Learning Generalized Unsolvability Heuristics for Classical Planning

Simon Ståhlberg[1] , **Guillem Francès**[2] and **Jendrik Seipp**[1]
[1]Linköping University, Sweden
[2]Universitat Pompeu Fabra, Spain
simon.stahlberg@liu.se, guillem.frances@upf.edu, jendrik.seipp@liu.se

Abstract

Recent work in classical planning has introduced dedicated techniques for detecting unsolvable states, i.e., states from which no goal state can be reached. We approach the problem from a generalized planning perspective and learn first-order-like formulas that characterize unsolvability for entire planning domains. We show how to cast the problem as a self-supervised classification task. Our training data is automatically generated and labeled by exhaustive exploration of small instances of each domain, and candidate features are automatically computed from the predicates used to define the domain. We investigate three learning algorithms with different properties and compare them to heuristics from the literature. Our empirical results show that our approach often captures important classes of unsolvable states with high classification accuracy. Additionally, the logical form of our heuristics makes them easy to interpret and reason about, and can be used to show that the characterizations learned in some domains capture exactly all unsolvable states of the domain.

1 Introduction

For solving planning tasks efficiently via search it is often crucial to detect unsolvable states, i.e., states from which the goal cannot be reached [Junghanns and Schaeffer, 1998; Kolobov *et al.*, 2012; Muise *et al.*, 2012; Cserna *et al.*, 2018]. In classical planning, unsolvable states have traditionally only been recognized implicitly, as a byproduct of the computation of heuristics designed to estimate the cost of reaching the goal from a given state. Most of the standard heuristics are *safe* unsolvability estimators too, in that a state with an infinite heuristic value is guaranteed to be unsolvable. In recent years, there has been a renewed interest in unsolvability detection methods [Bäckström *et al.*, 2013; Hoffmann *et al.*, 2014; Lipovetzky *et al.*, 2016; Ståhlberg, 2017; Eriksson *et al.*, 2017; Steinmetz and Hoffmann, 2017], as witnessed by the 2016 Unsolvability International Planning Competition [Muise and Lipovetzky, 2016; Seipp *et al.*, 2016; Torralba, 2016]. Many of these methods incorporate a preprocessing phase that computes an unsolvability heuristic tailored to the particular instance at hand.

In this paper, we approach the problem of unsolvability detection from a *generalized* perspective, and learn characterizations of unsolvable states for entire planning domains. Although this is obviously not easier than characterizing the unsolvable states of a single instance of the domain, the advantage is that the learned knowledge can be used for many instances. Although deciding unsolvability for a single instance is in general PSPACE-complete, many standard domains have relatively simple descriptions of what constitutes an unsolvable state, which we aim to capture with our approach.

We frame this problem in terms of a standard self-supervised binary classification task, where training data and labels are generated by exhaustive exploration of a few small instances of the planning domain. The features our classifiers use are derived automatically from the first-order predicates and constants that describe the domain, avoiding the need for manual feature engineering. Our approach thus follows the footsteps of recent works that learn logic-based domain control knowledge from the observation of a few instances of the domain [Bonet *et al.*, 2019; Francès *et al.*, 2021]. Whereas these works aim at obtaining general policies, here we focus squarely on characterizing unsolvability.

We present three different learning algorithms that learn simple formulas in disjunctive normal form (DNF) over the domain features. The three approaches differ in the guarantees they provide and in their computational demands. We show that the resulting functions are concise and hence fast to evaluate. Our experiments also confirm that some domains have simple characterizations of unsolvable states that our approaches capture successfully. Furthermore, we leverage the interpretability of the learned formulas in order to show that some of them are sound and complete characterizations for the entire domain.

2 Background

In this section, we review classical planning, description logic and its use in classical planning.

2.1 Classical Planning

We consider deterministic, fully-observable planning problems represented in a fragment of PDDL [Haslum *et al.*,

2019]. Namely, we consider ground PDDL planning problems defined as a tuple $P = \langle \sigma, \mathcal{A}, s_0, \gamma \rangle$. The *function-free vocabulary* σ consists of a set of constant symbols (also called PDDL *objects*) and a set of predicate symbols. We assume w.l.o.g. that σ contains only predicates with arity at most two; higher-arity symbols can be compiled into multiple binary predicates in the standard manner. PDDL types can also be compiled into unary predicates. The set of states $\mathcal{S}_P$ of problem P contains all possible sets of ground atoms over σ. We write $\mathcal{S}$ when P is clear from context. We call $s_0 \in \mathcal{S}$ the *initial state* of the problem and γ the (conjunctive) *goal*. A state $s \in \mathcal{S}$ is a *goal state* if $\gamma \subseteq s$. Each action a in the set of *ground actions* $\mathcal{A}$ has a *precondition* $pre(a)$, an *add* list $add(a)$ and a *delete* list $del(a)$, each of which is a set of ground atoms over σ.

A ground action $a \in \mathcal{A}$ is *applicable* in state $s \in \mathcal{S}$ if $pre(a) \subseteq s$. In that case, applying a to s results in the *successor state* $s[\![a]\!] = (s \setminus del(a)) \cup add(a)$. A sequence of actions $\pi = \langle a_1, \ldots, a_n \rangle$ is applicable in a state s iff for all i, a_i is applicable in state $s[\![a_1]\!] \cdots [\![a_{i-1}]\!]$. We write $s[\![\pi]\!]$ for the state that results from applying π to s. State s' is *reachable* from state s if there is a sequence π such that $s[\![\pi]\!] = s'$. We say that a state is *reachable* in P if it is reachable from the initial state s_0. A state s is *solvable* if there is a goal state that is reachable from s. A solution to the planning task, i.e., a *plan*, is a sequence of actions $\pi = \langle a_1, \ldots, a_n \rangle$ that is applicable in the initial state s_0 and leads to a goal state, i.e., $\gamma \subseteq s_0[\![\pi]\!]$.

Note that each state $s \in \mathcal{S}$ represents a first-order interpretation $\mathcal{I}(s)$ that assigns a truth value to all formulas over σ. We write $s \models \varphi$ if formula φ is true under $\mathcal{I}(s)$.

Planning domains. In this paper we refer to a *generalized planning domain* as a set $\mathcal{Q}$ of planning problems whose vocabularies all have the same predicate symbols, and possibly share some of the constants symbols. A typical instantiation of such a domain is given in PDDL domain files.

2.2 Description Logics

Description logics are a family of knowledge representation formalisms based on tractable fragments of first-order logic [Baader *et al.*, 2004]. They build on the notions of *concepts*, classes of objects that share some property, and *roles*, relations between these objects. Several description logics exist in the literature; we describe the one we use next.

Syntax. *Compound* concepts and roles are defined inductively starting from a given set of *primitive* concepts and roles. Primitive concepts are unary predicates, whereas primitive roles are binary predicates. Each primitive concept is a concept, and each primitive role is a role. The *universal concept* $\top$ and the *bottom concept* $\bot$ are also concepts. Let C and C' be concepts, and R and R' roles. The *negation* $\neg C$, the *union* $C \sqcup C'$, the *intersection* $C \sqcap C'$, the *existential restriction* $\exists R.C$, the *universal restriction* $\forall R.C$, and the *role-value-map* $R = R'$ are also concepts. If a is a constant symbol, the *nominal* $\{a\}$ is a concept. The *inverse* role R^{-1} and the (non-reflexive) *transitive closure* role R^+ are also roles.

Semantics. The semantics of concepts and roles are defined relative to a given *universe of discourse* Δ. A model $\mathcal{M}$ maps each constant symbol a to an element $a^{\mathcal{M}} \in \Delta$, each primitive concept C to a subset $C^{\mathcal{M}} \subseteq \Delta$, and each primitive role R to a subset $R^{\mathcal{M}} \subseteq \Delta \times \Delta$. $\mathcal{M}$ extends to compound concepts and roles as follows:

$$\top^{\mathcal{M}} = \Delta, \qquad \bot^{\mathcal{M}} = \emptyset,$$
$$(\neg C)^{\mathcal{M}} = \Delta \setminus C^{\mathcal{M}}, \qquad \{a\}^{\mathcal{M}} = \{a^{\mathcal{M}}\},$$
$$(C \sqcup C')^{\mathcal{M}} = C^{\mathcal{M}} \cup C'^{\mathcal{M}}, \qquad (C \sqcap C')^{\mathcal{M}} = C^{\mathcal{M}} \cap C'^{\mathcal{M}},$$
$$(\exists R.C)^{\mathcal{M}} = \{a \mid \exists b : (a,b) \in R^{\mathcal{M}} \wedge b \in C^{\mathcal{M}}\},$$
$$(\forall R.C)^{\mathcal{M}} = \{a \mid \forall b : (a,b) \in R^{\mathcal{M}} \rightarrow b \in C^{\mathcal{M}}\},$$
$$(R = R')^{\mathcal{M}} = \{a \mid \forall b : (a,b) \in R^{\mathcal{M}} \leftrightarrow (a,b) \in R'^{\mathcal{M}}\},$$
$$(R^{-1})^{\mathcal{M}} = \{(b,a) \mid (a,b) \in R^{\mathcal{M}}\},$$
$$(R^{+})^{\mathcal{M}} = \{(a_0,a_n) \mid \exists a_1, \ldots, a_{n-1} : (a_{i-1},a_i) \in R^{\mathcal{M}} \text{ for all } 1 \leq i \leq n\}.$$

Complexity. The *complexity* $\mathcal{K}(C)$ of a concept or role C is defined as the number of nodes of the parse tree used to represent it.

2.3 Use of Description Logics in Classical Planning

We follow several works in the literature on learning for planning that use description logics or similar formalisms as the foundation of first-order features useful for the design of generalized features and policies [Martín and Geffner, 2004; Fern *et al.*, 2006; Francès *et al.*, 2019; Francès *et al.*, 2021]. The connection between a planning problem P and a corresponding description logic language $\mathcal{DL}(P)$ is straightforward. The unary and binary predicates of P are taken to be primitive concepts and roles, respectively, and the universe of discourse Δ contains all constants in the problem, which we always consider to denote themselves. Hence, each state $s \in \mathcal{S}_P$ can also be seen as a model for $\mathcal{DL}(P)$, where $C^s = \{a \mid \mathcal{I}(s) \models C(a)\}$ for primitive concepts C, and $R^s = \{(a,b) \mid \mathcal{I}(s) \models R(a,b)\}$ for primitive roles R.

The description language $\mathcal{DL}(P)$ is usually enriched with *goal modalities* p_G of those predicates p that are used to define the goal of the problem [Khardon, 1999]. These are, to all effects, additional primitive concepts and roles, whose denotation is fixed in all states s by the atoms appearing in the goal conjunction. We also limit nominal concepts in $\mathcal{DL}(P)$ to PDDL-level constants, which are those PDDL objects that by definition appear in all instances of the domain. Because of this, all problems of a same planning domain share the same description language, and we sometimes speak of the description language of a domain.

Example. Take as example the Spanner domain, where an agent moves along a corridor, and can pick up single-use spanners that are scattered along the corridor. These spanners are needed at the end of the corridor, where several nuts have to be tightened. The corridor is one-way, hence as soon as the agent leaves sufficiently many spanners behind, the problem becomes unsolvable. The PDDL encoding has a unary predicate *tightened* and a binary predicate *at*, representing the status of nuts and the position of agent, spanners and nuts within the corridor, respectively. The PDDL types of

the original encoding can be compiled into unary predicates. The description language corresponding to this domain will hence have primitive concepts *man*, *nut*, *spanner* and *tightened*, and a primitive role *carrying*. The interpretation of these in a given state s represents sets of objects (for concepts) or object pairs (for roles) that satisfy some property in s. Compound concepts allow us to represent complex sets of objects. The concept $\exists at^{-1}.man$, for instance, represents the set of all corridor locations with an agent in them, whereas $tightened_G \sqcap \neg tightened$ represents the nuts that need to be tightened in the goal, but are not tightened in the current state. Both concepts have syntactic complexity 4.

3 Generalized Unsolvability Heuristics

We are interested in functions that take any state s of any instance in a given class $\mathcal{Q}$ of planning problems, and predict whether s is unsolvable. The next definition generalizes the one by Hoffmann *et al.* [2014]:

Definition 1. *A* generalized unsolvability heuristic *for a class $\mathcal{Q}$ of planning tasks is a function $h : \cup_{P\in\mathcal{Q}}\mathcal{S}_P \mapsto \{0, \infty\}$. We say that h is* safe *whenever $h(s) = \infty$ only if s is unsolvable, and* perfect *whenever $h(s) = \infty$ iff s is unsolvable.*

Example. In the Spanner domain, a state is unsolvable iff the number of unused spanners that are carried by the agent or that can still be picked up is smaller than the number of nuts that still need to be tightened. Any function that returns ∞ for a subset of these states is a safe unsolvability heuristic for the domain; any function that returns ∞ exactly for these states is a perfect unsolvability heuristic.

3.1 Hypothesis Space

Because of their simplicity and interpretability, we focus on learning unsolvability heuristics represented by formulas in disjunctive normal form (DNF) over literals from a given space $\mathcal{F}$ of binary features, which can also be seen as propositional atoms. Our formulas thus have the form $\bigvee_i \bigwedge_j p_{ij}$, where p_{ij} is either a feature in $\mathcal{F}$ or its negation. They can be understood as binary classifiers (a state s is unsolvable iff it satisfies the formula) and as unsolvability heuristics h ($h(s)$ is ∞ if s satisfies the formula, and 0 otherwise).

Feature Space

Our feature space extends the one from Bonet *et al.* [2019] by adding the arithmetic comparison of numerical quantities.

Definition 2. *Let $P = \langle \sigma, \mathcal{A}, s_0, \gamma\rangle$ be a planning problem. The* feature space $\mathcal{F}(P)$ *is the smallest set containing features $|C| > 0$, $|C| > |C'|$ and $|C| = |C'|$ for any two concepts C and C' in $\mathcal{DL}(P)$, and feature f_p for any nullary predicate p of σ. The* truth value *of these features in a state $s \in \mathcal{S}_P$ is given by $|C^s| > 0$, $|C^s| > |C'^s|$, $|C^s| = |C'^s|$, and $s \models p$, respectively. The (syntactic)* complexity $\mathcal{K}(f)$ *of each feature $f \in \mathcal{F}(P)$ is defined as 1 for nullary predicate features f_p, $2 + \mathcal{K}(C)$ for features of the form $|C| > 0$, and $1 + \mathcal{K}(C) + \mathcal{K}(C')$ for features of the form $|C| > |C'|$ and $|C| = |C'|$.*

Note that $\mathcal{F}(P)$ is infinite and contains binary features only. These are either direct translations f_p of nullary predicates p from the domain vocabulary or binary values based on counts $|C|$ of the number of objects that satisfy some property C in a given state, where C is represented in the description language of the problem. When P is clear from context, we simply use $\mathcal{F}$. Note that $\mathcal{F}$ is well-defined for all domains represented in PDDL; the features it contains are tailored to the domain, but they are *instance-independent*, that is, well-defined in all states of all instances of the domain. This is key for generalization. Arithmetic comparison features such as $|C| > |C'|$ and $|C| = |C'|$ are a significant addition to the feature grammar used by Bonet *et al.* [2019] and in related work. In Spanner, for example, they could be used to compare the number of carried spanners with the number of nuts that need to be tightened.

3.2 Learning the Heuristics

We now present three different methods for learning unsolvability heuristics. All of them require two inputs: (1) a finite set F of *candidate features*, and (2) a *training set* $\mathbb{T}$ of states labeled as "solvable" or "unsolvable". We denote with $\mathbb{T}^+$ the set of all states in $\mathbb{T}$ that are unsolvable, and with $\mathbb{T}^-$ the set of those that are solvable.

The set F of candidate features that we use in the experiments is the finite subset of $\mathcal{F}$ that contains features with syntactic complexity at most k. The methods we present, however, do not depend on how F is obtained. As discussed below, our methods have an inductive bias towards formulas that use simple features. This can be seen as a model regularization mechanism to increase the chance of generalizing from training instances to unseen planning problems. Additionally, simpler features are usually much easier to interpret.

The $\mathbb{T}$-PERFECT Heuristic

We first consider a classifier that perfectly discriminates all unsolvable states from all solvable states in the training set $\mathbb{T}$ with a DNF formula over literals from F that has minimum complexity. Here, we measure formula complexity as the sum of the complexities of all distinct features involved in the formula. Such a classifier might not exist if the features in F are not expressive enough. But when it does exist for a large enough training set, it is likely that it generalizes perfectly for the domain. We name this classifier $\mathbb{T}$-PERFECT, and call the DNF formula that it finds $\varphi_{\mathbb{T}\text{-PERFECT}}$.

For fixed training set $\mathbb{T}$ and candidate features F, the computation of $\mathbb{T}$-PERFECT reduces to finding a set of features $F^* \subseteq F$ with minimum complexity such that each pair of states $s \in \mathbb{T}^+$, $t \in \mathbb{T}^-$ can be distinguished by at least one feature $f \in F^*$ (that is, $f^s \neq f^t$). Formula $\varphi_{\mathbb{T}\text{-PERFECT}}$ is then

$$\varphi_{\mathbb{T}\text{-PERFECT}} \equiv \bigvee_{s\in\mathbb{T}^+} \bigwedge_{f\in F^*} L_{f,s},$$

where $L_{f,s}$ is the literal that describes the value of f in s, i.e., the literal f, if f is true in s, and the literal $\neg f$ otherwise.

We solve the problem of finding the set F^* with minimum complexity via a simple compilation to a Weighted Max-SAT problem $\Phi(F, \mathbb{T}^+, \mathbb{T}^-)$. The problem $\Phi(F, \mathbb{T}^+, \mathbb{T}^-)$ has a propositional variable $select(f)$ for each candidate feature $f \in F$, a soft clause $\neg select(f)$ with weight equal to the complexity of f for each feature $f \in F$, and a hard clause $\bigvee_{f^s\neq f^t} select(f)$ for each pair of states $s \in \mathbb{T}^+$, $t \in \mathbb{T}^-$.

The resulting Max-SAT problem thus has $|F|$ variables and $|F| + |\mathbb{T}^+| \cdot |\mathbb{T}^-|$ clauses. Therefore, learning times grow in the worst case quadratically with the size of $\mathbb{T}$ and exponentially with the size of F.

The $\mathbb{T}$-SAFE Heuristic

Since $\mathbb{T}$-PERFECT aims at perfect classification over the training set, it is bound to fail in domains where our feature space is not expressive enough to characterize unsolvability. A better alternative in that case is to aim at capturing some subclasses of unsolvable states by giving up on completeness, but not soundness.

To do this, the approach that we call $\mathbb{T}$-SAFE follows a similar idea as $\mathbb{T}$-PERFECT, but considers unsolvable states $s \in \mathbb{T}^+$ in isolation, looking for a propositional formula ϕ_s over features in F that distinguishes s from all solvable states in $\mathbb{T}^-$ and has minimum complexity. Because ϕ_s is a Boolean function required to be true for only one input, if it exists it can be represented with a conjunction of literals. This is a special case of the optimization problem that $\mathbb{T}$-PERFECT solves: while $\mathbb{T}$-PERFECT separates two sets of states, $\mathbb{T}$-SAFE separates a single state from a set of states. More formally, $\mathbb{T}$-SAFE finds a minimum-complexity set $F^* \subseteq F$ such that for each state $t \in \mathbb{T}^-$, some feature in F^* distinguishes s from t. Note that this set F^* can be computed by solving the Max-SAT problem $\Phi(F, \mathbb{T}^+, \mathbb{T}^-)$ that we defined above, but replacing $\mathbb{T}^+$ by the singleton set $\{s\}$.

If F^* exists, then $\phi_s \equiv \bigwedge_{f \in F^*} L_{f,s}$; otherwise ϕ_s is undefined. The DNF formula $\varphi_{\mathbb{T}\text{-SAFE}}$ that characterizes $\mathbb{T}$-SAFE is then the disjunction of the conjunctions ϕ_s for those states $s \in \mathbb{T}^+$ where ϕ_s is defined. Note that $\varphi_{\mathbb{T}\text{-SAFE}}$ is guaranteed to be safe (i.e., have no false positive) over the training set $\mathbb{T}$.

Our implementation of $\mathbb{T}$-SAFE considers each unsolvable state $s \in \mathbb{T}^+$ individually and therefore solves $|\mathbb{T}^+|$ Max-SAT problems, each of which has $|F|$ variables and $|F| + |\mathbb{T}^-|$ clauses. Thus, learning times have the same asymptotic growth as $\mathbb{T}$-PERFECT. As an optimization, we skip computing clauses ϕ_s for unsolvable states s that are already discriminated by some previously-computed clause. An advantage of $\mathbb{T}$-SAFE is that it can be used as an anytime algorithm: the set of DNF terms computed at any moment is a discriminator over the set of unsolvable states seen up until then.

The DECISIONTREE Heuristic

We finally consider unsolvability heuristics in the form of a binary decision tree [Breiman *et al.*, 1984], a well-known classification method that is easy to interpret and fast to train. In such a tree, inner nodes are labeled with a feature $f \in F$, edges with truth values, and leaf nodes with either "solvable" or "unsolvable". To predict whether a given state s is solvable or not, we traverse the tree by following at each node labeled with f the edge that corresponds to the truth value of f in s until we reach a leaf node, which then gives us the answer.

To learn a decision tree from training data, we use the standard CART algorithm [Breiman *et al.*, 1984]. CART grows the tree in a greedy manner, starting with a single node that predicts the most common class, then at each iteration selecting the single feature that best splits the remaining data according to some information-theoretical measure, until some maximum depth is reached. The overall time for learning the decision tree is linear in the number $|F|$ of features and quadratic in the size of $\mathbb{T}$. Because the learning algorithm is greedy, however, the learned tree is not guaranteed to be the one that best classifies the training data.

For each domain, we choose the tree that performs best on the training set (F1 score) in a 10-fold cross-validation that evaluates different combinations of maximum tree depth $(1, \ldots, 10)$ and maximum feature complexity $(1, \ldots, \max_{f \in F} \mathcal{K}(f))$. We break ties by preferring shallow trees and trees with simple features.

3.3 Learning Pipeline

To sum up, our learning pipeline consists of three steps: state labeling, feature generation, and heuristic learning.

State labeling. Taking as input a PDDL domain and some PDDL problems for the domain, we explore the (reachable part of the) state space of each instance by running a breadth-first search from its initial state. We add all visited states to the training set $\mathbb{T}$, and label them as solvable if a goal state is reachable from them, and unsolvable otherwise.

Feature generation. Taking as input a PDDL domain, the set $\mathbb{T}$, and two constants k and n, we first generate up to n description logic concepts with syntactic complexity at most k, prioritizing concepts with lower complexity. Then, we generate the set F of all Boolean features in our feature space that can be derived from those concepts and also have syntactic complexity of at most k.

Unsolvability heuristic learning. Taking the features in F and the labeled states in $\mathbb{T}$, we compute a DNF formula over atoms in F with one of the algorithms described above.

The entire pipeline is automated, and domain knowledge is only required for generating or selecting the problem instances that are taken as input to the first pipeline step. We describe this process in the next section.

4 Datasets

Before our empirical analysis, we describe the datasets used in our experiments. These are a contribution in their own right and were designed with the goal of being useful for other researchers. To generate the datasets, we need to select domains, problems from each domain, states from each problem, and then compute feature valuations for these states.

Domains. We consider six domains from previous International Planning Competitions (IPC): Barman, Childsnack, Hiking, Nomystery, Spanner and Woodworking. We use these domains because they contain unsolvable states and there are PDDL generators to create new instances of controlled size for them. Two of the domains, Hiking and Nomystery, contain predicates with arity 3. While it would be possible to reformulate the domain, we choose to simply ignore these predicates.

Problems. To label the states, we need instances that are (1) small enough to be explored completely with a breadth-first search, and (2) are as diverse as possible, to maximize the chances of generalization. Since IPC tasks are usually too large for (1), we use PDDL generators to generate new tasks. We choose suitable sets of (low) parameter values for

			Training			Test		
	$\|F\|$	k	$\|P\|$	$\|\mathbb{T}^+\|$	$\|\mathbb{T}^-\|$	$\|P\|$	$\|\mathbb{T}^+\|$	$\|\mathbb{T}^-\|$
Barman	83081	10	16	4110	4836	16	500	500
Childsnack	80617	12	13	2630	2542	12	500	500
Hiking	75638	13	22	457	3866	22	500	500
Nomystery	71719	16	24	677	633	24	500	500
Spanner	68417	15	18	74	318	18	163	500
Woodworking	40749	9	41	4954	4482	41	500	500

Table 1: Training and test sets. $|F|$ is the number of generated features, k is the maximum feature complexity attained. For both sets, $|P|$, $|\mathbb{T}^+|$ and $|\mathbb{T}^-|$ show, resp., the number of instances used to generate the set, and the number of unsolvable and solvable states.

each generator parameter (controlling the number of objects, locations, etc.) and generate tasks for all combinations of values. We keep the tasks that can be fully explored within 30 minutes and 4 GiB. We partition them into training and test sets such that both sets have roughly the same size, both contain instances with small and large state spaces, and in both sets each generator parameter has at least two different values. Table 1 provides an overview of the datasets.

States. For each domain we choose at most 10K states from the state spaces of the training set tasks to train on. We try to select states from the tasks in a balanced way: if the training set has n tasks, we randomly sample at most 5K/n unsolvable and 5K/n solvable states from each task. We prune from $\mathbb{T}^+$ and $\mathbb{T}^-$ states with a feature valuation equal to that of some other state in the same set. The test set is constructed similarly, and contains 1K states per domain. An important aspect of our state selection procedure is that we only keep those unsolvable states that can be reached from a solvable state by applying a single action. We do this because these are the states that one would like to identify when using an unsolvability heuristic as a forward-search pruning mechanism, and because representing this subset of unsolvable states can sometimes be easier than representing all of them.

Features. Finally, we construct a binary valuation matrix that any learning approach can use as input: each column is a feature f, each row is a state s, and each matrix entry holds the binary denotation of f in s. The last column stores the state label. We prune from the matrix duplicate columns, corresponding to redundant features, preferring always the feature with smaller complexity, ties broken arbitrarily.

5 Experiments

We now evaluate our algorithms on the datasets described above. As any inductive learning approach, our algorithms are prone to *generalization error*: even with perfect accuracy on the training set, they might classify unseen states incorrectly. We estimate this error by computing two standard classification metrics over a set of unseen test states: *precision* (how many of the states predicted as unsolvable are indeed unsolvable) and *recall* (how many of the unsolvable states are predicted as such). A safe unsolvability heuristic has precision 1; a perfect one has precision and recall of 1.

5.1 Experiment Setup

The methods $\mathbb{T}$-PERFECT and $\mathbb{T}$-SAFE use the Open-WBO Weighted Max-SAT solver [Martins *et al.*, 2014] and the DECISIONTREE algorithm uses the Scikit-learn machine learning library [Pedregosa *et al.*, 2011]. For all domains, we limit the feature complexity by k=16 and the number of concepts by n=80K in the feature generation step (see Section 3.3). We give each method a maximum of five hours to learn an unsolvability heuristic. We run $\mathbb{T}$-SAFE as an anytime algorithm, but it only reaches the time limit for Barman and Nomystery. All source code, benchmarks and datasets are available online.[1]

5.2 Domain-Independent Heuristics

The main focus of our work is to find out whether we can learn domain-dependent unsolvability heuristics that accurately characterize unsolvable states over whole domains. However, apart from this theoretical question, we are also interested in evaluating how such heuristics compare to existing domain-independent heuristics. The left part of Table 2 shows the recall on our test dataset for some heuristics from the literature: the causal graph heuristic h^{CG} [Helmert, 2004], the context-enhanced additive heuristic h^{CEA} [Helmert and Geffner, 2008], the state-equation heuristic h^{SEQ} [Bonet, 2013], the h^m heuristic for $m \in \{1, 2, 3\}$ [Haslum and Geffner, 2000], and k-consistency [Bäckström *et al.*, 2013] for $k \in \{1, 2, 3\}$. Except for h^{CG} and h^{CEA}, these heuristics are safe and hence guaranteed to have precision 1. Since the two unsafe heuristics have precision 1 on our test set, we only report recall for all of these heuristics.

Note that the data in Table 2 is obtained on small problem instances. Some of the heuristics are impractical to be used for larger tasks. For example, h^3 is very expensive to evaluate and rarely used in practice. We defer the comparison with more sophisticated unsolvability heuristics to a stage where we can directly compare their pruning power in a search on larger instances. This is because some of these heuristics, such as clause learning [Steinmetz and Hoffmann, 2017], are online learning approaches, while others, such as those based on abstraction heuristics [Seipp *et al.*, 2016; Torralba *et al.*, 2016], would yield perfect results on our small instances.

5.3 Domain Analyses

The right part of Table 2 presents the precision and recall of our methods on the test states, together with the size and maximum feature complexity of the DNFs. We next discuss the results for some of the domains.

Barman. In the Barman domain, a robot barman prepares cocktails with the help of shot glasses and a shaker. Unsolvable states in this domain are related to incorrect combinations of drinks in the shaker, which prevent the barman from preparing the right cocktail and, somewhat counterintuitively, prevent it from emptying the shaker as well. Of the standard heuristics that we test, h^2 and h^3 capture all unsolvable states in our test set, h^1 and h^{CEA} capture 88% of them, and the rest have low or zero recall. Among our heuristics, $\mathbb{T}$-PERFECT is

[1] https://doi.org/10.5281/zenodo.4740386

							k-consistency			$\mathbb{T}$-PERFECT						$\mathbb{T}$-SAFE						DECISIONTREE					
	h^{CG}	h^{CEA}	h^{SEQ}	h^1	h^2	h^3	k=1	k=2	k=3	prec	rec	C	L	k	t	prec	rec	C	L	k	t	prec	rec	C	L	k	t
Barman	0.00	0.88	0.39	0.88	1.00	1.00	0.00	0.00	0.00	–	–	–	–	–	–	0.97	0.36	11	18	9	5h	0.97	0.99	28	56	8	6s
Childsnack	0.58	0.58	0.09	0.58	0.94	1.00	0.00	0.27	0.27	–	–	–	–	–	–	1.00	1.00	7	9	11	1h	0.91	0.98	16	32	11	10s
Hiking	0.00	1.00	0.00	1.00	1.00	1.00	0.00	0.00	0.00	1.00	1.00	1	1	8	27m	1.00	1.00	1	1	8	1m	1.00	1.00	1	1	8	1s
Nomystery	0.00	0.53	0.00	0.31	0.91	1.00	0.00	0.00	0.83	–	–	–	–	–	–	0.87	0.12	22	39	17	5h	0.65	0.92	1	1	12	1s
Spanner	0.05	0.05	0.00	0.05	0.13	0.31	0.00	0.00	0.01	1.00	1.00	1	1	13	5m	1.00	1.00	1	1	13	4m	1.00	1.00	1	1	13	1s
Woodworking	1.00	1.00	0.00	1.00	1.00	1.00	1.00	1.00	1.00	0.98	1.00	1	1	8	50s	0.98	1.00	1	1	8	2m	0.98	1.00	1	1	9	6s

Table 2: Left: recall of standard heuristics on test data. Right: precision (prec) and recall (rec) of generalized unsolvability heuristics on test data. C, L, k and t denote the number of clauses, number of literals, max. complexity of any literal, and learning time, respectively.

unable to obtain a formula in less than five hours, and $\mathbb{T}$-SAFE computes a formula that does not generalize correctly, showing imperfect precision. Although DECISIONTREE has no guarantees of being safe on the training set, it finds a classifier with both high precision and recall.

Childsnack. In the Childsnack domain, sandwiches need to be prepared for a number of children, some of which are allergic to gluten. The key for that is to reserve enough gluten-free ingredients to serve the gluten-allergic children. It is easy to manually find a perfect characterization of unsolvable states that is within the hypothesis space of our classifiers, but the complexity of some of the required features (13) is higher than what our feature generator was able to generate before hitting the 80K concept limit that we use, ruling out the possibility of our algorithms learning such a characterization. Despite this, our $\mathbb{T}$-SAFE algorithm still finds a formula that has perfect precision and recall over the test set. DECISIONTREE achieves high accuracy, but is not safe, whereas $\mathbb{T}$-PERFECT times out after five hours while running the Max-SAT solver. Of the standard heuristics that we test, only h^2 and h^3 show similar performance to $\mathbb{T}$-SAFE, capturing 94% and 100% of the unsolvable states in our test set, respectively. The other heuristics have recall of at most 58%.

Hiking. In the Hiking domain, some hikers walk along the different legs of a circular hiking route. Before walking a leg, one of them needs to drive and set up a camping tent at the leg endpoint. A Hiking state is unsolvable when none of the available cars is parked in any of the locations with an agent. Interestingly, $\mathbb{T}$-PERFECT, $\mathbb{T}$-SAFE and DECISIONTREE all converge to the following DNF formula:

$$\varphi \equiv |\exists \mathit{at\_person}.(\exists \mathit{at\_car}^{-1}.\top)| = 0$$

The formula says that a state is unsolvable iff the number of agents at a location with a car is 0, which indeed characterizes all unsolvable states in the domain (hence the perfect precision and recall of our algorithms). Of the standard heuristics that we tested, the three critical path heuristics h^m and h^{CEA} have the same predictive power, whereas the rest of the heuristics fail to capture any unsolvable state.

Spanner. In Spanner, already introduced above, our three algorithms again learn the same formula φ:

$$\varphi \equiv |\mathit{loose} \sqcup \exists \mathit{at}.\left(\exists \mathit{link}^{+}.(\exists \mathit{at}^{-1}.\mathit{man})\right)| > |\mathit{usable}|$$

The inner concept $\exists \mathit{at}.\left(\exists \mathit{link}^{+}.(\exists \mathit{at}^{-1}.\mathit{man})\right)$ denotes the set of all spanners that the agent left behind, i.e., that are no longer reachable. The concept loose denotes the set of all nuts that have yet to be tightened. Since both sets are always disjoint, φ will evaluate to true iff the number of loose nuts is larger than the number of spanners that is usable (i.e., has not yet been discarded) and reachable. The formula can be proven a perfect characterization of all unsolvable states in the domain. This result and the simplicity of the formula are in stark contrast with the fact that Spanner is consistently hard for all the heuristic approaches in the left part of Table 2. Only h^3 captures a significant fraction of the unsolvable states in our test set (31%), whereas the other approaches capture only between 0 and 13%. This is no surprise, as Spanner is a well-known example showing the limitations of delete-relaxed heuristics in problems with consumable resources [Haslum and Geffner, 2001]. The accuracy of our methods illustrates the advantages of a first-order approach.

6 Conclusions

We studied the problem of learning unsolvability heuristics that work for entire classes of planning problems, from the exploration of a few small instances of that class. Our heuristics are simple logical and arithmetical combinations of features that are defined in the same language that is used to specify the planning task and can be generated without manual intervention through the exhaustive application of a standard description-logics grammar. We presented three different algorithms, each with different properties regarding optimality, classification power and learning time. Our evaluation of these algorithms on a number of standard domains shows that the logical characterizations of unsolvability that they learn not only often have strong predictive power, but in some cases can be proven perfect for the entire domain. This is remarkable, as many of the existing heuristics fail to detect many unsolvable states in our dataset.

Acknowledgments

We thank Hector Geffner for valuable insights. This work was supported by ERC Advanced Grant no. 885107, EU ICT-48 2020 project TAILOR (no. 952215) and by the Knut and Alice Wallenberg Foundation (WASP program). We used resources from the Swedish National Infrastructure for Computing (SNIC), partially funded by the Swedish Research Council (no. 2018-05973). G. Francès is supported by grant IJC2019-039276-I from MICINN, Spain.

References

[Baader *et al.*, 2004] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description logics. In *Handbook on ontologies*, pages 3–28. Springer, 2004.

[Bäckström *et al.*, 2013] Christer Bäckström, Peter Jonsson, and Simon Ståhlberg. Fast detection of unsolvable planning instances using local consistency. In *Proc. SoCS 2013*, pages 29–37, 2013.

[Bonet *et al.*, 2019] B. Bonet, G. Francès, and H Geffner. Learning features and abstract actions for computing generalized plans. In *Proc. AAAI 2019*, pages 2703–2710, 2019.

[Bonet, 2013] Blai Bonet. An admissible heuristic for SAS^+ planning obtained from the state equation. In *Proc. IJCAI 2013*, pages 2268–2274, 2013.

[Breiman *et al.*, 1984] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

[Cserna *et al.*, 2018] Bence Cserna, William Doyle, Jordan Ramsdell, and Wheeler Ruml. Avoiding dead ends in real-time heuristic search. In *Proc. AAAI 2018*, 2018.

[Eriksson *et al.*, 2017] Salomé Eriksson, Gabriele Röger, and Malte Helmert. Unsolvability certificates for classical planning. In *Proc. ICAPS 2017*, pages 88–97, 2017.

[Fern *et al.*, 2006] Alan Fern, Sungwook Yoon, and Robert Givan. Approximate policy iteration with a policy language bias: Solving relational markov decision processes. *J. Artif. Intell. Res.*, 25:75–118, 2006.

[Francès *et al.*, 2019] Guillem Francès, Augusto B. Corrêa, Cedric Geissmann, and Florian Pommerening. Generalized potential heuristics for classical planning. In *Proc. IJCAI 2019*, pages 5554–5561, 2019.

[Francès *et al.*, 2021] Guillem Francès, Blai Bonet, and Hector Geffner. Learning general policies from small examples without supervision. In *Proc. AAAI 2021*, 2021.

[Haslum and Geffner, 2000] Patrik Haslum and Héctor Geffner. Admissible heuristics for optimal planning. In *Proc. AIPS 2000*, pages 140–149, 2000.

[Haslum and Geffner, 2001] Patrik Haslum and Héctor Geffner. Heuristic planning with time and resources. In *Proc. ECP 2001*, pages 107–112, 2001.

[Haslum *et al.*, 2019] Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise. *An Introduction to the Planning Domain Definition Language*, volume 13 of *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool, 2019.

[Helmert and Geffner, 2008] Malte Helmert and Héctor Geffner. Unifying the causal graph and additive heuristics. In *Proc. ICAPS 2008*, pages 140–147, 2008.

[Helmert, 2004] Malte Helmert. A planning heuristic based on causal graph analysis. In *Proc. ICAPS 2004*, pages 161–170, 2004.

[Hoffmann *et al.*, 2014] Jörg Hoffmann, Peter Kissmann, and Álvaro Torralba. "Distance"? Who cares? Tailoring merge-and-shrink heuristics to detect unsolvability. In *Proc. ECAI 2014*, pages 441–446, 2014.

[Junghanns and Schaeffer, 1998] Andreas Junghanns and Jonathan Schaeffer. Single-agent search in the presence of deadlocks. In *Proc. AAAI 1998*, pages 419–425, 1998.

[Khardon, 1999] Roni Khardon. Learning action strategies for planning domains. *Artif. Intell.*, 113(1-2):125–148, 1999.

[Kolobov *et al.*, 2012] Andrey Kolobov, Mausam, and Daniel Weld. A theory of goal-oriented MDPs with dead ends. In *Proc. UAI 2012*, 2012.

[Lipovetzky *et al.*, 2016] Nir Lipovetzky, Christian Muise, and Hector Geffner. Traps, invariants, and dead-ends. In *Proc. ICAPS 2016*, pages 211–215, 2016.

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

[Martins *et al.*, 2014] Ruben Martins, Vasco Manquinho, and Inês Lynce. Open-WBO: A modular MaxSAT solver. In *Proc. SAT 2014*, pages 438–445, 2014.

[Muise and Lipovetzky, 2016] Christian Muise and Nir Lipovetzky, editors. *Unsolvability International Planning Competition: Planner Abstracts*, 2016.

[Muise *et al.*, 2012] Christian J. Muise, Sheila A. McIlraith, and J. Christopher Beck. Improved non-deterministic planning by exploiting state relevance. In *Proc. ICAPS 2012*, pages 172–180, 2012.

[Pedregosa *et al.*, 2011] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.*, 12:2825–2830, 2011.

[Seipp *et al.*, 2016] Jendrik Seipp, Florian Pommerening, Silvan Sievers, Martin Wehrle, Chris Fawcett, and Yusra Alkhazraji. Fast Downward Aidos. In *Unsolvability IPC: planner abstracts*, pages 28–38, 2016.

[Ståhlberg, 2017] Simon Ståhlberg. Tailoring pattern databases for unsolvable planning instances. In *Proc. ICAPS 2017*, pages 274–282, 2017.

[Steinmetz and Hoffmann, 2017] Marcel Steinmetz and Jörg Hoffmann. State space search nogood learning: Online refinement of critical-path dead-end detectors in planning. *AIJ*, 245:1–37, 2017.

[Torralba *et al.*, 2016] Álvaro Torralba, Jörg Hoffmann, and Peter Kissmann. MS-Unsat and SimulationDominance: Merge-and-Shrink and dominance pruning for proving unsolvability. In *Unsolvability IPC: planner abstracts*, pages 12–15, 2016.

[Torralba, 2016] Álvaro Torralba. Sympa: Symbolic perimeter abstractions for proving unsolvability. In *Unsolvability IPC: planner abstracts*, pages 8–11, 2016.