Learning General Policies and Helpful Action Classifiers from Partial State Spaces

Dominik Drexler[1] , **Javier Segovia-Aguas**[2] and **Jendrik Seipp**[1]
[1]Linköping University, Linköping, Sweden
[2]Universitat Pompeu Fabra, Barcelona, Spain
dominik.drexler@liu.se, javier.segovia@upf.edu, jendrik.seipp@liu.se

Abstract

Generalized planning aims to compute generalized plans that solve a whole class of problems from a given tractable planning domain. Recently, the D2L system showed how to learn generalized plans with the form of general policies in a self-supervised manner with a MaxSAT solver, where states and transitions are qualitatively abstracted by a set of description logics features. However, D2L requires to fully explore the state space of the input planning problems, which is a major bottleneck even for simple domains. Therefore, we propose the Incremental-D2L algorithm that only requires to explore small fragments of the input state spaces and show that it scales to harder training instances. For very hard domains, where we are unable to learn a general policy, Incremental-D2L yields a *partial* policy that we can use to enhance a greedy best-first search. Our experiments show that preferring learned *helpful actions*, i.e., actions compatible with the (partial) policy, significantly reduces the search effort for many of the considered domains.

1 Introduction

Classical planning is one of the original core AI areas. It is the task of finding a sequence of deterministic actions that transforms an initial fully-observed situation into one that satisfies a given goal condition [Ghallab *et al.*, 2004]. Traditional classical planning systems are domain-independent, i.e., they operate on a model of the task without knowing anything about the planning domain the task comes from. While domain-independent planners have certainly seen great success in recent years, they fail to scale to large and complex tasks.

In many cases, this limitation can be overcome by giving planners access to *domain control knowledge* (DCK), which allows them to better guide their search for a solution or even prune fragments of the search space. The planning literature contains many types of DCK and numerous ways of how to generate and use it: macros [Fikes *et al.*, 1972], control rules [Veloso *et al.*, 1995], temporal logic formulas [Bacchus and Kabanza, 2000; De Giacomo and Vardi, 2013], HTNs [Erol *et al.*, 1996], reactive policies [Yoon *et al.*, 2008; de la Rosa *et al.*, 2011], procedural DCK [Baier *et al.*, 2007; Segovia *et al.*, 2016], (hierarchical) finite state automata [Palacios and Geffner, 2009; Segovia-Aguas *et al.*, 2018], planning programs [Segovia-Aguas *et al.*, 2019; Segovia-Aguas *et al.*, 2021; Segovia-Aguas *et al.*, 2022], heuristics [Francès *et al.*, 2019; Ståhlberg *et al.*, 2021], general policies [Toyer *et al.*, 2018; Francès *et al.*, 2021], and policy sketches [Drexler *et al.*, 2021; Drexler *et al.*, 2022].

In this paper, we learn DCK in the form of general policies. Recently, Francès *et al.* [2021] presented the D2L system and showed that it can learn general policies in a self-supervised manner with a weighted MaxSAT solver, after abstracting the state space with a pool of description logic features. However, D2L requires to fully explore the state space of the input planning problems, which is a major bottleneck even for simple domains. To address this problem, we introduce the Incremental-D2L (ID2L) algorithm that learns policies from small fragments of the input state spaces, scaling up to larger input training instances and to more complex domains. ID2L iteratively increases the fragment of the input state space by sampling optimal plans and computing partial policies that work for the considered fragment of the state space.

For tractable domains in which the observed states and transitions are representative of the whole state space, ID2L converges to a general policy that encodes a solution for all tasks from the domain. While D2L is only applicable for tractable domains, ID2L yields a *partial* policy for intractable domains where no general policy exists, e.g., the PSPACE-complete Sokoban domain. We compare two approaches for using such (partial) policies to complement a greedy best-first search [Doran and Michie, 1966] and show that they often drastically reduce the search effort compared to a pure heuristic-based search.

The first approach is based on an algorithm by Yoon *et al.* [2008]: when GBFS expands a node, we perform a *policy lookahead* by repeatedly choosing a transition that is compatible with the learned policy and enqueue the successors of all states on the path. The second approach prioritizes states reached via policy-compatible transitions by putting them into an additional open list queue. GBFS then alternates between the two queues when selecting the next state to expand. This *dual-queue* strategy has been popularized by the LAMA planner [Richter and Westphal, 2010], which uses it to prioritize states reached via *preferred operators* (originally called *helpful actions*) [Hoffmann and Nebel, 2001].

2 Background

We begin by introducing classical planning, description logics, and their combination for generalized planning where we closely follow the definitions given by Francès *et al.*; Drexler *et al.* [2021; 2022].

2.1 Classical Planning

A *classical planning problem* or *task* is a pair $P = (\mathcal{D}, \mathcal{I})$ where $\mathcal{D}$ is a first-order *domain* with action schemas defined over predicates, and $\mathcal{I}$ is an instance consisting of a set of objects and two sets of ground literals, the initial and goal situations I and G. The initial situation is consistent and complete, meaning either a ground literal or its complement is in I. A task P induces a state model $S(P) = (S, I, S^G, A, app, succ)$ where the states in S are the truth valuations over the ground atoms represented by the set of literals that they make true, the initial state s_0 is I, the set of goal states S^G are those that make the goal literals in G true, and the actions A are the ground actions obtained from the schemas and objects. The ground actions in $app(s)$ are the ones that are applicable in a state s; namely, those whose preconditions are true in s, and the state transition function maps a state s and an action $a \in app(s)$ into the successor state $s' = succ(a, s)$. We say that $\langle s, a, s' \rangle$ is a *transition* if $s' = succ(a, s)$ and refer to the set of all transitions with T. An *s-plan* Π for P is a sequence of actions $a_0, \ldots, a_n$ that is executable in s and maps the state s into a goal state; i.e., $a_i \in app(s_i)$, $s_{i+1} = succ(a_i, s_i)$ for all $0 \leq i \leq n$, $s = s_0$, and $s_{n+1} \in S^G$. A state s is *solvable* if there exists an s-plan, otherwise it is *unsolvable* (also called *dead-end*), and a state s is *alive* if it is solvable and it is not a goal state. The *length* of a plan is the number of its actions, and a plan is *optimal* if there is no shorter plan. Our objective is to find suboptimal I-plans for *collections* of tasks $P = (\mathcal{D}, \mathcal{I})$ over fixed domains $\mathcal{D}$ denoted as $\mathcal{Q}_\mathcal{D}$ or simply as $\mathcal{Q}$.

2.2 Heuristics and Greedy Best-First Search

A *heuristic* is a function $h : S \to \mathbb{R}_0^+ \cup \{\infty\}$ that estimates the cost of an s-plan. In this work, we consider the blind heuristic h^{b}, which returns 1 for all non-goal states and 0 for all goal states, and the h^{FF} heuristic which is based on *delete relaxation* [Hoffmann and Nebel, 2001]. The main satisficing heuristic search algorithm is *greedy best-first search* [Doran and Michie, 1966], where the search repeatedly expands a search node with the lowest heuristic value among all previously generated but unexpanded search nodes, starting with the initial state I.

2.3 State Abstractions

A *state feature* is a function that maps a state s of a task P to either a truth value (Boolean feature) or a nonnegative integer (numerical feature). A *state abstraction* $f_\Phi(s)$ of a state $s \in S$ over features $\Phi = \{f_1, \ldots, f_n\}$ is a vector of feature valuations, i.e., $f_\Phi(s) = (f_1(s), \ldots, f_n(s))$. We construct features by iteratively applying grammar rules from description logics [Francès *et al.*, 2021].

Description logics [Baader *et al.*, 2003] is based one concepts and roles, where concepts represent unary relations, and roles represent binary relations over the universe Δ, that is, the set of objects in a task. The concepts and roles we use are similar to the ones used in work on computing general policies [Francès *et al.*, 2021]. However, we compute a richer pool of primitives by projecting n-ary predicates to concepts and roles, which often allows deriving meaningful information in domains with predicates of arity three or greater. We use the same feature grammar as in work on learning policy sketches [Drexler *et al.*, 2022], and hence we closely follow their definitions. Consider a state s in a task P. For every n-ary predicate p in P and $1 \leq i \leq n$, there is a concept that evaluates to the set of objects that occur at the i-th position of the respective ground atoms of p that are true in s. Similarly, for every n-ary predicate p in P and $1 \leq i < j \leq n$, there is a role that evaluates to the set of pairs of objects that occur at the i-th and j-th position of the respective ground atoms of p that are true in s. Furthermore, for every such primitive concept and role as above, there is a goal version that is evaluated in the goal instead of the state s. Next, for each constant x in the planning domain, there is a concept that evaluates to the singleton set $\{x\}$. Last, there are the following additional primitive and compositional concepts and roles. Consider concepts C, D, and roles R, S.

- *universal concept* $\top$ with denotation $\top^s \equiv \Delta$,
- *bottom concept* $\bot$ with denotation $\bot^s \equiv \emptyset$,
- *role-value mapping* $R = S$ with denotation $(R = S)^s \equiv \{a \in \Delta \mid (a, b) \in R^s \leftrightarrow (a, b) \in S^s\}$,
- *concept intersection* $C \sqcap D$ with denotation $(C \sqcap D)^s = C^s \cap D^s$,
- *concept negation* $\neg C$ with denotation $(\neg C)^s \equiv \Delta \setminus C^s$,
- *existential abstraction* $\exists R.C$ with denotation $(\exists R.C)^s \equiv \{a \in \Delta \mid \exists b : (a, b) \in R^s \wedge b \in C^s\}$,
- *universal abstraction* $\forall R.C$ with denotation $(\forall R.C)^s \equiv \{a \in \Delta \mid \forall b : (a, b) \in R^s \to b \in C^s\}$,
- *role inverse* R^{-1} with denotation $(R^{-1})^s \equiv \{(b, a) \mid (a, b) \in R^s\}$,
- *role restriction* $R : C$ with denotation $(R : C)^s \equiv R^s \sqcap (\Delta \times C^s)$,
- *role composition* $R \circ S$ with denotation $(R \circ S)^s \equiv \{(a, c) \in \Delta \times \Delta \mid (a, b) \in R^s \wedge (b, c) \in S^s\}$,
- *transitive closure role* R^+ with denotation $(R^+)^s \equiv \bigcup_{n \geq 1} (R^s)^n$ where the iterated composition is defined as $(R^s)^0 = \{(d, d) \mid d \in C^s\}$ and $(R^s)^{n+1} = (R^s)^n \circ R^s$.

We place additional restrictions on the above grammar. Similar to Francès *et al.* [2021], we omit role composition, and only allow the transitive closure role, inverse role, and restrict role on primitive concepts and roles. We obtain Boolean and numerical features via an additional level of composition. Consider a state s in a task P. For every nullary predicate p, there is a Boolean feature that is true in s iff the ground atom of p is true in s. Next, let C, D be concepts, R be a role, and X be either a concept or a role. There are the following Boolean and Numerical features.

- *Boolean empty feature* $Empty(X)^s$ is true iff $|X^s| = \emptyset$,
- *numerical counting feature* $Count(X)^s \equiv |X^s|$, and
- *numerical distance feature* $Distance(C, R, D)$, which returns the smallest $n \in \mathbb{N}_0$, s.t., there are objects $x_0, \dots, x_n$ in P, $x_0 \in C^s, x_n \in D^s$, and $(x_{i-1}, x_i) \in R^s$ for all $1 \leq i \leq n$. If no such n exists then the feature evaluates to ∞.

Our distance features are richer than those used by Francès *et al.* [2021] as we allow arbitrary concepts C in them, and R can have at most complexity 2.

The *feature complexity* is the number of grammar rules applied. Features with the smallest feature complexity of 1 are either primitive *concepts* corresponding to the unary projection of domain predicates or primitive *roles* corresponding to the binary projection of domain predicates. Our feature grammar ensures that every feature is computable in cubic time in the number of atoms. However, most features have linear or quadratic runtimes.

2.4 General Policies

The objective in *generalized planning* is finding knowledge suitable for solving a (possibly infinite) set of classical planning problems $\mathcal{Q} = \{P_i \mid i \geq 1\}$ that share a common planning domain [Jiménez *et al.*, 2019]. One type of solution to $\mathcal{Q}$ are *general policies*.

The language of general policies that we use in this paper closely follows the definition by Francès *et al.* [2021]. A *Boolean feature condition* is b or $\neg b$ if b is a Boolean feature and $n = 0$ or $n > 0$ if n is a numerical feature. A *feature effect* is b (positive), $\neg b$ (negative) or $b?$ (any) if b is a Boolean feature and $n\uparrow$ (increases), $n\downarrow$ (decreases) or $n?$ (any) if n is a numerical feature.

Definition 1 (General policy)**.** *A* policy π_Φ *is a set of* policy rules *over the features Φ. Each rule has the form $C \mapsto E$ where C is a set of Boolean feature conditions, and E is a set of feature effects. A transition $\langle s, a, s'\rangle$ is* compatible *with a policy π_Φ (or π_Φ-compatible) iff π_Φ contains a rule $C \mapsto E$ such that:*

- *C is true in $f_\Phi(s)$,*
- *$b(s')$ for all positive Boolean effects b,*
- *$\neg b(s')$ for all negative Boolean effects $\neg b$,*
- *$n(s) < n(s')$ for all increasing numerical effects $n\uparrow$,*
- *$n(s) > n(s')$ for all decreasing numerical effects $n\downarrow$, and*
- *$f(s) = f(s')$ for all features f that are not in E.*

Effects $b?$ or $n?$ allow b and n to change arbitrarily.

The policy is general *iff from every alive state there is a π_Φ-compatible transition, it is terminating and always reaches a goal state.*

Example 1. *In the Visitall domain, an agent starts at an arbitrary location of an $n \times n$ grid, and has to visit all locations. A general policy π_Φ for this domain requires two features $\Phi = \{v, d\}$, where v denotes the number of visited locations and d is the distance from the current agent location to the closest unvisited location. The two policy rules are $\{d > 0\} \mapsto \{v\uparrow, d?\}$ and $\{d > 0\} \mapsto \{d\downarrow\}$ which let the agent take an action that visits a new location or decreases the distance to the closest unvisited location.*

Definition 2 (Partial general policy)**.** *A policy π_Φ is partial if there is an alive source state from which no outgoing transition is* π_Φ-compatible.

3 General Policies as Transition Classifiers

A general policy π_Φ is a transition classifier because transitions are either π_Φ-compatible (*Good*), or not π_Φ-compatible (*Bad*). More formally, a δ-optimal binary transition classifier is a function $c_\delta : T \to \{Good, Bad\}$ that maps transitions $t = \langle s, a, s'\rangle \in T$ to either *Good* or *Bad* such that the following condition holds: if $c_\delta(t)$ is *Good* then s is alive, s' is solvable, and action a starts a δ-optimal s-plan, or otherwise, if it is *Bad* then either there is no δ-optimal s-plan or there is an alternative π_Φ-compatible transition from s. Thus, a δ-optimal general policy π_Φ acts as a c_δ transition classifier. We can use transition classifiers and thus general policies to characterize *helpful actions*.

Definition 3 (π_Φ-helpful Action)**.** *Given a general policy π_Φ and a state s, an action a is considered to be π_Φ-helpful iff there is a transition $\langle s, a, s'\rangle \in T$ that is π_Φ-compatible.*

We borrow the term π_Φ-helpful from Hoffmann and Nebel [2001] who define the helpful actions in a state s as the set of actions that are applicable in s and part of a relaxed s-plan.

Example 2. *Consider the general policy π_Φ from Example 1 and the following Visitall state: the grid has size 2×2, the agent is in the bottom right location, and only the top right location is unvisited. The agent can either move up or left. Moving up is π_Φ-helpful because it is compatible with both rules of the policy: the agent visits a new location and decreases the distance to an unvisited location. Moving left is not π_Φ-helpful because the transition is incompatible with the policy.*

4 Learning Helpful Actions

In this section, we show that general policies and thus transition classifiers can be incrementally learned and refined with a weighted MaxSAT solver.

4.1 Learning a Transition Classifier

Learning a binary classifier that labels actions as helpful or unhelpful can be turned into learning a general policy π_Φ as per Definition 3, since there is a one-to-one correspondence from *Good* transitions to *helpful actions*, and from *Bad* transitions to *unhelpful actions*.

The `D2L` system already showed that learning a general policy π_Φ can be cast as a self-supervised problem and formulated as a propositional theory in conjunctive normal form (CNF). However, `D2L` requires all reachable states from the input planning tasks. Thus, the approach is only tractable for simple domains and tiny input tasks. Even for small tasks, any stage in the `D2L` pipeline can become infeasible to compute: the preprocessing, the encoding to CNF, or the MaxSAT solving.

We avoid the shortcomings by expanding only a fragment $\mathcal{S}_i \subseteq S_i$ of states, which we obtain by sampling optimal plans for each task P_i in a given set of training tasks $\mathcal{P} \subseteq \mathcal{Q}$. Furthermore, with $\mathcal{G}_i \subseteq S_i$, we refer to the set of generated states that consists of all expanded states and their successors, inducing the set of sampled transitions $\mathcal{T}_i$ and optimal cost to goal $V_i^*(s)$ for each $s \in \mathcal{G}_i$. Like D2L, we compute a pool of features $\mathcal{F}$ up to a certain complexity k (see Section 2.3) to abstract states and transitions, and allow the policy π_Φ with $\Phi \subseteq \mathcal{F}$ to compute δ-optimal plans from any source state in the set of states $\mathcal{S}_i$. Thus, our propositional encoding $\Gamma = \mathit{CNF}(\{\mathcal{S}_i, V_i^*\}_{i=1}^n, \mathcal{P}, \mathcal{F}, \delta)$ uses the same types of propositional variables as D2L but some constraints are adapted for the expanded fragments. The propositional variables in Γ are:

- $\mathit{Select}(f)$: feature $f \in \mathcal{F}$ is used in the policy π_Φ,
- $\mathit{Good}_i(s, s')$: transition $\langle s, a, s'\rangle \in \mathcal{T}_i$ is π_Φ-compatible,
- $V_i(s, d)$: goal distance d is labeled δ-optimal for s in P_i.

We now describe the formulas in the theory Γ. In the constraints for each $P_i \in \mathcal{P}$, the symbols s, t range over all expanded states $\mathcal{S}_i$, and s', t' range over all generated states $\mathcal{G}_i$. $\Delta_f(s, s')$ expresses how the feature f changes from state s to s', i.e., from false to true ($\uparrow$), from true to false ($\downarrow$), increases ($\Uparrow$), decreases ($\Downarrow$), or stays the same ($=$). $V_i^*(s)$ denotes the cost of an optimal s-plan.

C1 For each state s, there exists at least one good transition: $\bigvee_{s'} \mathit{Good}_i(s, s')$.

C2 For good transitions, require that the assigned $V_i(s, d)$ labels are descending: $\mathit{Good}_i(s, s') \wedge V_i(s, d) \rightarrow \bigvee_{d' \in [V_i^*(s'), min(d-1, \delta V_i^*(s'))]} V_i(s', d')$.

C3 For each solvable state s, there is exactly one δ-optimal label: $|\{V_i(s, d) : d \in [V_i^*(s), \delta V_i^*(s)]\}| = 1$.

C4 Each transition $\langle s, a, s'\rangle \in T_i$ cannot be *Good* if s' is labeled with the maximum V_i value, i.e., $V_i(s', d')$ where $d' = \delta \max_s V_i^*(s)$.

C5 Each pair of good and bad transition must be distinguishable by at least one feature: $\mathit{Good}_i(s, s') \wedge \neg\mathit{Good}_i(t, t') \rightarrow \bigvee_{f \in \mathcal{F}: \Delta_f(s,s') \neq \Delta_f(t,t')} \mathit{Select}(f)$.

C6 At least one feature must be selected: $\bigvee_{f \in \mathcal{F}} \mathit{Select}(f)$.

C7 Each transition $\langle s, a, s'\rangle \in T_i$ to an unsolvable state s' is bad: $(A \wedge B) \rightarrow C$ where $A \equiv V_i^*(s) > 0$, $B \equiv V_i^*(s') = \infty$, and $C \equiv \neg\mathit{Good}_i(s, s')$.

C8 Minimize the total cost of Γ, i.e., satisfying $\mathit{Select}(f)$ has a cost equal to the *feature complexity* of f.

The propositional theory Γ differs from the one used by D2L in the following constraints: C1 specifically defines policies over the set of *expanded* states, not all states; C4 simplifies the decision task of labeling good transitions by not forcing the policy to move towards the states that are the furthest from goals in each input graph, i.e., any transition to one of the following states $\{argmax_{s \in \mathcal{G}_i} V_i^*(s)\}$ is bad; and C6 forbids empty policies which occur when all transitions in the observed fragment of the state space are descending and thus, any action will get closer to a goal without using any feature. Furthermore, we do not require the solver to find a solution with at least one feature that distinguishes all goal states from all non-goal states. Instead, we allow combinations of features, i.e., a conjunction or disjunction of features to be goal distinguishing. Next, we state that considering fragments of the state space can only increase the solution space.

Lemma 1. *Consider tasks $\mathcal{P}$, a pool of features $\mathcal{F}$, $\delta \in \mathbb{R}_{\geq 1}$, and for each $P_i \in \mathcal{P}$ the set of fully expanded states $\mathcal{S}_i \subseteq S_i$, with their corresponding cost to goal $V_i^*(s)$ for all $s \in \mathcal{S}_i$. If the theory $\Gamma_{full} = CNF(\{S_i, V_i^*\}_{i=1}^n, \mathcal{P}, \mathcal{F}, \delta)$ is satisfiable then the theory $\Gamma_{partial} = CNF(\{\mathcal{S}_i, V_i^*\}_{i=1}^n, \mathcal{P}, \mathcal{F}, \delta)$ is satisfiable.*

Proof. By assumption, $\Gamma_{\text{full}} = \mathit{CNF}(\{S_i, V_i^*\}_{i=1}^n, \mathcal{P}, \mathcal{F}, \delta)$ is satisfiable. Since, $\mathcal{S}_i \subseteq S_i$ and $\mathcal{T}_i \subseteq T_i$, we can remove the clauses from Γ_{full} containing $V_i(s, d)$ variables s.t. $s \notin \mathcal{G}_i$, and variables $\mathit{Good}_i(s, s')$ s.t. $(s, s') \notin \mathcal{T}_i$ to obtain a solution for $\Gamma_{\text{partial}} = \mathit{CNF}(\{\mathcal{S}_i, V_i^*\}_{i=1}^n, \mathcal{P}, \mathcal{F}, \delta)$. The remaining clauses and variable assignments in Γ_{partial} will remain satisfiable. □

4.2 Incremental D2L

Models that satisfy and optimize the D2L propositional theory for a given domain and collection of tasks are perfect transition classifiers since they observe all the corner cases of the input state spaces and guarantee sound δ-optimal plans from any expanded state. The downside of knowing about all corner cases is that all state spaces have to be fully explored, which is only feasible for very simple domains and small tasks. To mitigate this limitation while preserving the useful properties of D2L, we propose the Incremental-D2L (ID2L) algorithm to incrementally explore state spaces, taking advantage of the new propositional theory formulation Γ, which allows computing general policies over the *expanded* state space instead of the *full* state space. The solution preservation in $\mathcal{S}_i$ and $\mathcal{T}_i$ is proven in Lemma 1.

ID2L (Algorithm 1) is divided into three phases that are repeated until converging to a general policy π_Φ or until a time limit is reached. The *data generation* phase (lines 6–16) computes optimal s-plans for each root state (starting with the set of initial states), adds the s-plans to the partially expanded state space, assigns the optimal costs to each new expanded state and their immediate generated successors with their corresponding optimal cost to goal, and computes the pool of features $\mathcal{F}$ (introduced in Section 2.3) up to a certain complexity bound k over the generated states. Once the data is generated, the *learning* phase (lines 17–20), encodes the data into the propositional theory Γ, which is solved with a MaxSAT solver. If the solver proves the theory to be unsatisfiable, either the feature complexity bound k is too low, or no general policy exists, as is the case for all intractable domains. The *validation* phase (line 21) validates the computed policy π_Φ over all unexpanded states, which consists of randomly applying ground actions that are compatible with the policy, until reaching a goal condition or failing to do so. If the validation fails, it is because a new fragment of the state space has been observed where the policy either generates a cycle, or there is no π_Φ-compatible transition. Each failure can be understood as a flaw of the policy and can be used as one of the roots in the next iteration. The algorithm terminates when no more roots (or flaws) are detected.

Algorithm 1 Incremental-D2L

```
Input: set of planning tasks $\mathcal{P} = \{P_i\}_{i=1}^n$, slack factor $\delta$,
       maximum feature complexity $k$
Output: general policy $\pi_\Phi$
 1: roots $\leftarrow \{I_i\}_{i=1}^n$  // initial state of each $P_i \in \mathcal{P}$
 2: $\pi_\Phi \leftarrow \emptyset$
 3: for $i = 1$ to $n$ do
 4:     $\mathcal{S}_i, \mathcal{G}_i \leftarrow \emptyset, \emptyset$
 5: while roots $\neq \emptyset$ do
 6:     for all $s_i \in$ roots do
 7:         $\langle a_1, \ldots, a_m \rangle \leftarrow$Optimal-Planner($P_i[s_i]$)
 8:         for all $j = 1$ to $m$ do
 9:             $V_i^*(s_i) \leftarrow m - j + 1$   // distance to goal
10:             $\mathcal{S}_i \leftarrow \mathcal{S}_i \cup \{s_i\}$
11:             $\mathcal{G}_i \leftarrow \mathcal{G}_i \cup \{s_i\}$
12:             for all $\langle s_i, a, s_i' \rangle \in T_i$ do
13:                 $V_i^*(s_i') \leftarrow |$Optimal-Planner($P_i[s_i']$)$|$
14:                 $\mathcal{G}_i \leftarrow \mathcal{G}_i \cup \{s_i'\}$
15:             $s_i \leftarrow succ(a_j, s_i)$
16:     $\mathcal{F} \leftarrow$ Compute-Features($\{\mathcal{G}_i\}_{i=1}^n, k$)
17:     $\Gamma \leftarrow CNF(\{\mathcal{S}_i, V_i^*\}_{i=1}^n, \mathcal{P}, \mathcal{F}, \delta)$
18:     $\pi_\Phi \leftarrow$MaxSAT-Solver($\Gamma$)
19:     if $\pi_\Phi = \emptyset$ then
20:         return $\pi_\Phi$   // UNSAT theory
        // get alive states that are loopy or not $\pi_\Phi$-compatible
21:     roots $\leftarrow$ Validate($\pi_\Phi, \{\mathcal{G}_i \setminus \mathcal{S}_i\}_{i=1}^n$)
22: return $\pi_\Phi$
```

Theorem 1. *(Termination) The* `ID2L` *algorithm is terminating.*

Proof. By contradiction. Assume `ID2L` is non-terminating. This can only happen if the set of roots never becomes empty and the propositional theory Γ is always satisfiable. If Γ is satisfiable, then only unexpanded and alive states can be roots in the next iteration. In every iteration, the roots are expanded. Hence they cannot be roots in the next iteration, which monotonically enlarges the set of expanded states. Once all states in the given state spaces have been expanded, a general policy π_Φ exists because Γ is satisfiable. Since there are no more unexpanded states to validate, the new set of roots is empty, which contradicts the initial assumption, proving that `ID2L` is terminating. □

Lemma 2. *Policies returned by* `ID2L` *are δ-optimal.*

Proof. By contradiction. Assume that the policy π_Φ yields an s-plan with total cost d' which is not δ-optimal for a state $s \in \mathcal{S}_i$. Then at least one C3 clause must contain the variable $V_i(s, d')$. However, C3 is defined as $\{V_i(s,d) : d \in [V_i^*(s), \delta V_i^*(s)]\}$ for all $s \in \mathcal{S}_i$ but $d' > \delta V_i^*(s)$. Hence, $V_i(s, d')$ does not exist for any $s \in \mathcal{S}_i$, proving that π_Φ is δ-optimal in the set of expanded states. □

Theorem 2. *(Soundness) Let π_Φ be the policy returned by* `ID2L`*, where $\Phi \in \mathcal{F}$ is a non-empty set of features. Then, π_Φ is a solution to Γ.*

Proof. To show that π_Φ is a solution for Γ, we need to prove that every constraint from C1–C7 is satisfied in Γ.

- C1–C4 and C7. Since π_Φ is δ-optimal by Lemma 2, it is defined in every expanded state $s \in \mathcal{S}_i$, where the generated transitions (s, s') in the corresponding s-plan must be $Good_i(s, s')$ and descending $V_i(s,d) > V_i(s', d')$.
- C5. At least one feature in Φ must distinguish good from bad transitions, otherwise π_Φ would not be δ-optimal.
- C6. Since Φ is a non-empty set, at least one feature has been selected.

Hence, the policy π_Φ is a solution to Γ. □

Theorem 3. *(Completeness) If there is a δ-optimal policy π_Φ with $\Phi \subseteq \mathcal{F}$, then the* `ID2L` *algorithm will find it.*

Proof. The `ID2L` algorithm explores the state space incrementally, adding states and their expansions in every iteration if they are counter-examples of the theory, which is known to be terminating from Theorem 1. In each iteration `ID2L` considers a fragment of the state space $\mathcal{S}_i$ and sampled transitions $\mathcal{T}_i$. From Lemma 1 it follows that any π_Φ that is a solution in the full state space S and transitions T, is also a solution in the observed state space. Therefore, `ID2L` is terminating and guarantees that δ-optimal policies can be found in smaller observed fragments of the state space. □

Theorem 3 guarantees that if a δ-optimal policy exists `ID2L` will find it. However, if the returned policy π_Φ found correctly solves the observed state space fragment, `ID2L` will return it as a solution to the problem, even though there could be unobserved corner cases that are incompatible with the returned π_Φ. We named this phenomenon *early convergence*, which can be either provably mitigated after observing the full state space, or statistically by gathering more input tasks and sampling more data. Also, low δ values such as $\delta = 1$, which is the minimum value that forces the policy to be optimal from any state, could make the theory unsatisfiable, while larger values could turn it into a satisfiable theory by paying an extra cost in the number of variables and clauses in the encoding Γ. Thus, low δ values, intractable domains, resource limits (time and memory), and early convergence are the main reasons that cause a returned policy to be incomplete (a.k.a. partial policy) over the full state space when observing only a fragment.

5 Planning with a Helpful Actions Classifier

A binary transition classifier c that distinguishes between good and bad transitions can be beneficial for state space search. In the best case, where the classifier is perfect, we can prune all bad transitions and the search remains complete. However, in our case, where the learned policies are not necessarily perfect, we need to use algorithms capable of filling gaps of missing information. We use two methods for enhancing a greedy best-first search with a (partial) policy.

5.1 Policy Lookahead

The first approach, *policy lookahead*, is based on an algorithm by Yoon *et al.* [2008]: instead of adding only the successors of the expanded state to the open list like plain GBFS, their algorithm follows the learned policy for n steps and enqueues the successors of all states on the path. Since our learned partial policies do not always match exactly one of the outgoing transitions of the expanded state, we randomly select one of the policy-compatible transitions if there are multiple options. In case none of the outgoing transitions is compatible with the policy, we stop the lookahead.

5.2 Preferring Policy-Compatible Actions

Our second approach for enhancing a GBFS with a (partial) policy is to mark the executable actions that conform to the policy as *helpful actions*, and biasing GBFS towards executing helpful actions, which are also called *preferred operators* [Richter and Helmert, 2009]. Usually, preferred operators are a byproduct of the heuristic evaluation: for example, for a state s, the FF heuristic computes a relaxed s-plan (i.e., an s-plan for a copy of the task where all delete-effects are removed), and uses it to estimate the true goal distance of s [Hoffmann and Nebel, 2001]. Then FF marks all operators that are part of the delete-relaxed plan as preferred.

For brevity, we call states that are reached via preferred operators *preferred states*. In many domains, there are plans that only traverse states preferred by FF. In general, however, pruning all non-preferred operators renders the search incomplete. This is why preferred operators are usually used in a dual-queue approach, which prioritizes preferred states but does not prune any non-preferred states: one queue contains all open nodes (generated but not expanded), the other one contains only the nodes reached via preferred operators [Richter and Helmert, 2009]. GBFS then selects states from the two queues in an alternating manner. This setup makes it more likely that preferred states are expanded early. To further amplify this effect, Richter and Helmert [2009] use the following *boosting* method: whenever expanding a preferred state generates a successor state with a new minimum heuristic value, they only consider the preferred operator queue for the next N expansions.

6 Experiments

In this section, we evaluate ID2L on 23 common classical planning domains. The purpose of our experiments is two-fold: for tractable domains, we show that ID2L scales to larger tasks and more complicated domains than D2L. For intractable domains, where no general policy exists, we show that ID2L learns a partial policy that can be used as a helpful actions classifier, which enhances a GBFS. Our benchmark set consists of the 15 tractable domains Barman, Blocks, Blocks-clear, Blocks-on, Childsnack, Delivery, Depots, Ferry, Gripper, Miconic, Reward, Satellite, Spanner, Visitall and Zenotravel; and the eight intractable domains Driverlog, Freecell, Nomystery, N-Puzzle, Parking, Pipesworld-NoTankage, Pipesworld-Tankage and Sokoban.

6.1 Learning Policies with ID2L

Table 1 summarizes the results of learning policies with ID2L using A* and the LM-Cut heuristic [Helmert and Domshlak, 2009] from Fast Downward [Helmert, 2006] as the underlying optimal planner. The common training setting for all domains is $\delta = 2$ which forces policies to be 2-optimal over the training data, a time limit of 10h and a memory bound of 16 GiB. The maximum complexity k for computing features ranges from 5 to 9, depending on the domain (see k^* in Table 1). Note that $\mathcal{T}/\!\sim$ reports the total number of observed transitions T and the number of equivalence transition classes [Francès *et al.*, 2021], where two transitions belong to the same class iff they cannot be distinguished by the feature pool $\mathcal{F}$.

ID2L terminates in most tractable domains (10/15), from which the last learned policy is general in 9 domains. This is the case for Blocks, Blocks-clear, Blocks-on, Ferry, Gripper, Miconic, Reward, Spanner and Visitall. The reasons for failing to learn a general policy in a tractable domain are diverse: either k or δ are too low, or the training data is not representative enough of the state space. Modifying these parameters could allow usto learn general policies but these changes make it harder for optimal planners and MaxSAT solvers.

In the intractable domains, we cannot hope to find a general policy but D2L even fails to find a policy that solves the small set of training tasks. However, in these domains, it is still possible to compute a *partial policy* using ID2L. This shows the benefit of considering only fragments of the state spaces.

6.2 Planning With Transition Classifiers

Now, we analyze whether the learned policies capture useful information by using the learned policies in a GBFS. For the domains Barman, Depots, Driverlog, Freecell, Gripper, Miconic, Nomystery, Parking, Pipesworld-NoTankage, Pipesworld-Tankage, Satellite, Sokoban, Visitall, Zenotravel we use the Autoscale 20.10 tasks as our test set [Torralba *et al.*, 2021]. For the other domains, where no Autoscale tasks are available, we generated 30 sufficiently difficult tasks ourselves. For running the experiments, we use the Lab toolkit [Seipp *et al.*, 2017] on a compute cluster with Intel Xeon Gold and Ubuntu 20.04 LTS 64-bit. We set a runtime limit of 30 minutes, a 4 GB memory limit and use a total of eight different configurations. All configurations use GBFS and either the blind or the FF heuristic with boosting parameter $N = 1000$ and are all part of or were implemented into Fast Downward [Helmert, 2006].

Table 2 shows the number of solved tasks, and the expansion and runtime scores for the eight configurations. The expansion and runtime score [Richter and Helmert, 2009] are values between 0 and 1: The lowest score of 0 is reached with 10^6 expansions (resp. 30 minutes search time) or when the task is not solved. The highest score of 1 is reached with 10^2 or fewer expansions (resp. 1 second search time). Intermediate values are interpolated on a log-scale to account for the exponential scaling of problem difficulty.

Capturing Preferred Operators Information. The configuration blind$^\pi$ solves 14 more tasks than blind which does not use preferred operators (148 vs. 134 solved tasks), and

	$\lvert\mathcal{P}\rvert$	it.	$\mathcal{T}/\sim$	$\lvert\mathcal{F}\rvert$	t	c_Φ	$\lvert\Phi\rvert$	k^*	$\lvert\pi_\Phi\rvert$
Barman	1	5	–	–	–	14	5	4	30
Blocks	2	3	1374/763	894	1496	13	3	7	12
Blocks-clear	2	1	63/15	67	2	3	2	2	2
Blocks-on	1	5	228/90	249	7	9	3	4	16
Childsnack	3	8	–	–	–	21	6	7	43
Delivery	2	10	–	–	–	32	7	7	169
Depots	1	36	–	–	–	36	6	7	37
Driverlog	2	5	–	–	–	22	6	6	67
Ferry	12	2	789/531	900	172	13	3	7	8
Freecell	2	2	–	–	–	4	2	2	8
Gripper	1	2	420/44	78	8	6	2	4	5
Miconic	1	6	5254/817	153	1670	10	3	4	12
N-puzzle	3	14	–	–	–	57	8	9	141
Nomystery	1	7	–	–	–	41	8	7	68
Parking	1	6	–	–	–	21	5	5	16
Pipes-nt	1	5	–	–	–	19	4	7	41
Pipes-t	1	4	–	–	–	16	4	6	21
Reward	10	2	354/338	235	38	8	2	6	4
Satellite	2	4	–	–	–	26	8	4	51
Sokoban	1	2	–	–	–	31	5	9	46
Spanner	4	1	532/56	449	133	12	2	7	6
Visitall	1	2	114/29	29	2	7	2	5	4
Zenotravel	3	3	1540/852	382	561	30	7	6	39

Table 1: Overview of ID2L results. $\lvert\mathcal{P}\rvert$ is the size of the training set, "it." is the number of iterations to compute a policy that solves the training set, $\mathcal{T}/\sim$ is the number of observed transitions/distinguishable equivalence classes, $\lvert\mathcal{F}\rvert$ is the size of the feature pool, t is the total training wall-clock time in seconds, c_Φ is the optimal cost of the MaxSAT solution, $\lvert\Phi\rvert$ is number of selected features, k^* is the complexity of the most complex feature in the policy, $\lvert\pi_\Phi\rvert$ is the number of rules in the policy. We omit t, $\mathcal{T}/\sim$ and $\lvert\mathcal{F}\rvert$ when ID2L exceeds a 10h time limit.

achieves a slightly higher expansion score (0.13 vs. 0.08). Furthermore, the per-domain expansion score of blind$^\pi$ is always at least as high as the score of blind. Similarly, the configuration FF$^\pi$ solves 16 more tasks in comparison to FF which does not use preferred operators, with 414 and 398 solved tasks, respectively, and again has a slightly higher expansion score. This indicates that the learned policies capture useful information about preferred operators.

Policy Lookahead. We use the configuration FF$_\infty$ that performs lookaheads with $n = \infty$ to test whether it pays off to follow the policy greedily. If a policy is indeed a general policy, then this results in a single lookahead which is very fast to compute. This is the case in the domains Blocks, Blocks-clear, Blocks-on, Ferry, Gripper, Miconic, Reward, Spanner, and Visitall. If the policy is not a general policy, as is the case in Parking, then the policy lookahead can still pay off, as the increase in coverage from 9 to 18 in Parking shows. We also tested different values for the parameter n in the lookahead. Yoon *et al.* [2008] report that $n = 50$ yielded the best results. We compared this value to $n = \infty$, i.e., aborting the lookahead only if no operator is compatible with the policy, and found that the difference between the two versions is negligible for most domains. For Spanner and Visitall, however, $n = \infty$ solves many more tasks than $n = 50$, so we use $n = \infty$ for all lookaheads.

Preferring Policy-Compatible Operators. When comparing FF^r with FF$^\pi$, we observe that extracting preferred operators from relaxed plans usually works significantly better than from for our learned policies. However, there are several domains where FF$^\pi$ has a higher expansion score and in Spanner where FF^r fails to solve any task but FF$^\pi$ solves 27 tasks. Overall, FF^r solves 475 tasks and FF$^\pi$ solves 414 tasks. The reason for the difference could be that we may have stopped iterating in ID2L too early, the features are not sufficiently complex for finding a classifier, or the feature grammar is not sufficiently rich.

Preferred Operators With Policy Lookahead. In our last configuration FF$^r_\infty$, we combine the strongest methods from our experiments which are the preferred operators from relaxed plans and the policy lookahead. We obtain the highest number of solved tasks of 501, the highest expansion score of 0.62, and the highest runtime score of 0.58. The additional coverage is mostly due to the domains Spanner and Visitall.

7 Related Work

The idea of using fragments of the state space for learning policies was previously used in the context of generalized task and motion planning where state spaces are usually infinitely large due to variables having continuous domain. Curtis *et al.* [2022] discretize the variables to obtain a finite state space by sampling executable plans, i.e., poses and trajectories are collision free. The relevant part of the state space then consists only of states over facts that occur in these plans.

In contrast to filtering transitions, there is work on filtering subgoals. A subgoal is a promising state that must not necessarily be achieved in a single step. Baier *et al.* [2008] use the high-level Golog language for defining subgoals and their language representations are compiled directly into PDDL, making it possible to use off-the-shelf classical planners. Segovia *et al.* [2016] compute DCK as a curriculum of generalized planning tasks, where each solution is a planning program that can call as a procedure previously learned programs. Scala *et al.* [2020] use subgoaling-based decomposition techniques for numeric planning that safely relaxes those tasks for computing (in)admissible heuristics. Drexler *et al.* [2021] use the same language of general policies [Francès *et al.*, 2021] with different semantics for filtering subgoals and learn task decompositions automatically for tractable domains [Drexler *et al.*, 2022]. Other works aim at learning heuristics casting the problem as a black-box function optimization, i.e., loss function optimization [Orseau and Lelis, 2021], deep learning [Shen *et al.*, 2020; Karia and Srivastava, 2021], or a reinforcement learning problem [Gehring *et al.*, 2021]. The DCK learned in our works is simpler and easier to explain because our partial policies are representations of a target language with suitable semantics and much smaller than neural networks consisting of millions of parameters.

8 Acknowledgements

This work was partially supported by project TAILOR, funded by EU Horizon 2020 (grant agreement no. 952215),

	blind			blind$^{\pi}$			blind$_{\infty}$			FF			FF^{r}			FF$^{\pi}$			FF$_{\infty}$			FF$^{r}_{\infty}$		
	S	E	T	S	E	T	S	E	T	S	E	T	S	E	T	S	E	T	S	E	T	S	E	T
Barman (30)	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	6	0.05	0.08	**23**	**0.50**	**0.34**	8	0.06	0.09	7	0.08	0.11	20	0.39	0.28
Blocks (30)	2	0.02	0.07	2	0.06	0.07	**30**	**0.99**	**0.84**	26	0.85	0.48	26	0.86	0.48	26	0.86	0.47	24	0.80	0.44	26	0.86	0.43
Blocks-clear (30)	13	0.33	0.43	15	0.37	0.46	**30**	**1.00**	**1.00**	**30**	0.98	**1.00**	**30**	**1.00**	**1.00**	**30**	**1.00**	**1.00**	**30**	**1.00**	0.99	**30**	**1.00**	0.99
Blocks-on (30)	1	0.00	0.03	1	0.01	0.02	**30**	**1.00**	**1.00**	**30**	0.95	**1.00**	**30**	**1.00**	**1.00**	**30**	0.95	**1.00**	**30**	**1.00**	**1.00**	**30**	**1.00**	**1.00**
Childsnack (30)	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	1	0.00	0.01	7	0.12	0.16	3	0.02	0.03	1	0.02	0.03	**8**	**0.20**	**0.21**
Delivery (30)	**30**	0.43	0.95	**30**	0.78	0.91	**30**	0.78	0.91	**30**	0.99	**1.00**	**30**	**1.00**	**1.00**	**30**	**1.00**	**1.00**	**30**	**1.00**	**1.00**	**30**	**1.00**	**1.00**
Depots (30)	1	0.01	0.04	1	0.01	0.02	1	0.01	0.02	5	0.15	0.20	**6**	**0.18**	**0.21**	5	0.10	0.12	**6**	0.16	0.18	5	0.16	0.17
Driverlog (30)	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	8	0.13	0.16	**18**	**0.34**	**0.29**	4	0.06	0.07	6	0.10	0.10	7	0.14	0.13
Ferry (30)	10	0.09	0.29	19	0.32	0.45	**30**	**1.00**	**1.00**	**30**	**1.00**	**1.00**	**30**	**1.00**	**1.00**	**30**	**1.00**	**1.00**	**30**	**1.00**	**1.00**	**30**	**1.00**	**1.00**
Freecell (30)	9	0.03	0.20	16	0.12	0.31	16	0.27	0.40	27	0.49	0.63	26	**0.54**	**0.65**	26	0.40	0.57	26	0.48	0.62	**27**	0.52	**0.65**
Gripper (30)	0	0.00	0.00	0	0.00	0.00	**30**	**0.87**	**1.00**	**30**	0.61	0.81	**30**	0.68	0.80	**30**	0.54	0.57	**30**	**0.87**	0.94	**30**	**0.87**	0.94
Miconic (30)	0	0.00	0.00	0	0.00	0.00	**30**	**0.90**	0.85	**30**	**0.90**	0.97	**30**	**0.90**	**0.96**	**30**	**0.90**	0.76	**30**	**0.90**	0.84	**30**	**0.90**	0.84
N-puzzle (30)	20	0.21	0.67	20	0.37	0.62	20	0.37	0.62	**30**	0.88	**1.00**	**30**	0.87	**1.00**	**30**	**0.89**	**1.00**	**30**	0.88	**1.00**	**30**	0.87	**1.00**
Nomystery (30)	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	7	0.09	0.13	**10**	**0.22**	**0.26**	3	0.05	0.02	6	0.17	0.11	6	0.16	0.11
Parking (30)	0	0.00	0.00	0	0.00	0.00	2	0.04	0.06	9	0.21	0.18	11	0.29	0.25	6	0.10	0.08	**18**	**0.51**	**0.35**	17	0.49	**0.35**
Pipes-nt (30)	3	0.01	0.05	2	0.01	0.03	2	0.01	0.02	12	0.17	0.26	**24**	0.52	0.62	13	0.24	0.32	11	0.15	0.22	**24**	**0.63**	**0.71**
Pipes-t (30)	5	0.00	0.09	3	0.00	0.02	3	0.00	0.02	11	0.14	0.22	**28**	**0.65**	**0.63**	13	0.17	0.24	12	0.16	0.23	25	0.61	0.61
Reward (30)	26	0.57	0.85	29	0.79	0.83	**30**	**0.99**	**1.00**	**30**	0.92	**1.00**	**30**	0.94	**1.00**	**30**	**0.99**	0.99	**30**	**0.99**	**1.00**	**30**	**0.99**	**1.00**
Satellite (30)	1	0.01	0.05	1	0.01	0.02	1	0.01	0.02	10	0.33	0.36	**14**	**0.47**	**0.44**	10	0.31	0.34	10	0.33	0.36	13	0.43	0.42
Sokoban (30)	13	0.07	0.32	8	0.07	0.12	8	0.07	0.12	19	0.24	0.45	**22**	**0.26**	**0.46**	15	0.24	0.32	14	0.24	0.32	13	0.23	0.31
Spanner (30)	0	0.00	0.00	0	0.00	0.00	**30**	**0.87**	**0.91**	0	0.00	0.00	0	0.00	0.00	27	0.74	0.13	**30**	**0.87**	0.30	**30**	**0.87**	0.30
Visitall (30)	0	0.00	0.00	0	0.00	0.00	**25**	**0.59**	**0.44**	5	0.05	0.11	5	0.06	0.11	6	0.12	0.14	**25**	**0.59**	**0.44**	**25**	**0.59**	**0.44**
Zenotravel (30)	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	12	0.34	0.43	**15**	**0.49**	**0.67**	9	0.21	0.26	9	0.24	0.35	**15**	0.45	0.55
Sum (690)	134	0.08	0.18	148	0.13	0.17	348	0.42	0.44	398	0.45	0.50	475	0.56	**0.58**	414	0.48	0.46	445	0.54	0.52	**501**	**0.62**	**0.58**

Table 2: Number of solved tasks (S), expansion score (E), and runtime score (T) for different GBFS configurations. We highlight the maximum value of all configurations with boldface. The configuration superscripts indicate that preferred operators are used: superscript r indicates that an operator is preferred if it starts a relaxed plan and superscript π indicates that the configuration prefers policy-compatible operators. The subscript ∞ denotes that the configuration uses policy lookahead.

and by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The computations were enabled by resources provided by the Swedish National Infrastructure for Computing (SNIC), partially funded by the Swedish Research Council through grant agreement no. 2018-05973.

9 Conclusions

We introduced the ID2L algorithm for learning DCK in the form of a general policy that is refined by incrementally exploring fragments of the input state spaces that are non-compatible with the policy. In addition, we show that *policy lookaheads* and *preferring policy-compatible operators* are methods to take advantage of learned policies for planning even if the policies are not general, which is always the case for intractable domains. This is a promising line of research to deal with harder *tractable* planning domains for which the complexity can be studied with the computed policies. Also, the ID2L algorithm can be improved if any of its baselines are improved, e.g., more efficient computation of features, optimal plans, or policy validation. Regarding *intractable* domains, they could still contain tractable subproblems for which general policies can be learned and used as multiple transition classifiers for planning.

References

[aaa, 2021] *Proc. AAAI 2021*, 2021.

[Baader *et al.*, 2003] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[Bacchus and Kabanza, 2000] Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *AIJ*, 116(1–2):123–191, 2000.

[Baier *et al.*, 2007] Jorge A. Baier, Christian Fritz, and Sheila A. McIlraith. Exploiting procedural domain control knowledge in state-of-the-art planners. In *Proc. ICAPS 2007*, pages 26–33, 2007.

[Baier *et al.*, 2008] Jorge A. Baier, Christian Fritz, Meghyn Bienvenu, and Sheila A. McIlraith. Beyond classical planning: Procedural control knowledge and preferences in state-of-the-art planners. In *Proc. AAAI 2008*, pages 1509–1512, 2008.

[Curtis *et al.*, 2022] Aidan Curtis, Tom Silver, Joshua B. Tenenbaum, Tomas Lozano-Perez, and Leslie Pack Kaelbling. Discovering state and action abstractions for generalized task and motion planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 2022.

[De Giacomo and Vardi, 2013] Guiseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Proc. IJCAI 2013*, pages 854–860, 2013.

[de la Rosa *et al.*, 2011] Tomás de la Rosa, Sergio Jiménez, Raquel Fuentetaja, and Daniel Borrajo. Scaling up heuristic planning with relational decision trees. *JAIR*, 40:767–813, 2011.

[Doran and Michie, 1966] James E. Doran and Donald Michie. Experiments with the graph traverser program. *Proceedings of the Royal Society A*, 294:235–259, 1966.

[Drexler *et al.*, 2021] Dominik Drexler, Jendrik Seipp, and Hector Geffner. Expressing and exploiting the common subgoal structure of classical planning domains using sketches. In *Proc. KR 2021*, pages 258–268, 2021.

[Drexler *et al.*, 2022] Dominik Drexler, Jendrik Seipp, and Hector Geffner. Learning sketches for decomposing planning problems into subproblems of bounded width. In *Proc. ICAPS 2022*, 2022.

[Erol *et al.*, 1996] Kutluhan Erol, James A. Hendler, and Dana S. Nau. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence (AMAI)*, 18(1):69–93, 1996.

[Fikes *et al.*, 1972] Richard E Fikes, Peter E Hart, and Nils J Nilsson. Learning and executing generalized robot plans. *Artificial intelligence*, 3:251–288, 1972.

[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 planning policies from small examples without supervision. In *Proc. AAAI 2021* [2021], pages 11801–11808.

[Gehring *et al.*, 2021] Clement Gehring, Masataro Asai, Rohan Chitnis, Tom Silver, Leslie Pack Kaelbling, Shirin Sohrabi, and Michael Katz. Reinforcement learning for classical planning: Viewing heuristics as dense reward generators. *arXiv preprint arXiv:2109.14830*, 2021.

[Ghallab *et al.*, 2004] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[Helmert and Domshlak, 2009] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. ICAPS 2009* [2009], pages 162–169.

[Helmert, 2006] Malte Helmert. The Fast Downward planning system. *JAIR*, 26:191–246, 2006.

[Hoffmann and Nebel, 2001] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *JAIR*, 14:253–302, 2001.

[ica, 2009] *Proc. ICAPS 2009*, 2009.

[Jiménez *et al.*, 2019] Sergio Jiménez, Javier Segovia-Aguas, and Anders Jonsson. A review of generalized planning. *The Knowledge Engineering Review*, 34, 2019.

[Karia and Srivastava, 2021] Rushang Karia and Siddharth Srivastava. Learning generalized relational heuristic networks for model-agnostic planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 8064–8073, 2021.

[Orseau and Lelis, 2021] Laurent Orseau and Levi H. S. Lelis. Policy-guided heuristic search with guarantees. In *Proc. AAAI 2021* [2021], pages 12382–12390.

[Palacios and Geffner, 2009] Hector Palacios and Hector Geffner. Compiling uncertainty away in conformant planning problems with bounded width. *JAIR*, 35:623–675, 2009.

[Richter and Helmert, 2009] Silvia Richter and Malte Helmert. Preferred operators and deferred evaluation in satisficing planning. In *Proc. ICAPS 2009* [2009], pages 273–280.

[Richter and Westphal, 2010] Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR*, 39:127–177, 2010.

[Scala *et al.*, 2020] Enrico Scala, Patrik Haslum, Sylvie Thiébaux, and Miquel Ramirez. Subgoaling techniques for satisficing and optimal numeric planning. *Journal of Artificial Intelligence Research*, 68:691–752, 2020.

[Segovia-Aguas *et al.*, 2018] Javier Segovia-Aguas, Sergio Jiménez-Celorrio, and Anders Jonsson. Computing hierarchical finite state controllers with classical planning. *Journal of Artificial Intelligence Research*, 62:755–797, 2018.

[Segovia-Aguas *et al.*, 2019] Javier Segovia-Aguas, Sergio Jiménez, and Anders Jonsson. Computing programs for generalized planning using a classical planner. *Artificial Intelligence*, 272:52–85, 2019.

[Segovia-Aguas *et al.*, 2021] Javier Segovia-Aguas, Sergio Jiménez, and Anders Jonsson. Generalized planning as heuristic search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, pages 569–577, 2021.

[Segovia-Aguas *et al.*, 2022] Javier Segovia-Aguas, Sergio Jiménez, Anders Jonsson, and Laura Sebastiá. Scaling-up generalized planning as heuristic search with landmarks. *arXiv preprint arXiv:2205.04850*, 2022.

[Segovia *et al.*, 2016] Javier Segovia, Sergio Jiménez, and Anders Jonsson. Generalized planning with procedural domain control knowledge. In *Proc. ICAPS 2016*, pages 285–293, 2016.

[Seipp *et al.*, 2017] Jendrik Seipp, Florian Pommerening, Silvan Sievers, and Malte Helmert. Downward Lab. https://doi.org/10.5281/zenodo.790461, 2017.

[Shen *et al.*, 2020] William Shen, Felipe Trevizan, and Sylvie Thiébaux. Learning domain-independent planning heuristics with hypergraph networks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 574–584, 2020.

[Ståhlberg *et al.*, 2021] Simon Ståhlberg, Guillem Francès, and Jendrik Seipp. Learning generalized unsolvability heuristics for classical planning. In *Proc. IJCAI 2021*, pages 4175–4181, 2021.

[Torralba *et al.*, 2021] Álvaro Torralba, Jendrik Seipp, and Silvan Sievers. Automatic instance generation for classical planning. In *Proc. ICAPS 2021*, pages 376–384, 2021.

[Toyer *et al.*, 2018] Sam Toyer, Felipe Trevizan, Sylvie Thiébaux, and Lexing Xie. Action schema networks: Generalised policies with deep learning. In *Proc. AAAI 2018*, pages 6294–6301, 2018.

[Veloso *et al.*, 1995] Manuela Veloso, Jaime Carbonell, Alicia Perez, Daniel Borrajo, Eugene Fink, and Jim Blythe. Integrating planning and learning: The prodigy architecture. *Journal of Experimental & Theoretical Artificial Intelligence*, 7(1):81–120, 1995.

[Yoon *et al.*, 2008] Sungwook Yoon, Alan Fern, and Robert Givan. Learning control knowledge for forward search planning. *JMLR*, 9:683–718, 2008.