Learning Features and Abstract Actions for Computing Generalized Plans

Blai Bonet
Universidad Simón Bolívar
Caracas, Venezuela
bonet@usb.ve

Guillem Francès
University of Basel
Basel, Switzerland
guillem.frances@unibas.ch

Hector Geffner
ICREA & Universitat Pompeu Fabra
Barcelona, Spain
hector.geffner@upf.edu

Abstract

Generalized planning is concerned with the computation of plans that solve not one but multiple instances of a planning domain. Recently, it has been shown that generalized plans can be expressed as mappings of feature values into actions, and that they can often be computed with fully observable non-deterministic (FOND) planners. The actions in such plans, however, are not the actions in the instances themselves, which are not necessarily common to other instances, but *abstract* actions that are defined on a set of *common* features. The formulation assumes that the features and the abstract actions are given. In this work, we address this limitation by showing how to learn them automatically. The resulting account of generalized planning combines learning and planning in a novel way: a *learner*, based on a Max SAT formulation, yields the features and abstract actions from sampled state transitions, and a FOND *planner* uses this information, suitably transformed, to produce the general plans. Correctness guarantees are given and experimental results on several domains are reported.

Introduction

Generalized planning studies the computation of plans that solve multiple instances (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, the plan that iteratively picks a clear block above x and places it on the table, achieves the goal $clear(x)$ in *any* instance of the Blocksworld where the gripper is initially empty. Once this general plan or policy is derived, it can be applied to solve an infinite collection of instances that involve different initial states, different objects, and different (ground) actions.

In the basic formulation due to Hu and De Giacomo (2011), a generalized plan is a mapping of observations into actions that are assumed to be common among all the instances. More recently, this formulation has been extended by Bonet and Geffner (2018) to account for relational domains like Blocksworld where the sets of objects and actions change from instance to instance. In the new formulation, the observations are replaced by a set of boolean and numerical features F and a set of *abstract actions* A_F. These abstract actions are *sound* and *complete* if they track the effects of the actions on the features in a suitable way. The resulting generalized plans map feature values into abstract actions, and soundness ensures that the application of an abstract action can be mapped back into the application of a concrete action with the same effect over the features. Moreover, the form of the abstract actions ensures that generalized plans can be computed using fully observable non-deterministic (FOND) planners, once the generalized planning problem is transformed into a FOND problem.

Bonet and Geffner's formulation of generalized planning assumes that the features and abstract actions are given. In this work, we address this limitation by *showing how the features and abstract actions can be learned from the primitive predicates used to define the instances and from sampled state transitions.* For example, the general policy for achieving $clear(x)$ is obtained using a FOND planner on an abstraction that consists of a boolean feature H, that tracks whether the gripper holds a block, a numerical feature $n(x)$ that counts the number of blocks above x, and two abstract actions: one with preconditions $\neg H$ and $n(x) > 0$, and effects H and $n(x)\downarrow$ (decrement of $n(x)$), and the other with precondition H and effect $\neg H$. We here show how to obtain such policies from STRIPS instances alone, without having to provide the features and the abstract actions by hand.

This work relates to a number of research threads in planning, knowledge representation, and machine learning. We make use of SAT solvers and description logics for learning features and abstract actions. The abstract actions provide a model from which the plans are obtained via transformations and FOND planners (Geffner and Bonet 2013; Ghallab, Nau, and Traverso 2016). The model is not the model of an instance but a generalized model for obtaining plans that work for multiple instances. In this sense, the work is different than action model learning (Yang, Wu, and Jiang 2007) and model-based reinforcement learning and closer in aims to work on learning general policies from examples or experience (Martín and Geffner 2004; Fern, Yoon, and Givan 2006; Sukhbaatar et al. 2015; Zhang et al. 2018). Generalized planning has also been formulated as a problem in first-order logic (Srivastava, Immerman, and Zilberstein 2011), 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). Our ap-

proach differs from first-order approaches in the use of propositional planners, and from purely learning approaches in the formal guarantees that are characteristic of planning: if the learned abstract actions are sound, the resulting general plans must be correct.

The paper is organized as follows. We first provide the relevant background, and then show how features and abstract actions can be learned by enforcing soundness and completeness over a set of samples, from a pool of candidate features derived from the domain predicates. The computational model is then summarized, followed by experimental results and a discussion.

Background

We review generalized planning, abstract actions, and solutions following Bonet and Geffner (2018).

Generalized Planning

A *generalized planning problem* $\mathcal{Q}$ is a collection of planning instances P. An instance P is a classical planning problem expressed in some compact language as a tuple $P = \langle V, I, G, A\rangle$ where V is a set of state variables that can take a finite set of values (boolean or not), I is a set of atoms over V defining an initial state s_0, G is a set of literals over V describing the goal states, and A is a set of actions a with their preconditions and effects, which define the set $A(s)$ of actions applicable in any state s, and the successor state function $f(a, s)$, for any state s and $a \in A(s)$. A state is a valuation over V, and a solution to P is an applicable action sequence $\pi = a_0, \ldots, a_n$ that generates a state sequence $s_0, s_1, \ldots, s_n$ where s_n is a goal state (makes G true). In this sequence, $a_i \in A(s_i)$ and $s_{i+1} = f(a_i, s_i)$ for $i = 0, \ldots, n-1$. A state s is reachable in P if $s = s_n$ for one such sequence. A solution to the *generalized problem* $\mathcal{Q}$ is a solution to all instances $P \in Q$. The form of such solutions is described below.

A *feature* f for a class $\mathcal{Q}$ of problems represents a function ϕ_f that takes an instance P from $\mathcal{Q}$ and a state s reachable in P, and results in a value $\phi_f(s)$. A feature is *boolean* if it results in boolean values, and *numeric* if it results in numerical values, here assumed to be non-negative. For example, H and $n(x)$ are two features in Blocksworld: H, boolean, tracks whether the gripper is empty, while $n(x)$, numerical, tracks the number of blocks above x.

Symbols like x denote *parameters* whose value depends on the instance P in $\mathcal{Q}$. For example, if $\mathcal{Q}_{clear}$ denotes the Blocksworld instances with goals of the form $clear(x)$, then a problem P with goal $clear(A)$ will belong to $\mathcal{Q}_{clear}$ and the value of x in P will be (the block) A.

While instances P in a generalized problem $\mathcal{Q}$ would normally share some structure (Bonet and Geffner 2015), like the same planning domain and predicate symbols, this is not strictly necessary. On the other hand, the features f must be common to all the instances in $\mathcal{Q}$ and represent functions $\phi_f(s)$ that are well defined over all reachable states s.

Abstract Actions

An *abstract action* for a generalized problem $\mathcal{Q}$ and a set F of features is a pair $\bar{a} = \langle Pre; \mathit{Eff}\rangle$ where Pre and Eff are the preconditions and effects expressed in terms of the set V_F of boolean and numerical *state variables* associated with the features. Preconditions and effects over boolean state variables p are literals of the form p and $\neg p$, that abbreviate the atoms $p = true$ and $p = false$, while preconditions and effects over numerical state variables n are of the form $n = 0$ and $n > 0$, and $n\downarrow$ (decrements) and $n\uparrow$ (increments), respectively. The language for the abstraction, that combines boolean and numerical variables that can be decreased or increased by unspecified amounts, is the language of *qualitative numerical problems (QNPs)*. Unlike standard numerical planning problems (Helmert 2002), QNPs are decidable and can be solved effectively by means of FOND planners (Srivastava et al. 2011; Bonet et al. 2017).

Features f refer to *state functions* $\phi_f(s)$ in the instances P of the generalized problem $\mathcal{Q}$, but to *state variables* in the abstraction. An *abstract state* is a *truth valuation* over the atoms p and $n = 0$ defined over the state variables p and n associated with the boolean and numerical features. The abstract state $\bar{s}$ that *corresponds* to a concrete state s in an instance P of $\mathcal{Q}$ is the truth valuation that makes p true iff $\phi_p(s) = true$, and $n = 0$ true iff $\phi_n(s) = 0$. An abstract action $\bar{a}$ is applicable in s if its preconditions are true in $\bar{s}$.

An action a and an abstract action $\bar{a} = \langle Pre; \mathit{Eff}\rangle$ *have the same qualitative effects over the features F in a state s* when both are applicable in s with the same effects on the boolean features and the same *qualitative effects* on the numerical features. If s' is the result of applying a in s, this means that (Bonet and Geffner 2018): 1) $p \in \mathit{Eff}$ and $\neg p \in Pre$ iff $\phi_p(s)$ is false and $\phi_p(s')$ is true (i.e., p becomes true), 2) $\neg p \in \mathit{Eff}$ and $p \in Pre$ iff $\phi_p(s)$ is true and $\phi_p(s')$ is false (p becomes false), 3) $n\uparrow \in \mathit{Eff}$ iff $\phi_n(s') > \phi_n(s)$ (n increases), and 4) $n\downarrow \in \mathit{Eff}$ iff $\phi_n(s') < \phi_n(s)$ (n decreases).

Example. Let $\mathcal{Q}_{clear}$ stand for all Blocksworld instances with stack and unstack actions and goal $clear(x)$, let $F = \{H, n(x)\}$ be the set with the two features above, and let s be a reachable state in an instance P in $\mathcal{Q}_{clear}$ where the gripper is empty, the atoms $on(A, B)$ and $clear(A)$ are true, and the block A is above x. In this state s, the abstract action $\bar{a} = \langle \neg H, n(x) > 0; H, n(x)\downarrow\rangle$ and the concrete action $a = \text{Unstack}(A, B)$ have the same effect over the features. Indeed, $\bar{a}$ and a are both applicable in s, and while $\bar{a}$ makes the variable H true and decreases $n(x)$, the action a results in a state s' where $\phi_H(s')$ is true and $\phi_{n(x)}(s') < \phi_{n(x)}(s)$. □

Sound and Complete Abstractions

Soundness and completeness are the key properties that enable us to reason about the collection of instances $\mathcal{Q}$ in terms of abstract actions that operate at the level of the features:

Definition 1. *A set of abstract actions A_F over the features F is* sound *relative to $\mathcal{Q}$ iff for any reachable state s over an instance P in $\mathcal{Q}$, if an abstract action $\bar{a}$ in A_F is applicable in $\bar{s}$, there is an action a in P that is applicable in s and has the* same qualitative effects *over the features as $\bar{a}$.*

Definition 2. *A set of abstract actions A_F is* complete *iff for any reachable state s over an instance P in $\mathcal{Q}$ and any ac-*

tion a in P that is applicable in s, there is an abstract action $\bar{a}$ in A_F that is applicable in $\bar{s}$ and has the same qualitative effects over the features as a.

Let us say that action a *instantiates* abstract action $\bar{a}$ in a state s, and $\bar{a}$ *captures* a in s, when a and $\bar{a}$ are both applicable in s and have the same qualitative effects over the features. Then, soundness means that any abstract action that is applicable in some reachable state can always be instantiated by a concrete action, while completeness means that any concrete action that is applicable in some reachable state is always captured by an abstract action.

It may well be possible that a feature set F does not support a set of abstract actions A_F that is both sound and complete relative to $\mathcal{Q}$. Soundness, however, is crucial for deriving general plans that are valid. An important intuition is that the features in F support a set of sound abstract actions A_F when the effects of the concrete actions over F can be *predicted*; that is, the qualitative values of the features in a state s of an instance P, as captured by $\bar{s}$, *determine* the possible ways in which the actual values of the features may change in the transitions from s to a successor state s'. Hence, if a and b are two actions applicable in s such that one makes p true and the other increases the value of n, then in any other state t such that $\bar{t} = \bar{s}$, there should be applicable actions a' and b' with the same qualitative effects on p and n.

Example. The abstract action set $A_F = \{\bar{a}, \bar{a}'\}$ where $\bar{a} = \langle \neg H, n(x) > 0; H, n(x)\downarrow \rangle$ and $\bar{a}' = \langle H; \neg H \rangle$, defined over the feature set $F = \{H, n(x)\}$, is sound but not complete relative to $\mathcal{Q}_{clear}$. The actions in A_F capture the concrete actions that pick blocks from above x and put them away, but not other actions like stacking a block above x, or picking a block that is not above x. □

Solutions

A solution to a problem $\mathcal{Q}$ given the features F and abstract actions A_F is a partial function π that maps *abstract states* into *abstract actions* such that π solves all instances P in $\mathcal{Q}$. The plan or policy π induces a trajectory $s_0, a_0, s_1, \ldots, s_n$ in an instance P of $\mathcal{Q}$ iff 1) s_0 is the initial state in P, 2) a_i is one of the actions that instantiate $\pi(\bar{s}_i)$ in s_i, 3) a_i is applicable in s_i and $s_{i+1} = f(a_i, s_i)$, and 4) s_n is a goal state of P, or $\pi(\bar{s}_n)$ is undefined, not applicable in $\bar{s}_n$, or no applicable action a_n in P instantiates $\pi(\bar{s}_n)$. The policy π solves P iff all trajectories induced by π reach a goal state of P.

Example. A solution for $\mathcal{Q}_{clear}$ with $F = \{H, n(x)\}$ is the policy π given by the rules $\neg H, n(x) > 0 \Rightarrow \bar{a}$ and $H, n(x) > 0 \Rightarrow \bar{a}'$ for the abstract actions $\bar{a}$ and $\bar{a}'$ above. The policy picks blocks above x and puts them aside (never above x) until $n(x)$ becomes zero. □

Computation

The steps for obtaining policies π for a generalized problem $\mathcal{Q}$ using the features F and abstract actions A_F are as follows (Bonet and Geffner 2018):

1. The state variables V_F and the abstract actions A_F are extended with initial and goal formulas I_F and G_F over V_F to yield the *abstraction* $Q_F = \langle V_F, I_F, G_F, A_F \rangle$, which is a QNP. For soundness, I_F must be such that the initial states of instances P in $\mathcal{Q}$ all satisfy I_F, while all states that satisfy G_F must be goal states of P.
2. The abstraction $Q_F = \langle V_F, I_F, G_F, A_F \rangle$ is converted into a *boolean FOND* problem $Q'_F = \langle V'_F, I'_F, G'_F, A'_F \rangle$ by replacing the numerical variables $n \in N$ by the symbols $n = 0$, the first-order literals $n = 0$ by propositional literals $n = 0$, the effects $n\uparrow$ by effects $n > 0$, and the effects $n\downarrow$ by non-deterministic effects $n > 0 \,|\, n = 0$.
3. The solutions computed by a FOND planner on Q'_F are the strong cyclic solutions of Q'_F (Cimatti, Roveri, and Traverso 1998). Such solutions however do not necessarily solve Q_F because the non-deterministic effects $n > 0 \,|\, n = 0$ in Q'_F are not *fair* but *conditionally fair*: infinite decrements of n ensure that $n = 0$ is true eventually but *only* when the number of increments of n is finite. The problem of obtaining solutions of Q'_F that only assume conditional fairness, the so-called terminating solutions (Srivastava et al. 2011), is mapped into the problem of solving an amended FOND Q^+_F that assumes standard fairness (Bonet et al. 2017).[1]

Theorem 3 (Bonet and Geffner (2018))**.** *If the abstract actions A_F are sound relative to the generalized problem $\mathcal{Q}$, the solutions to Q^+_F computed by FOND planners off-the-shelf are solutions to $\mathcal{Q}$.*

Example. Let us restrict $\mathcal{Q}_{clear}$ to those instances where the gripper is initially empty and there are blocks on top of x. For $F = \{H, n(x)\}$ and $A_F = \{\bar{a}, \bar{a}'\}$ as above, $Q_F = \langle V_F, I_F, G_F, A_F \rangle$ may be defined with $I_F = \{\neg H, n(x) > 0\}$ and $G_F = \{n(x) = 0\}$. Q'_F is like Q_F but with $n(x) = 0$ regarded as a propositional symbol, $n(x) > 0$ as its negation, and the effect $n(x)\downarrow$ replaced by $n(x) > 0 \,|\, n(x) = 0$. Since no action in A_F increases $n(x)$, the strong solutions of Q'_F are solutions to $\mathcal{Q}_{clear}$ (Bonet et al. 2017). □

Approximate Soundness and Completeness

The formulation and computational model above, from (Bonet and Geffner 2018), assume that the features F and abstract actions A_F are given. The contribution of this work is a *method for learning them automatically by enforcing a form of soundness and completeness over a set of samples:*

Definition 4. *For a generalized problem $\mathcal{Q}$, a* sample set *$\mathcal{S}$ is a non-empty set of tuples (s, s', P) such that 1) P is an instance of $\mathcal{Q}$, 2) s is a reachable state in P, 3) s' is a successor of s in P; i.e., $s' = f(a, s)$ for some action $a \in A(s)$, and 4) $\mathcal{S}$ is closed in the sense that if $(s, s', P) \in \mathcal{S}$, then $(s, s'', P) \in \mathcal{S}$ for any successor s'' of s in P.*

By assuming that sampled states s are tagged with the instance P, we abbreviate the tuples (s, s', P) as (s, s'), and refer to the states s' in the pairs $(s, s') \in \mathcal{S}$ as the successors of s in $\mathcal{S}$. The closure condition 4) requires that states s appearing *first* in pairs (s, s') must be fully expanded in $\mathcal{S}$; namely, all possible transitions (s, s'') must be in the sample. We call them the *expanded* states in $\mathcal{S}$.

[1] Translator available at https://github.com/bonetblai/qnp2fond.

For defining soundness and completeness *over a sample set*, we say that a transition (s, s') and an abstract action $\bar{a}$ have the same qualitative effects over the features, when in the state s, the action a that maps s into s' and the abstract action $\bar{a}$ have the same qualitative effects over the features:

Definition 5. *A set of abstract actions A_F is sound* relative to a sample set $\mathcal{S}$ *for* $\mathcal{Q}$ *iff for any abstract action $\bar{a}$ in A_F applicable in an expanded state s of $\mathcal{S}$, there is a transition $(s, s') \in \mathcal{S}$ with the same qualitative effects over F as $\bar{a}$.*

Definition 6. *A set of abstract actions A_F is complete* relative to a sample set $\mathcal{S}$ *for* $\mathcal{Q}$ *iff for each transition (s, s') in $\mathcal{S}$, there is an abstract action $\bar{a}$ in A_F with the same qualitative effects over F that is applicable in s.*

For a sufficiently large sample set, the approximate and exact notions of soundness and completeness converge.

Learning Features and Abstract Actions

In order to learn the features and abstract actions from a sample set $\mathcal{S}$ for $\mathcal{Q}$, we define a propositional formula $T(\mathcal{S}, \mathcal{F})$, where $\mathcal{F}$ represents a large pool of candidate features, that is *satisfiable* iff there is a set of abstract actions A_F over $F \subseteq \mathcal{F}$ that is sound and complete relative to $\mathcal{S}$.

SAT Encoding: $T(\mathcal{S}, \mathcal{F})$ and $T_G(\mathcal{S}, \mathcal{F})$

For each transition $(s, s') \in \mathcal{S}$ and each feature $f \in \mathcal{F}$, $\Delta_f(s, s') \in \{+, -, \uparrow, \downarrow, \perp\}$ denotes the *qualitative change of value* of feature f along the transition (s, s'), which can go from false to true ($+$), from true to false ($-$) or remain unchanged ($\perp$), for boolean features, and can increase ($\uparrow$), decrease ($\downarrow$), or remain unchanged ($\perp$), for numerical features. The *propositional variables* in $T(\mathcal{S}, \mathcal{F})$ are then:

- $selected(f)$ for each $f \in \mathcal{F}$, true iff f selected (in F).
- $D_1(s, t)$ for states s and t expanded in $\mathcal{S}$, true iff the selected features distinguish s from t; i.e., if s and t disagree on the truth value of some selected feature, either a boolean or numeric feature,
- $D_2(s, s', t, t')$ for each (s, s') and (t, t') in $\mathcal{S}$, true iff some selected feature f distinguishes the two transitions; i.e., $\Delta_f(s, s') \neq \Delta_f(t, t')$ for some selected feature f.

The first formulas in $T(\mathcal{S}, \mathcal{F})$ capture the meaning of D_1:

$$D_1(s, t) \;\Leftrightarrow\; \bigvee_f selected(f) \tag{1}$$

where f ranges over the features in $\mathcal{F}$ with different qualitative values in s and t; namely, boolean features p with different truth values in s and t, and numerical features n for which the atom $n = 0$ has different truth values in s and t.

The second class of formulas encode the meaning of D_2:

$$D_2(s, s', t, t') \;\Leftrightarrow\; \bigvee_f selected(f) \tag{2}$$

where f ranges over the features in $\mathcal{F}$ that have the same qualitative values in s and t but which change differently in the two transitions; i.e., $\Delta_f(s, s') \neq \Delta_f(t, t')$.

The third class of formulas relate D_1 and D_2 by enforcing *soundness* and *completeness* over the sample:

$$\neg D_1(s, t) \;\Rightarrow\; \bigvee_{t'} \neg D_2(s, s', t, t') \tag{3}$$

where s and t are expanded states in $\mathcal{S}$, s' is a successor of s in $\mathcal{S}$, and t' ranges over the successors of t in $\mathcal{S}$. This formula is crucial. It says that *if the selected features do not distinguish state s from t, then for each action a that maps s into s', there must be an action b that maps the state t into a state t' such that the two transitions (s, s') and (t, t') affect the selected features in the same way.* The formula does not mention the actions a and b because their identity does not matter; it is just the state transitions that count. In addition, the formula does not involve abstract actions as they will be obtained from the transitions and the satisfying assignment. Indeed, for each transition (s, s') in the sample, there will be an abstract action $\bar{a}$ that accounts for the transition; i.e., with the same qualitative effects over the selected features. Moreover, if $D_1(s, t)$ and $D_2(s, s', t, t')$ are both false in the satisfying assignment, the two transitions (s, s') and (t, t') will be captured by the same abstract action.

The fourth and last class of formulas force the selected features to distinguish goal from non-goal states as:

$$D_1(s, t) \tag{4}$$

where s and t are expanded states in $\mathcal{S}$ such that exactly one of them is a goal state. For this, it is assumed that the states in the sample are labeled as goal or non-goal states.

The SAT theory $T(\mathcal{S}, \mathcal{F})$ given by formulas (1)–(4) has $|\mathcal{F}| + m^2(b^2 + 1)$ propositional variables, where m is the number of expanded states in $\mathcal{S}$, and b is their average branching factor (transitions per state). The number of clauses is bounded by $m^2(2 + b^2 + b + |\mathcal{F}|(b^2 + 1))$.

We also consider an alternative SAT theory $T_G(\mathcal{S}, \mathcal{F})$ that is similar to $T(\mathcal{S}, \mathcal{F})$ but smaller. It is obtained by marking some transitions (s, s') in $\mathcal{S}$ as *goal relevant*. Then, rather than creating abstract actions to account for all the transitions, we only create abstract actions to account for the marked transitions. This is achieved by drawing the states s and the transitions (s, s') in formulas (1) and (2) from the set of marked transitions in $\mathcal{S}$. The states t and the transitions (t, t'), on the other hand, are drawn from the whole sample set $\mathcal{S}$ as before. This simplification preserves soundness over $\mathcal{S}$ but completeness is not over $\mathcal{S}$ but over the marked transitions in $\mathcal{S}$. The goal-relevant transitions in $\mathcal{S}$ are obtained by computing one plan for each sampled instance P from $\mathcal{Q}$.

Extracting F and A_F

For a satisfying assignment σ of the theories $T(\mathcal{S}, \mathcal{F})$ and $T_G(\mathcal{S}, \mathcal{F})$, let F_σ be the set of features f in $\mathcal{F}$ such that $selected(f)$ is true in σ, and let A_σ be the set of abstract actions that *capture* all transitions (s, s') in $\mathcal{S}$, in the case of theory T, and the goal-relevant transitions (s, s') in $\mathcal{S}$, in the case of T_G. The abstract action $\bar{a} = \langle Pre; Eff \rangle$ that captures the transition (s, s') has the *precondition* p (resp. $\neg p$) if p is a boolean feature in F_σ that is true (resp. false) in s, and has the precondition $n = 0$ (resp. $n > 0$) if n is a numerical feature in F_σ such that $n = 0$ is true (resp. false) in s. Similarly, $\bar{a}$ has the *effect* p (resp. $\neg p$) if p is a boolean feature in F_σ that is true (resp. false) in s' but false (resp. true) in s, and the effect $n\downarrow$ (resp. $n\uparrow$) if n is a numerical feature in F_σ whose values decreases (resp. increases) in the transition

from s to s'. Duplicate abstract actions are removed, and if two abstract actions $\bar{a}$ and $\bar{a}'$ differ only in the sign of a precondition, the two abstract actions are merged into one with the precondition dropped.

Theorem 7. *The theory $T(\mathcal{S}, \mathcal{F})$ is satisfiable iff there is a set of features $F \subseteq \mathcal{F}$ and a set of abstract actions A_F over F such that A_F is sound and complete relative to $\mathcal{S}$.*

Theorem 8. *If σ is a satisfying assignment of $T(\mathcal{S}, \mathcal{F})$, the set A_σ of abstract actions over F_σ is sound and complete relative to $\mathcal{S}$.*

Theorem 9. *If σ is a satisfying assignment of $T_G(\mathcal{S}, \mathcal{F})$, the set A_σ of abstract actions over F_σ is sound relative to $\mathcal{S}$ and complete relative to the marked transitions in $\mathcal{S}$.*

While the initial and goal expressions I_F and G_F of the abstract problem can be learned from the satisfying assignment σ as well, for simplicity, in the examples below they are provided by hand.[2]

Different assignments σ result in different feature sets F_σ and abstract action sets A_σ. Simpler and more meaningful features are found by minimizing a cost measure such as $|F_\sigma|$ or, more generally, $\sum_{f \in F_\sigma} cost(f)$. To achieve this, we cast our problem into a constrained optimization problem, more specifically, a Weighted Max-SAT problem, where clauses resulting from the theories T or T_G are taken as *hard* clauses, and we add, for each feature $f \in \mathcal{F}$, a *soft* unit clause $\neg selected(f)$ with weight $cost(f)$, defined below.

Feature Pool

The feature pool $\mathcal{F}$ used in $T(\mathcal{S}, \mathcal{F})$ is obtained from the predicates encoding the instances in $\mathcal{Q}$ that are assumed to be common to all such instances. From these *primitive predicates* and some composition rules, we define a large set of *derived predicates r of arity 1* whose denotation r^s in a state s over an instance P refers to the set of constants (objects) c in P that have the property r in s; i.e., $r^s = \{c \,|\, r(c) \text{ is true in } s\}$. Boolean and numerical features p_r and n_r can then be defined from r, denoting the functions $\phi_{p_r}(s)$ and $\phi_{n_r}(s)$: the first is 0 if $|r^s| = 0$ and else is 1, the second represents the cardinality $|r^s|$ of r in s.

Unary predicates $r(x)$ can be created by definitions of the form "$r(x)$ iff $\exists y[q(x, y) \wedge t(y)]$" where q and t are primitive predicates of arity 2 and 1 respectively. We use instead the syntax and grammar of description logics (Baader et al. 2003). In description logics, unary and binary predicates are referred to as *concepts* C and *roles* R. Description logics have been used for capturing general policies using purely learning methods (Martín and Geffner 2004; Fern, Yoon, and Givan 2006).

Concepts

The concepts and roles denoted as C_p and R_p represent the *primitive predicates* of arity 1 and 2 respectively. The grammar for generating new concepts and roles from them is:

- $C_A \leftarrow C_p, C_u, C_x$, primitive, universal, nominals: C_u denotes universe, C_x denotes $\{x\}$ for parameter x if any,
- $C \leftarrow \neg C_A$, negation on primitive, universal, nominals,
- $C \leftarrow C \sqcap C'$, conjunctions,
- $C \leftarrow \exists R.C, \forall R.C$, first denotes $\{x : \exists y[R(x, y) \wedge C(y)]\}$, the second denotes $\{x : \forall y[R(x, y) \wedge C(y)]\}$,
- $C \leftarrow R = R'$, denotes $\{x : \forall y[R(x, y) = R'(x, y)]\}$,
- $R \leftarrow R_p, R_p^{-1}, R_p^+, [R_p^{-1}]^+$: primitive, inverse, and transitive closure of both.

The denotations C^s and R^s for concepts and roles follows from the rules and the denotation of primitive concepts and roles. For example, for the concept $C : \exists on^+.C_x$ in Blocksworld, C^s denotes the set of blocks that are above x.

Candidate Features

The *complexity* of a concept or role is the minimum number of grammar rules needed to generate it. The set of concepts and roles with complexity no greater than k is referred to as $\mathcal{G}^k$. When generating $\mathcal{G}^k$, redundant concepts are incrementally pruned. A concept is deemed redundant when its denotation coincides with the denotation of a previously generated concept over all states in $\mathcal{S}$. The set $\mathcal{F} = \mathcal{F}^k$, defined from $\mathcal{G}^k$, contains the following features:

- For each nullary primitive predicate p, a *boolean* feature b_p that is true in s iff p is true in s.
- For each concept C, a *boolean* feature b_C, if $|C^s| \in \{0, 1\}$ for all sampled states s, and a *numerical* feature n_C otherwise. The value of b_C in s is true iff $|C^s| > 0$; the value of n_C is $|C^s|$.
- Numerical features *dist*$(C_1, R{:}C, C_2)$ that represent the smallest n such that there are objects $x_1, \ldots, x_n$ satisfying $C_1^s(x_1)$, $C_2^s(x_n)$, and $(R{:}C)^s(x_i, x_{i+1})$ for $i = 1, \ldots, n$. The denotation $(R{:}C)^s$ contains all pairs (x, y) in R^s such that $y \in C^s$.

The measure $cost(f)$ is set to the complexity of C for n_C and b_C, to 0 for b_p, and to the sum of the complexities of C_1, R, C, and C_2, for *dist*$(C_1, R{:}C, C_2)$. Only features with cost bounded by k are allowed in $\mathcal{F}^k$.

Computational Model: Summary

The steps for computing general plans are then:

1. a sample set $\mathcal{S}$ is computed from a few instances P of a given generalized problem $\mathcal{Q}$,
2. a pool of features $\mathcal{F}$ is obtained from the predicates in the instances, the grammar, a bound k, and the sample $\mathcal{S}$ (used for pruning),
3. an assignment σ of $T(\mathcal{S}, \mathcal{F})$ or $T_G(\mathcal{S}, \mathcal{F})$ that minimizes $\sum_{f \in F_\sigma} cost(f)$ is obtained using a Max SAT solver,
4. features F and abstract actions A_F are extracted from σ,
5. the abstraction $Q_F = \langle V_F, I_F, G_F, A_F \rangle$ is defined with initial and goal conditions I_F and G_F provided by hand to match $\mathcal{Q}$,

[2] $G_F = G_\sigma$ may be defined as the DNF formula whose terms correspond to the abstract states over F_σ that correspond to goal states in the sample.

6. the FOND problem Q_F^+, constructed automatically from Q_F, is solved with an off-the-shelf FOND planner.

The policy π that results from the last step deals with propositional symbols that correspond to atoms p and $n = 0$ for boolean and numerical features p and n in F. When this policy is applied to an instance P of $\mathcal{Q}$, the value of the features f is obtained from the functions $\phi_f(s)$ that they denote. For example, if $n = n_C$ for a concept C, then $\phi_n(s) = |C^s|$.

If we write $A \models B$ to express that all the states that satisfy A, satisfy B, the formal guarantees can be expressed as:

Theorem 10. *If the abstract actions A_F that are sound relative to the sample set $\mathcal{S}$ are sound, then a policy π that solves Q_F^+ is guaranteed to solve all instances P of $\mathcal{Q}$ with initial and goal situations I and G such that $I \models I_F$ and $G_F \models G$.*

Experimental Results

We evaluate the computational model on four generalized problems $\mathcal{Q}$. For each $\mathcal{Q}$, we select a few "training" instances P in $\mathcal{Q}$ by hand, from which the sample sets $\mathcal{S}$ are drawn. $\mathcal{S}$ is constructed by collecting the first m states generated by a breadth-first search, along with the states generated in an *optimal* plan. The plans ensure that $\mathcal{S}$ contains some goal states and provide the state transitions that are marked as goal relevant when constructing the theory $T_G(\mathcal{S}, \mathcal{F})$, which is the one used in the experiments. $\mathcal{S}$ is closed by fully expanding the states selected. The value of m is chosen so that the resulting number of transitions in $\mathcal{S}$, which depends on the branching factor, is around 500. The bound k for $\mathcal{F} = \mathcal{F}^k$ is set to 8. Distance features $dist$ are used only in the last problem. The Weighted-Max Solver is Open-WBO (Martins, Manquinho, and Lynce 2014) and the FOND planner is SAT-FOND (Geffner and Geffner 2018). The translation from Q'_F to Q_F^+ is very fast, in the order of 0.01 seconds in all cases. The whole computational pipeline summarized by the steps 1–6 above is processed on Intel Xeon E5-2660 CPUs with time and memory cutoffs of 1h and 32GB. Table 1 summarizes the relevant data for the problems, including the size of the CNF encodings corresponding to the theories T and T_G.

Clearing a block. $\mathcal{Q}_{clear}$ contains the Blocksworld instances with goals of the form $clear(x)$ and stack/unstack actions. The primitive predicates, i.e., those appearing in the instances P of $\mathcal{Q}_{clear}$, are $on(\cdot,\cdot)$, $clear(\cdot)$, $ontable(\cdot)$, $holding(\cdot)$, and $handempty$. For this problem, a single training instance P with 5 blocks suffices to learn an abstract model from which a general plan is computed. The set of features F learned from the theory T_G is:

- $H : holding$ (whether some block is being held),
- $X : holding \sqcap C_x$ (whether block x is being held),
- $n(x) : |\exists\, on^+.C_x|$, (number of blocks above block x).

The set A_F of abstract actions learned is:[3]

- put-aside $= \langle \neg X, H; \neg H \rangle$,
- pick-above-$x = \langle \neg X, \neg H, n(x) > 0; H, n(x)\downarrow \rangle$.

The abstraction $Q_F = \langle V_F, I_F, G_F, A_F \rangle$ for $\mathcal{Q}_{clear}$ is set with $I_F = \{\neg H, \neg X, n(x) > 0\}$ and $G_F = \{n(x) = 0\}$. As mentioned above, since no action increments the variable $n(x)$, the strong-cyclic solutions of Q'_F solve $\mathcal{Q}_{clear}$. One such policy is found with the FOND planner in 0.46 seconds. The plan implements a loop that applies the pick-above-x action followed by put-aside, until x becomes clear.

Stacking two blocks. $\mathcal{Q}_{on}$ consists of Blocksworld instances with goals of the form $on(x, y)$, and initial situations where the blocks x and y are in *different towers*. The primitive predicates are the same as in $\mathcal{Q}_{clear}$. Three training instances are used to learn F from T_G; F contains the boolean features E, X and G, for the gripper being empty, the block x being held, and x being on block y, and the numerical features $n(x)$ and $n(y)$ that count the number of blocks above x and y respectively. The learned set of actions A_F is:

- pick-ab-$x = \langle E, \neg X, \neg G, n(x) > 0, n(y) > 0; \neg E, n(x)\downarrow \rangle$,
- pick-ab-$y = \langle E, \neg X, \neg G, n(x) = 0, n(y) > 0; \neg E, n(y)\downarrow \rangle$,
- put-aside-1 $= \langle \neg E, \neg X, \neg G, n(x) = 0; E \rangle$,
- put-aside-2 $= \langle \neg E, \neg X, \neg G, n(x) > 0, n(y) > 0; E \rangle$,
- pick-$x = \langle E, \neg X, \neg G, n(x) = 0, n(y) = 0; \neg E, X \rangle$,
- put-x-aside $= \langle \neg E, X, \neg G, n(x) = 0, n(y) > 0; E, \neg X \rangle$,
- put-x-on-$y = \langle \neg E, X, \neg G, n(x) = 0, n(y) = 0; E, \neg X, G, n(y)\uparrow \rangle$.

The abstraction Q_F with $I_F = \{E, \neg X, \neg G, n(x) > 0, n(y) > 0\}$ and $G_F = \{G\}$ is translated into the FOND problem Q_F^+ which is then solved by the planner in 7.56 seconds. The resulting policy solves the generalized problem $\mathcal{Q}_{on}$: it implements a loop to clear block x, followed by a loop to clear block y, picking then x and placing it on y.

Gripper. $\mathcal{Q}_{gripper}$ involves a robot with grippers whose goal is to move a number of balls from one room into a target room x. Each gripper may carry one ball at a time. The STRIPS predicates are $at\text{-}robby(l)$, $at\text{-}ball(b, l)$, $free(g)$, $carry(b, g)$ that denote, respectively, whether the robot (the ball) is at location l, whether gripper g is free, and whether gripper g carries ball b, plus unary type predicates $room$, $ball$, and $gripper$.

Pipeline 1–6 is fed with two instances P from $\mathcal{Q}_{gripper}$ with two rooms each, one with 4 balls and the other with 5. The set of features that is learned from the theory T_G is:

- $X : at\_robby \sqcap C_x$ (whether robby is in target room),
- $B : |\exists\, at.\neg C_x|$ (number of balls not in target room),
- $C : |\exists\, carry.C_u|$ (number of balls carried),
- $G : |free|$ (number of empty grippers).

The set of abstract actions A_F that is learned is:

- drop-ball-at-$x = \langle C > 0, X; C\downarrow, G\uparrow \rangle$,
- move-to-x-half-loaded $= \langle \neg X, B = 0, C > 0, G > 0; X \rangle$,
- move-to-x-fully-loaded $= \langle \neg X, C > 0, G = 0; X \rangle$,

[3] Feature and action names are provided to make their meaning explicit; the meaning follows from their syntactic form.

				$T(\mathcal{S},\mathcal{F})$		$T_G(\mathcal{S},\mathcal{F})$						
	n	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{F}\rvert$	np	nc	np	nc	t_{SAT}	$\lvert F\rvert$	$\lvert A_F\rvert$	t_{FOND}	$\lvert\pi\rvert$
$\mathcal{Q}_{clear}$	1	927	322	535K	59.6M	7.7K	767K	0.01	3	2	0.46	5
$\mathcal{Q}_{on}$	3	420	657	128K	25.8M	18.3K	3.3M	0.02	5	7	7.56	12
$\mathcal{Q}_{gripper}$	2	403	130	93K	4.7M	8.1K	358K	0.01	4	5	171.92	14
$\mathcal{Q}_{reward}$	2	568	280	184K	11.9M	15.9K	1.2M	0.01	2	2	1.36	7

Table 1: Results: n is number of training instances P, $|\mathcal{S}|$ is number of transitions in $\mathcal{S}$, $|\mathcal{F}|$ is size of feature pool, np and nc are numbers of propositions and clauses in $T(\mathcal{S},\mathcal{F})$ and $T_G(\mathcal{S},\mathcal{F})$, t_{SAT} is time for SAT solver on T_G, $|F|$ and $|A_F|$ are number of selected features and abstract actions, t_{FOND} is time for planner, and $|\pi|$ is size of the resulting policy. Times in seconds.

– pick-ball-not-in-$x = \langle \neg X, B > 0, G > 0; B{\downarrow}, G{\downarrow}, C{\uparrow}\rangle$,

– leave-$x = \langle X, C = 0, G > 0; \neg X\rangle$.

The abstraction Q_F with $I_F = \{B > 0, \neg X, G > 0, C = 0\}$ and $G_F = \{B = 0, C = 0\}$ is translated into Q_F^+ which is solved by the planner in 171.92 seconds. The resulting policy provably solves $\mathcal{Q}_{gripper}$, meaning that it works for any number of grippers and balls.

Collecting rewards. $\mathcal{Q}_{reward}$ contains instances where an agent needs to navigate a rectangular grid to pick up rewards spread on the grid while avoiding *blocked* cells. This is a variation of an example by Garnelo, Arulkumaran, and Shanahan (2016). The STRIPS instances have the primitive predicates $reward(\cdot)$, $at(\cdot)$, $blocked(\cdot)$ and $adjacent(\cdot,\cdot)$ that denote the position of the rewards, of the agent, of the blocked cells, and the grid topology respectively. Pipeline 1–6 is fed with two instances of the problem of sizes 4×4 and 5×5, and having different distributions of blocked cells and rewards. Two numerical features are learned:

– $R : |reward|$ (number of remaining rewards),

– $D : dist(at, adjacent{:}\neg blocked, reward)$ (distance from current cell to closest cell with reward, traversing adjacent, unblocked cells only).

The learned set of abstract actions is:

– collect-reward $= \langle D = 0, R > 0; R{\downarrow}, D{\uparrow}\rangle$,

– move-to-closest-reward $= \langle R > 0, D > 0; D{\downarrow}\rangle$.

The resulting abstraction Q_F with $I_F = \{R > 0, D > 0\}$ and $G_F = \{R = 0\}$ is translated into Q_F^+ which is solved by the planner in 1.36 seconds. The policy moves the agent one step at a time towards the uncollected reward that is closest, as measured by the numerical feature D. Once the reward is reached, the reward is consumed, and the process repeats until there are no more rewards to be collected.

Discussion

The computational bottleneck of the approach is in the size of the SAT theories used to derive the features and the actions. This is the reason why we have chosen to use the more compact theories T_G in the experiments. Additional ideas, however, are required to improve scalability and to make the computational model captured by steps 1–6 more robust.

From the point of view of expressivity, it is not clear how restrictive is the assumption that the features can be obtained from a pool of features defined from the primitive predicates and a general grammar. A relevant argument made by Bonet and Geffner (2018) is that generalized problems over domains with bounded width (Lipovetzky and Geffner 2012) have compact policies in terms of features f whose value $\phi_f(s)$ can be computed in polynomial time for any state s. It is an open question whether such features are captured by the proposed grammar, or a suitable variation. The feature language, however, seems adequate for dealing with arbitrary goals, once *goal predicates* are added to the set of primitive predicates. Goal predicates are "copies" p_G of the primitive predicates p that appear in the goal, and have fixed interpretation[4] (Martín and Geffner 2004).

Finally, our formulation can be extended to handle nondeterministic actions. For this, it is sufficient to replace the formula (3) that links D_1 and D_2 atoms with a formula asserting that, if two states s and t are indistinguishable (i.e., $\neg D_1(s,t)$), then for each action a in s, there is an action b in t such that the set of transitions (t, b, t') generated by b in t cannot be distinguished from the set of transitions (s, a, s') generated by a in s. For this, transitions (s, s') in the sample would need to be tagged with the actions that generated them as (s, a, s').

Summary and Future Work

We have introduced a scheme for computing general plans that mixes learning and planning: a *learner* infers an abstraction made up of features and abstract actions by enforcing soundness and completeness over the samples, and a *planner* uses the abstraction, suitably transformed, to compute general plans. The number of samples required for obtaining correct general plans is small, as the learner does not have to produce plans; it just has to learn the relevant features for the planner to track. Unlike purely learning approaches, the features and the policies are transparent, and the scope and correctness of the resulting general plans can be assessed.

There is an interesting relation between sound and complete abstractions, on the one hand, and the ideas of dimensionality reduction and embeddings in machine learning, on the other (Hamilton, Ying, and Leskovec 2017). Sound

[4] If p is a primitive predicate that appears in the goal, for any state s and tuple $\bar{u}$ of objects, $s \vDash p_G(\bar{u})$ iff $p(\bar{u})$ holds in the goal.

and complete abstractions map the states of the problem instances, whose size is not bounded, into valuations over a small and bounded set of features, while preserving the essential properties of states; namely, how they are changed by actions and whether they denote goal states or not. An interesting challenge for the future is to generalize the proposed methods to other contexts, such as learning from "screen states" in video games, where the sampled states have no known structure and there are no primitive predicates. One way for achieving this would be precisely by learning embeddings that yield sound and complete abstractions or suitable approximations.

Acknowledgments

B. Bonet is partially funded by 2018 Cátedra de Excelencia UC3M-Banco Santander award. G. Francès is funded by the SSX project, European Research Council. H. Geffner is partially funded by grant TIN-2015-67959-P, MINECO, 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., and Geffner, H. 2015. Policies that generalize: Solving many planning problems with the same policy. In *Proc. IJCAI*, 2798–2804.

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

Bonet, B.; De Giacomo, G.; Geffner, H.; and Rubin, S. 2017. Generalized planning: Non-deterministic abstractions and trajectory constraints. In *Proc. IJCAI*, 873–879.

Bonet, B.; Palacios, H.; and Geffner, H. 2009. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *Proc. ICAPS-09*, 34–41.

Boutilier, C.; Reiter, R.; and Price, B. 2001. Symbolic dynamic programming for first-order MDPs. In *Proc. IJCAI*, volume 1, 690–700.

Cimatti, A.; Roveri, M.; and Traverso, P. 1998. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proc. AAAI-98*, 875–881.

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* 25:75–118.

Garnelo, M.; Arulkumaran, K.; and Shanahan, M. 2016. Towards deep symbolic reinforcement learning. *arXiv preprint arXiv:1609.05518*.

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

Geffner, T., and Geffner, H. 2018. Compact policies for non-deterministic fully observable planning as SAT. In *Proc. ICAPS*, 88–96.

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

Hamilton, W.; Ying, R.; and Leskovec, J. 2017. Representation learning on graphs: Methods and applications. *IEEE Data Engineering* 40(3):52–74.

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.

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

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.

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*. Springer. 253–292.

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.

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.

Sukhbaatar, S.; Szlam, A.; Synnaeve, G.; Chintala, S.; and Fergus, R. 2015. Mazebase: A sandbox for learning from games. *arXiv preprint arXiv:1511.07401*.

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

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted Max-SAT. *Artificial Intelligence* 171(2-3):107–143.

Zhang, A.; Lerer, A.; Sukhbaatar, S.; Fergus, R.; and Szlam, A. 2018. Composable planning with attributes. *arXiv preprint arXiv:1803.00512*.