P_1, \ldots, P_k$*. This task is formulated as a Weighted Max-SAT problem over a suitable propositional theory T, with score $\sum_{f\in\Phi} w(f)$ to minimize.

This learning scheme is **unsupervised** as the sample instances do not come with their plans. Since the sample instances are assumed to be sufficiently small (small state spaces) this is not a crucial issue, and by letting the learning algorithm choose which plans to generalize, the resulting approach becomes more flexible. In particular, if we ask for the policy π_Φ to generalize given plans as in (Bonet, Francès, and Geffner 2019), it may well happen that there are policies in the feature space but none of which generalizes the plans provided by the teacher.

We next describe the propositional theory T assuming that the feature pool $\mathcal{F}$ and the feature weights $w(f)$ are given, and then explain how they are generated. Our SAT formulation is different from (Bonet, Francès, and Geffner 2019) as it is aimed at capturing a more expressive class of policies without requiring QNP planners.

Learning the General Policy as Weighted Max-SAT

The propositional theory $T = T(\mathcal{S}, \mathcal{F})$ that captures our learning task takes as inputs the pool of features $\mathcal{F}$ and the state space $\mathcal{S}$ made up of the (reachable) states s, the possible state transitions (s, s'), and the sets of (reachable) goal states in each of the sample problem instances $P_1, \ldots, P_n$. The handling of dead-end states is explained below. States arising from the different instances are assumed to be different even if they express the same set of ground atoms. The propositional variables in T are

- $Select(f)$: feature f from pool $\mathcal{F}$ makes it into Φ,
- $Good(s, s')$: transition (s, s') is compatible with π_Φ,
- $V(s, d)$: num. labels $V(s) = d$, $V^*(s) \leq d \leq \delta V^*(s)$.

The true atoms $Select(f)$ in the satisfying assignment define the features $f \in \Phi$, while the true atoms $Good(s, s')$, along with the selected features, define the policy π_Φ. More precisely, there is a rule $C \mapsto E_1 \mid \cdots \mid E_m$ in the policy iff for each effect E_i, there is a true atom $Good(s, s_i)$ for which $C = [\![\phi(s)]\!]$, and E_i captures the way in which the selected features change across the transition (s, s_i). The formulas in the theory use numerical labels $V(s) = d$, for $V^*(s) \leq d \leq \delta V^*(s)$ where $V^*(s)$ is the minimum distance from s to a goal, and $\delta \geq 1$ is a *slack parameter* that controls the degree of suboptimality that we allow. All experiments in this paper use $\delta = 2$. These values are used to ensure that the policy determined by the $Good(s, s')$ atoms solves all instances P_i as well as all instances $P_i[s]$ that are like P_i but with s as the initial state, where s is a state reachable in P_i and is not a dead-end. We call the $P_i[s]$ problems **variants** of P_i. Dead-ends are states from which the goal cannot be reached, and they are labeled as such in $\mathcal{S}$.

The formulas are the following. States s and t, and transitions (s, s') and (t, t') range over those in $\mathcal{S}$, excluding transitions where the first state of the transition is a dead-end or a goal. $\Delta_f(s, s')$ expresses how feature f changes across transition (s, s'): for boolean features, $\Delta_f(s, s') \in \{\uparrow, \downarrow, \bot\}$, meaning that f changes from false to true, from true to false, or stays the same. For numerical features, $\Delta_f(s, s') \in \{\uparrow, \downarrow \bot\}$, meaning that f can increase, decrease, or stay the same. The formulas in $T = T(\mathcal{S}, \mathcal{F})$ are:

1. Policy: $\bigvee_{(s,s')} Good(s, s')$, s is non-goal state,
2. V_1.: Exactly-1 $\{V(s, d) : V^*(s) \leq d \leq \delta V^*(s)\}$,[1]
3. V_2: $Good(s, s') \rightarrow V(s, d) \wedge V(s', d')$, $d' < d$,
4. Goal: $\bigvee_{f : [\![f(s)]\!] \neq [\![f(s')]\!]} Select(f)$, one $\{s, s'\}$ is goal,
5. Bad trans: $\neg Good(s, s')$ for s solvable, and s' dead-end,
6. D2-sep: $Good(s, s') \wedge \neg Good(t, t') \rightarrow D2(s, s'; t, t')$, where $D2(s, s'; t, t')$ is $\bigvee_{\Delta_f(s,s') \neq \Delta_f(t,t')} Select(f)$.

[1]This implies that $V(s, 0)$ iff s is a goal state.

The first formula asks for a good transition from any non-goal state s. The good transitions are transitions that will be compatible with the policy. The second and third formulas ensure that these good transitions lead to a goal state, and furthermore, that they can capture any non-deterministic policy that does so. The fourth formulation is about separating goal from non-goal states, and the fifth is about excluding transitions into dead-ends. Finally, the D2-separation formula says that if (s, s') is a "good" transition (i.e., compatible with the resulting policy π_Φ), then any other transition (t, t') in $\mathcal{S}$ where the selected features change exactly as in (s, s') must be "good" as well. $\Delta_f(s, s')$ above captures how feature f changes across the transition (s, s'), and the selected features f change in the same way in (s, s') and (t, t') when $\Delta_f(s, s') = \Delta_f(t, t')$.

The propositional encoding is **sound** and **complete** in the following sense:

Theorem 6. *Let $\mathcal{S}$ be the state space associated with a set $P_1, \ldots, P_k$ of sample instances of a class of problems $\mathcal{Q}$ over a domain D, and let $\mathcal{F}$ be a pool of features. The theory $T(\mathcal{S}, \mathcal{F})$ is **satisfiable** iff there is a general policy π_Φ over features $\Phi \subseteq \mathcal{F}$ that discriminates goals from non-goals and solves $P_1, \ldots, P_k$ and their variants.*

For the purpose of generalization outside of the sample instances, instead of looking for **any** satisfying assignment of the theory $T(\mathcal{S}, \mathcal{F})$, we look for the satisfying assignments that **minimize** the complexity of the resulting policy, as measured by the sum of the costs $w(f)$ of the clauses $Select(f)$ that are true, where $w(f)$ is the complexity of feature $f \in \mathcal{F}$.

We sketched above how a general policy π_Φ is extracted from a satisfying assignment. The only thing missing is the precise meaning of the line "E_i captures the way in which the selected features change in the transition from s to s_i". For this, we look at the value of the expression $\Delta_f(s, s_i)$ computed at preprocessing, and place f ($\neg f$) in E_i if f is boolean and $\Delta_f(s, s_i)$ is '$\uparrow$' (resp. $\downarrow$), and place $f\uparrow$ ($f\downarrow$) in E_i if f is numerical and $\Delta_f(s, s_i)$ is '$\uparrow$' (resp. $\downarrow$). Duplicate effects E_i and E_j in a policy rule are merged. The resulting policy delivers the properties of Theorem 6:

Theorem 7. *The policy π_Φ and features Φ that are determined by a satisfying assignment of the theory T solves the sample problems $P_1, \ldots, P_k$ and their variants.*

Feature Pool

The feature pool $\mathcal{F}$ used in the theory $T(\mathcal{S}, \mathcal{F})$ is obtained following the method described by Bonet, Francès, and Geffner (2019), where the (primitive) domain predicates are combined through a standard description logics grammar (Baader et al. 2003) in order to build a larger set of (unary) concepts c and (binary) roles r. Concepts represent *properties* that the objects of any problem instance can fulfill in a state, such as the property of being a package that is in a truck on its target location in a standard logistics problem. For primitive predicates p mentioned in the goal, a "goal predicate" p_G is added that is evaluated not in the state but in the goal, following (Martín and Geffner 2004).

From these concepts and roles, we generate *cardinality features* $|c|$, which evaluate to the number of objects that satisfy concept c in a given state, and *distance features* $Distance(c_1, r, c_2)$, which evaluate to the minimum number of r-steps between two objects that (respectively) satisfy c_1 and c_2. We refer the reader to the appendix for more detail (Francès, Bonet, and Geffner 2021a). Both types of features are lower-bounded by 0 and upper-bounded by the total number of objects in the problem instance. Cardinality features that only take values in $\{0, 1\}$ are made into boolean features. The complexity $w(f)$ of feature f is given by the size of its syntax tree. The feature pool $\mathcal{F}$ used in the experiments below contains all features up to a certain complexity bound $k_{\mathcal{F}}$.

Experimental Results

We implemented the proposed approach in a C++/Python system called D2L and evaluated it on several problems. Source code and benchmarks are available online[2] and archived in Zenodo (Francès, Bonet, and Geffner 2021b). Our implementation uses the Open-WBO Weighted Max-SAT solver (Martins, Manquinho, and Lynce 2014). All experiments were run on an Intel i7-8700 CPU@3.2GHz with a 16 GB memory limit.

The domains include all problems with simple goals from (Bonet, Francès, and Geffner 2019), e.g. clearing a block or stacking two blocks in Blocksworld, plus standard PDDL domains such as Gripper, Spanner, Miconic, Visitall and Blocksworld. In all the experiments, we use $\delta = 2$ and $k_{\mathcal{F}} = 8$, except in Delivery, where $k_{\mathcal{F}} = 9$ is required to find a policy. We next describe two important optimizations.

Exploiting indistinguishability of constraints. A fixed feature pool $\mathcal{F}$ induces an equivalence relation over the set of all transitions in the training sample that puts two transitions in the same equivalence class iff they cannot be distinguished by $\mathcal{F}$. The theory $T(\mathcal{S}, \mathcal{F})$ above can be simplified by arbitrarily choosing one transition (s, s') for each of these equivalence classes, then using a single SAT variable $Good(s, s')$ to denote the goodness of any transition in the class and to enforce the D2-separation clauses.

Incremental constraint generation. Since the number of D2-separation constraints in the theory $T(\mathcal{S}, \mathcal{F})$ grows quadratically with the number of equivalence classes among the transitions, we use a *constraint generation loop* where these constraints are enforced incrementally. We start with a set τ_0 of pairs of transitions (s, s') and (t, t') that contains all pairs for which $s = t$ plus some random pairs from $\mathcal{S}$. We obtain the theory $T_0(\mathcal{S}, \mathcal{F})$ that is like $T(\mathcal{S}, \mathcal{F})$ but where the D2-separation constraints are restricted to pairs in τ_0. At each step, we solve $T_i(\mathcal{S}, \mathcal{F})$ and validate the solution to check whether it distinguishes all good from bad transitions in the entire sample; if it does not, the offending transitions are added to $\tau_{i+1} \supset \tau_i$, and the loop continues until the solution to $T_i(\mathcal{S}, \mathcal{F})$ satisfies the D2-separation formulas for all pairs of transitions in $\mathcal{S}$, not just those in τ_i.

[2]https://github.com/rleap-project/d2l.

Results

Table 1 provides an overview of the execution of D2L over all generalized domains. The two main conclusions to be drawn from the results are that 1) our generalized policies are more expressive and result in policies that cannot be captured in previous approaches (Bonet, Francès, and Geffner 2019), 2) our SAT encoding is also simpler and scales up much better, allowing to tackle harder tasks with reasonable computational effort. Also, the new formulation is unsupervised and complete, in the sense that if there is a general policy in the given feature space that solves the instances, the solver is guaranteed to find it.

In all domains, we use a modified version of the Pyperplan planner[3] to check empirically that the learned policies are able to solve a set of test instances of significantly larger dimensions than the training instances. For standard PDDL domains with readily-available instances (e.g., Gripper, Spanner, Miconic), the test set includes all instances in the benchmark set,[4] whereas for other domains such as $\mathcal{Q}_{rew}$, $\mathcal{Q}_{deliv}$ or $\mathcal{Q}_{bw}$, the test set contains at least 30 randomly-generated instances.

We next briefly describe the policy learnt by D2L in each domain; the appendix contains detailed descriptions and proofs of correctness for all these policies (Francès, Bonet, and Geffner 2021a).

Clearing a block. $\mathcal{Q}_{clear}$ is a simplified Blocksworld where the goal is to get $clear(x)$ for a distinguished block x. We use the standard 4-op encoding with stack and unstack actions. Any 5-block training instance suffices to compute the following policy over features $\Phi = \{c, H, n\}$ that denote, respectively, whether x is clear, whether the gripper holds a block, and the number of blocks above x:[5]

$$
\begin{aligned}
r_1 &: \{\neg c, H, n = 0\} \mapsto \{c, \neg H\} , \\
r_2 &: \{\neg c, \neg H, n > 0\} \mapsto \{c?, H, n\!\downarrow\} , \\
r_3 &: \{\neg c, H, n > 0\} \mapsto \{\neg H\} .
\end{aligned}
$$

Rule r_1 applies only when x is held (the only case where $n = 0$ and $\neg c$), and puts x on the table. Rule r_2 picks any block above x that can be picked, potentially making x clear, and r_3 puts down block $y \neq x$ anywhere *not* above x. Note that this policy is slightly more complex than the one defined in (1) because the SAT theory enforces that goals be distinguishable from non-goals, which in the standard encoding cannot be achieved with H and n alone.

Stacking two blocks. $\mathcal{Q}_{on}$ is another simplification of Blocksworld where the goal is $on(x, y)$ for two designated blocks x and y. One training instance with 5 blocks yields a policy over features $\Phi = \{e, c(x), on(y), ok, c\}$. The first four are boolean and encode whether the gripper is empty, x is clear, some block is on y, and x is on y; the last is numerical and encodes the number of clear objects. This version

[3]https://github.com/aibasel/pyperplan.

[4]We have used the benchmark distribution in https://github.com/aibasel/downward-benchmarks.

[5]All features discussed in this section are automatically derived with the description-logic grammar, but we label them manually for readability.

	$\|P_i\|$	dim	$\mathcal{S}$	$\mathcal{S}/\!\sim$	d_{max}	$\|\mathcal{F}\|$	$vars$	$clauses$	t_{all}	t_{SAT}	c_Φ	$\|\Phi\|$	k^*	$\|\pi_\Phi\|$
$\mathcal{Q}_{clear}$	1	5	1,161	55	7	532	7.9K	243.7K(242.3K)	6	< 1	8	3	4	3
$\mathcal{Q}_{on}$	1	5	1,852	329	10	1,412	17.3K	376.6K(281.5K)	33	22	13	5	5	7
$\mathcal{Q}_{grip}$	1	4	1,140	61	12	835	6.5K	102.6K(100.8K)	2	< 1	9	3	4	4
$\mathcal{Q}_{rew}$	1	5×5	432	361	15	514	5.5K	214.9K(98.9K)	2	< 1	7	2	6	2
$\mathcal{Q}_{deliv}$	2	4×4	42,473	5442	56	1,373	753.4K	38.2M(23.5M)	3071	2902	30	4	14	6
$\mathcal{Q}_{visit}$	1	3×3	2,396	310	8	188	13.9K	244.5K(160.6K)	3	< 1	7	2	5	1
$\mathcal{Q}_{span}$	3	(6, 10)	10,777	96	19	764	85.0K	2.2M(2.2M)	32	< 1	9	3	6	2
$\mathcal{Q}_{micon}$	2	(4, 7)	4,706	4,636	14	1,073	23.8K	23.6M(2.4M)	41	61	11	4	5	5
$\mathcal{Q}_{bw}$	2	5	4,275	4,275	8	1,896	22.1K	9.3M(390.0K)	80	40	11	3	6	1

Table 1: *Overview of results*. $|P_i|$ is number of training instances, and dim is size of largest training instance along main generalization dimension(s): number of blocks ($\mathcal{Q}_{clear}$, $\mathcal{Q}_{on}$, $\mathcal{Q}_{bw}$), number of balls ($\mathcal{Q}_{grip}$), grid size ($\mathcal{Q}_{rew}$, $\mathcal{Q}_{deliv}$, $\mathcal{Q}_{visit}$), number of locations and spanners ($\mathcal{Q}_{span}$), number of passengers and floors ($\mathcal{Q}_{micon}$). We fix $\delta = 2$ and $k_{\mathcal{F}} = 8$ in all experiments except $\mathcal{Q}_{deliv}$, where $k_{\mathcal{F}} = 9$. $\mathcal{S}$ is number of transitions in the training set, and $\mathcal{S}/\!\sim$ is the number of distinguishable equivalence classes in $\mathcal{S}$. d_{max} is the max. diameter of the training instances. $|\mathcal{F}|$ is size of feature pool. "Vars" and "clauses" are the number of variables and clauses in the (CNF form) of the theory $T(\mathcal{S}, \mathcal{F})$; the number in parenthesis is the number of clauses in the last iteration of the constraint generation loop. t_{all} is total CPU time, in sec., while t_{SAT} is CPU time spent solving Max-SAT problems. c_Φ is optimal cost of SAT solution, $|\Phi|$ is number of selected features, k^* is cost of the most complex feature in the policy, $|\pi_\Phi|$ is number of rules in the resulting policy. CPU times are given for the incremental constraint generation approach.

of the problem is more general than that in (Bonet, Francès, and Geffner 2019), where x and y are assumed to be initially in different towers.

Gripper. $\mathcal{Q}_{grip}$ is the standard Gripper domain where a two-arm robot has to move n balls between two rooms A and B. Any 4-ball instance is sufficient to learn a simple policy with features $\Phi = \{r_B, c, b\}$ that denote whether the robot is at B, the number of balls carried by the robot, and the number of balls not yet left in B:

$$\begin{aligned} r_1 &: \{\neg r_B, c = 0, b > 0\} \mapsto \{c\!\uparrow\}, \\ r_2 &: \{r_B, c = 0, b > 0\} \mapsto \{\neg r_B\}, \\ r_3 &: \{r_B, c > 0, b > 0\} \mapsto \{c\!\downarrow, b\!\downarrow\}, \\ r_4 &: \{\neg r_B, c > 0, b > 0\} \mapsto \{r_B\}. \end{aligned}$$

In any non-goal state, the policy is compatible with the transition induced by some action; overall, it implements a loop that moves balls from A to B, one by one. Bonet, Francès, and Geffner (2019) also learn an abstraction for Gripper, but need an extra feature g that counts the number of free grippers in order to keep the soundness of their QNP model. Our approach does not need to build such a model, and the policies it learns often use features of smaller complexity.

Picking rewards. $\mathcal{Q}_{rew}$ consists on an agent that navigates a grid with some non-walkable cells in order to pick up scattered reward items. Training on a single 5×5 grid with randomly-placed rewards and non-walkable cells results in the same policy as reported by Bonet, Francès, and Geffner (2019), which moves the agent to the closest unpicked reward, picks it, and repeats. In contrast with that work, however, our approach does not require sample plans, and its propositional theory is one order of magnitude smaller.

Delivery. $\mathcal{Q}_{deliv}$ is the previously discussed Delivery problem, where a truck needs to pick m packages from different locations in a grid and deliver them, one at a time, to a single target cell t. The policy learnt by `D2L` is a generalization to m packages of the one-package policy discussed before.

Visitall. $\mathcal{Q}_{visit}$ is the standard Visitall domain where an agent has to visit all the cells in a grid at least once. Training on a single 3×3 instance produces a single-rule policy based on features $\Phi = \{u, d\}$ that represent the number of unvisited cells and the distance to a closest unvisited cell. The policy, similar to the one for $\mathcal{Q}_{rew}$, moves the agent greedily to a closest unvisited until all cells have been visited.

Spanner. $\mathcal{Q}_{span}$ is the standard Spanner domain where an agent picks up spanners along a corridor that are used at the end to tighten some nuts. Since spanners can be used only once and the corridor is one-way, the problem becomes unsolvable as soon as the agent moves forward and leaves some needed Spanner behind. We feed `D2L` with 3 training instances with different initial locations of spanners, and it computes a policy with features $\Phi = \{n, h, e\}$ that denote the number of nuts that still have to be tightened, the number of objects not held by the agent and whether the agent location is empty, i.e. has no spanner or nut in it:

$$\begin{aligned} r_1 &: \{n > 0, h > 0, e\} \mapsto \{e?\}, \\ r_2 &: \{n > 0, h > 0, \neg e\} \mapsto \{h\!\downarrow, e?\} \mid \{n\!\downarrow\}. \end{aligned}$$

The policy dictates a move when the agent is in an empty location; else, it dictates either to pick up a spanner or tighten a nut. Importantly, it never allows the agent to leave a location with some unpicked spanner, thereby avoiding dead-ends. Note that the features and policy are fit to the domain actions. For instance, an effect $\{e?\}$ as in r_1 could not appear if the domain had *no-op* actions, as the resulting *no-op* transitions would comply with r_1 without making progress to the goal. The learned policy solves the 30 instances of the learning track of the 2011 International Planning Competition, and can actually be formally proven correct over all Miconic instances.

Miconic. $\mathcal{Q}_{micon}$ is the domain where a single elevator moves across different floors to pick up and deliver passengers to their destinations. We train on two instances with a few floors and passengers with different origins and destinations. The learned policy uses 4 numerical features that encode the number of passengers onboard in the lift, the number of passengers waiting to board, the number of passengers waiting to board on the same floor where the lift is, and the number of passengers boarded when the lift is on their target floor. The policy solves the 50 instances of the standard Miconic distribution.

Blocksworld. $\mathcal{Q}_{bw}$ is the classical Blocksworld where the goal is to achieve some desired arbitrary configuration of blocks, under the assumption that each block has a goal destination (i.e., the goal picks a single goal state). We use a standard PDDL encoding where blocks are moved atomically from one location to another (no gripper). The only predicates are on and $clear$, and the set of objects consists of n blocks and the table, which is always clear. We use a single training instance with 5 blocks, where the *target* location of all blocks is specified. We obtain a policy over the features $\Phi = \{c, t', bwp\}$ that stand for the number of clear objects, the number of objects that are not *on* their target location, and the number of objects such that all objects below are well-placed, i.e., in their goal configuration. Interestingly, the value of all features in non-goal states is always positive ($bwp > 0$ holds trivially, as the table is always well-placed and below all blocks). The computed policy has one single rule with four effects:

$$\{c > 0, t' > 0, bwp > 0\} \;\mapsto\; \{c\!\uparrow\} \mid \{c\!\uparrow, t'?, bwp\!\uparrow\} \mid \{c\!\uparrow, t'\!\downarrow\} \mid \{c\!\downarrow, t'\!\downarrow\}\,.$$

The last effect in the rule is compatible with any move of a block from the table into its final position, where everything below is already well-placed (this is the only move away from the table compatible with the policy), while the remaining effects are compatible with moving into the table a block that is not on its final position. The policy solves a set of 100 test instances with 10 to 30 blocks and random initial and goal configurations, and can actually be proven correct.

Discussion of Results. On dead-end free domains where all instances of the same size (same objects) have isomorphic state spaces, `D2L` is able to generate valid policies from one single training instance. In these cases, the only choice we have made regarding the training instance is selecting a size for the instance which is sufficiently large to avoid *overfitting*, but sufficiently small to allow the expansion of the entire state space. As we have seen, though, the approach is also able to handle domains with dead-ends ($\mathcal{Q}_{span}$) or where different instances with the same objects can give rise to non-isomorphic state spaces ($\mathcal{Q}_{rew}$, $\mathcal{Q}_{micon}$). In these cases, the selection of training instances needs to be done more carefully so that sufficiently diverse situations are exemplified in the training set.

As it can be seen in Table 1, the two optimizations discussed at the beginning are key to scale up in different domains. Considering indistinguishable classes of transitions instead of individual transitions offers a dramatic reduction in the size of the theory $T(\mathcal{S}, \mathcal{F})$ for domains with a large number of symmetries such as Spanner, Visitall, and Gripper. On the other hand, the incremental constraint generation loop also reduces the size of the theory up to one order of magnitude for domains such as Miconic and Blocksworld.

Overall, the size of the propositional theory, which is the main bottleneck in (Bonet, Francès, and Geffner 2019), is much smaller. Where they report a number of clauses for $\mathcal{Q}_{clear}$, $\mathcal{Q}_{on}$, $\mathcal{Q}_{grip}$ and $\mathcal{Q}_{rew}$ of, respectively, 767K, 3.3M, 358K and 1.2M, the number of clauses in our encoding is 242.3K, 281.5K, 100.8K and 98.9K, that is up to one order of magnitude smaller, which allows `D2L` to scale up to several other domains. Our approach is also more efficient than the one in (Francès et al. 2019). which requires several hours to solve a domain such as Gripper.

Conclusions

We have introduced a new method for learning features and general policies from small problems without supervision. This is achieved by means of a novel formulation in which a large but finite pool of features is defined from the predicates in the planning examples using a general grammar, and a small subset of features is sought for separating "good" from "bad" state transitions, and goals from non-goals. The problems of finding such a "separating surface" while labeling the transitions as "good" or "bad" are addressed jointly as a Weighted Max-SAT problem. The formulation is complete in the sense that if there is a general policy with features in the pool that solves the training instances, the solver will find it, and by computing the simplest such solution, it ensures a better generalization outside of the training set. In comparison with existing approaches, the new formulation is conceptually simpler, more scalable (much smaller propositional theories), and more expressive (richer class of non-deterministic policies, and value functions that are not necessarily linear in the features). In the future, we want to study extensions for synthesizing provable correct policies exploiting related results in QNPs.

Acknowledgements

This 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 also a Wallenberg Guest Professor at Linköping University, Sweden. G. Francès is partially supported by grant IJC2019-039276-I from MICINN, Spain.

References

Baader, F.; Calvanese, D.; McGuinness, D.; Patel-Schneider, P.; and Nardi, D. 2003. *The description logic handbook: Theory, implementation and applications*. Cambridge U.P.

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

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

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

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.

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*, volume 33, 7530–7537.

Chevalier-Boisvert, M.; Bahdanau, D.; Lahlou, S.; Willems, L.; Saharia, C.; Nguyen, T. H.; and Bengio, Y. 2019. BabyAI: A Platform to Study the Sample Efficiency of Grounded Language Learning. In *Proc. ICLR*.

Fern, A.; Yoon, S.; and Givan, R. 2006. Approximate policy iteration with a policy language bias: Solving relational Markov decision processes. *JAIR* 25: 75–118.

Francès, G.; Bonet, B.; and Geffner, H. 2021a. Extended version of “Learning General Policies from Small Examples Without Supervision”. https://arxiv.org/abs/2101.00692.

Francès, G.; Bonet, B.; and Geffner, H. 2021b. Source code and benchmarks for the paper “Learning General Policies from Small Examples Without Supervision”. https://doi.org/10.5281/zenodo.4322798.

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

Garg, S.; Bajpai, A.; and Mausam. 2020. Symbolic Network: Generalized Neural Policies for Relational MDPs. In *Proc. Machine Learning Research*, 3397–3407.

Garnelo, M.; and Shanahan, M. 2019. Reconciling deep learning with symbolic artificial intelligence: representing objects and relations. *Current Opinion in Behavioral Sciences* 29: 17–23.

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

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

Helmert, M. 2002. Decidability and Undecidability Results for Planning with Numerical State Variables. In *Proc. AIPS*, 44–53.

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.

Khardon, R. 1999. Learning action strategies for planning domains. *Artificial Intelligence* 113: 125–148.

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

Martins, R.; Manquinho, V.; and Lynce, I. 2014. Open-WBO: A modular MaxSAT solver. In *Proc. SAT*, 438–445.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540): 529.

van Otterlo, M. 2012. Solving Relational and First-Order Logical Markov Decision Processes: A Survey. In Wiering, M.; and van Otterlo, M., eds., *Reinforcement Learning*, 253–292. Springer.

Sanner, S.; and Boutilier, C. 2009. Practical Solution Techniques for First-Order MDPs. *Artificial Intelligence* 173(5-6): 748–788.

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

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

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.

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

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

Wang, C.; Joshi, S.; and Khardon, R. 2008. First Order Decision Diagrams for Relational MDPs. *Journal of Artificial Intelligence Research* 31: 431–472.