Generating SAS$^+$ Planning Tasks of Specified Causal Structure

Michael Katz, Junkyu Lee, Shirin Sohrabi

IBM T.J. Watson Research Center
1101 Kitchawan Rd, Yorktown Heights, NY 10598, USA
{michael.katz1, junkyu.lee}@ibm.com, ssohrab@us.ibm.com

Abstract

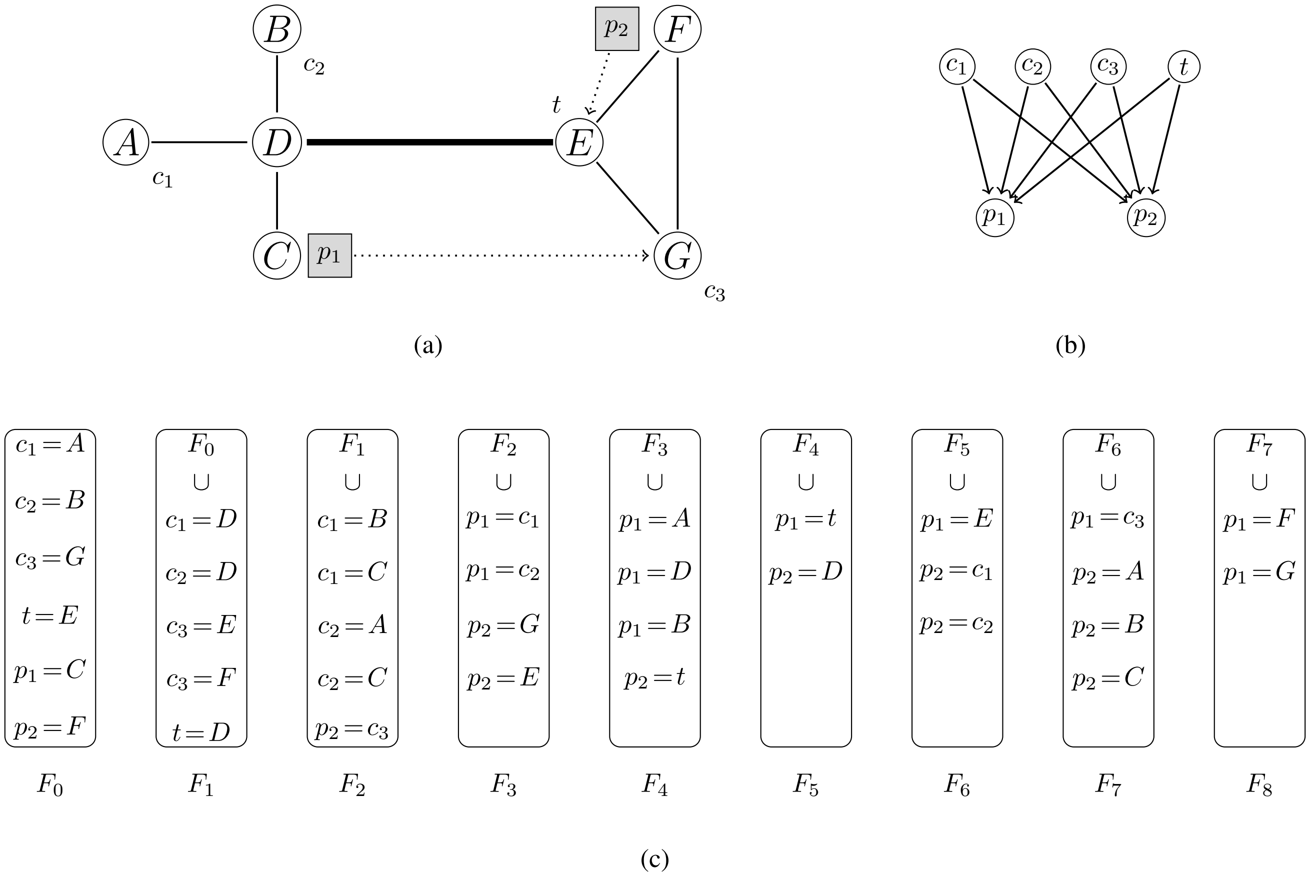
Recent advances in data-driven approaches in AI planning demand more and more planning tasks. The supply, however, is somewhat limited. Past International Planning Competitions (IPCs) have introduced the de-facto standard benchmarks with the domains written by domain experts. The few existing methods for sampling random planning tasks severely limit the resulting problem structure. In this work we show a method for generating planning tasks of any requested causal graph structure, alleviating the shortage in existing planning benchmarks. We present an algorithm for constructing random SAS$^+$ planning tasks given an arbitrary causal graph and offer random task generators for the well-explored causal graph structures in the planning literature. We further allow to generate a planning task equivalent in causal structure to an input SAS$^+$ planning task. We generate two benchmark sets: 26 collections for select well-explored causal graph structures and 42 collections for existing IPC domains. We evaluate both benchmark sets with the state-of-the-art optimal planners, showing the adequacy for adopting them as benchmarks in cost-optimal classical planning. The benchmark sets and the task generator code are publicly available at https://github.com/IBM/fdr-generator.

Introduction

Since the first encoded planning tasks in STRIPS (Fikes and Nilsson 1971), data has been the cornerstone and one of the main drivers of research in planning. The International Planning Competitions (IPC) are the primary source of benchmarks with slightly over 70 domains since the first IPC in 1998 (McDermott 2000). While most of the existing domains are hand-crafted, some correspond to machine translation from a different problem (e.g., Palacios and Geffner 2009; Bonet, Palacios, and Geffner 2009; Grastien and Scala 2018; Sohrabi et al. 2018).

A major focus in classical planning over the last two decades has been on heuristic search, with automatically obtained heuristics for planning tasks. These heuristics, either explicitly or implicitly, exploit the task structure in their computation. In most cases that task structure is the causal structure. A few examples include the *causal graph heuristic* (Helmert 2004) and the *structural pattern heuristics* (Katz and Domshlak 2010; Katz and Keyder 2012). The former ignores some of the causal graph edges to make it acyclic, while the latter abstracts the planning task to have particular causal graph structures that, together with some additional restrictions, make cost-optimal planning tractable. Other heuristic functions, such as *pattern databases* (Edelkamp 2001) exploit the causal information in e.g., pattern selection (Haslum et al. 2007). *Merge-and-shrink heuristics* (Helmert, Haslum, and Hoffmann 2007) use the causal graph for guiding the merge process. Most existing heuristics that work on the multi-valued representation exploit the causal information in one way or another. Further, starting with the seminal work of Bäckström and Nebel (1995), the research on the complexity of planning tasks had a major focus on the characterization of planning fragments by their causal graph structure (Domshlak and Dinitz 2001; Domshlak and Brafman 2002; Katz and Domshlak 2007, 2008a,b, 2010; Giménez and Jonsson 2008, 2009; Katz and Keyder 2012; Bäckström and Jonsson 2013; Aghighi, Jonsson, and Ståhlberg 2015; Bäckström, Jonsson, and Ordyniak 2019), as well as some local structural characteristics, such as *k-dependence* (Katz and Domshlak 2007, 2008a; Giménez and Jonsson 2012), classifying these fragments into a variety of complexity classes. For these two reasons, various planners' performance heavily relies on structural characteristics of the input planning task.

In this work, we aim to enable generating planning tasks of the requested structure. Here, we focus on tasks characterized by their causal graph structure. Namely, we propose a way of generating a SAS$^+$ task whose causal graph matches any graph over the multi-valued variables provided as an input. This approach automatically generates a diverse collection of planning tasks, as large as needed for various purposes such as learning a good planner selection strategy (Sievers et al. 2019), or offering an additional source of benchmarks for empirical evaluation. We empirically validate that our generated tasks are sufficiently challenging for the existing state-of-the-art classical cost-optimal planners and that their relative performance on these domains is somewhat different from the one on auto-generated IPC instances (Torralba, Seipp, and Sievers 2021).



B
c_2
p_2
F
t
A
D
E
c_1
C
p_1
G
c_3
(a)
c_1
c_2
c_3
t
p_1
p_2
(b)
$c_1 = A$, $c_2 = B$, $c_3 = G$, $t = E$, $p_1 = C$, $p_2 = F$
F_0 ∪ $c_1 = D$, $c_2 = D$, $c_3 = E$, $c_3 = F$, $t = D$
F_1 ∪ $c_1 = B$, $c_1 = C$, $c_2 = A$, $c_2 = C$, $p_2 = c_3$
F_2 ∪ $p_1 = c_1$, $p_1 = c_2$, $p_2 = G$, $p_2 = E$
F_3 ∪ $p_1 = A$, $p_1 = D$, $p_1 = B$, $p_2 = t$
F_4 ∪ $p_1 = t$, $p_2 = D$
F_5 ∪ $p_1 = E$, $p_2 = c_1$, $p_2 = c_2$
F_6 ∪ $p_1 = c_3$, $p_2 = A$, $p_2 = B$, $p_2 = C$
F_7 ∪ $p_1 = F$, $p_1 = G$
F_0
F_1
F_2
F_3
F_4
F_5
F_6
F_7
F_8
(c)

Figure 1: (a) A transportation planning example task, (b) its causal graph, and (c) the fact layers of the relaxed planning graph.

The rest of the paper is structured as follows. We first provide the necessary background and notation. Then, we describe the proposed planning task construction method and discuss some of the causal graph structures used. The experimental evaluation section describes the method used to generate a small set of instances challenging for state-of-the-art planners, as well as presents an evaluation of the generated collection with state-of-the-art cost-optimal planners. We conclude with the related work and discussion sections.

Preliminaries

A SAS$^+$ *planning task* (Bäckström and Nebel 1995) is given by a tuple $\langle \mathcal{V}, A, s_0, s_* \rangle$, where $\mathcal{V}$ is a finite set of *state variables* and A is a finite set of *actions*. Each state variable $v \in \mathcal{V}$ has a finite domain $dom(v)$. A *fact* is a pair $\langle v, \vartheta \rangle$ of variable $v \in \mathcal{V}$ and its value $\vartheta \in dom(v)$. By F_v we denote the set $\{\langle v, \vartheta \rangle \mid \vartheta \in dom(v)\}$ of facts for the variable v, and the set of all facts is denoted by $F := \bigcup_{v \in \mathcal{V}} F_v$. A (partial) assignment to the variables $\mathcal{V}$ is called a *(partial) state*. We view a partial state p as a set of facts with $\langle v, \vartheta \rangle \in p$ if and only if $p[v] = \vartheta$. For a partial state p, $\mathcal{V}(p) \subseteq \mathcal{V}$ denotes the subset of state variables instantiated by p. A partial state p is *consistent* with state s if $p \subseteq s$. We denote the set of states of a planning task by S. s_0 is the *initial state*, and the partial state s_* is the *goal*. Each *action* a is a pair $\langle \mathit{pre}(a), \mathit{eff}(a) \rangle$ of partial states called *preconditions* and *effects*. By $\mathit{prv}(a)$ we denote the part of the preconditions that corresponds to variables that do not participate in the action's effects, $\mathit{prv}(a) = \{\langle v, \vartheta \rangle \in \mathit{pre}(a) \mid v \notin \mathcal{V}(\mathit{eff}(a))\}$, also called *prevail conditions*. An *action cost* is a mapping $C : A \to \mathbb{R}^{0+}$. For simplicity, we assume that the preconditions are defined whenever the effects are defined, that is $\mathcal{V}(\mathit{eff}(a)) \subseteq \mathcal{V}(\mathit{pre}(a))$. An action a is applicable in a state $s \in S$ if and only if $\mathit{pre}(a) \subseteq s$. Applying a changes the value of $v \in \mathcal{V}(\mathit{eff}(a))$ to $\mathit{eff}(a)[v]$. The resulting state is denoted by $s[\![a]\!]$. An action sequence $\pi = \langle a_1, \dots, a_k \rangle$ is applicable in s if there exist states $s_0, \cdots, s_k$ such that (i) $s_0 = s$, and (ii) for each $1 \leq i \leq k$, a_i is applicable in s_{i-1} and $s_i = s_{i-1}[\![a_i]\!]$. We denote the state s_k by $s[\![\pi]\!]$. π is a plan iff π is applicable in s_0 and $s_* \subseteq s_0[\![\pi]\!]$. We denote by $\mathcal{P}(\Pi)$ (or just $\mathcal{P}$ when the task is clear from the context) the set of all plans of Π. The cost of a plan π, denoted by $C(\pi)$ is the summed cost of the actions in the plan.

To illustrate the definitions we use a transportation planning task example by Helmert (2006), depicted in Figure 1.

Example The task is to deliver parcel p_1 from C to G and parcel p_2 from F to E by the means of cars c_1, c_2, and c_3, as well as truck t. In SAS$^+$, it has state variables $\mathcal{V} = \{c_1, c_2, c_3, t, p_1, p_2\}$ with domains $dom(c_1) = dom(c_2) = \{A, B, C, D\}$, $dom(c_3) =$

$\{E, F, G\}$, $dom(t) = \{D, E\}$, and $dom(p_1) = dom(p_2) = \{A, B, C, D, E, F, G, c_1, c_2, c_3, t\}$. Actions are *drive*-$c$-$x$-$y$ for cars on thin edges (inner city roads) and *drive*-t-x-y for the truck on the thick edges (highways); *load*-p-v-x and *unload*-p-v-x for all parcels, vehicles, and locations. The action *drive*-c_1-A-D, for example, has preconditions $\{c_1 = A\}$ and effects $\{c_1 = D\}$ and the action *load*-p_1-c_1-C has preconditions $\{p_1 = C, c_1 = C\}$ and effects $\{p_1 = c_1\}$.

Causal Graph

A central role in what follows is played by *causal graphs* (Helmert 2004). The causal graph CG_Π of a task Π is a digraph with vertices $\mathcal{V}$. An arc (v, v') is in CG_Π iff $v \neq v'$ and there exists an action $a \in A$ such that $(v, v') \in [\mathcal{V}(\mathit{eff}(a)) \cup \mathcal{V}(\mathit{pre}(a))] \times \mathcal{V}(\mathit{eff}(a))$. The causal graph of the example task is shown in Figure 1 (b). The *move* actions do not contribute any edges, as these actions are preconditioned on the same variable they affect. All edges are contributed by the *load* and *unload* actions. Specifically, the action *load*-p_1-c_1-C contributes the edge from c_1 to p_1. For an action a, by E_a we denote the set of all such arcs, and by $E_{A'}$ we denote the union of all sets of arcs E_a for $a \in A'$.

Relaxed Planning Graph

Another structure typically used in planning for computing relaxation-based heuristics is *relaxed planning graph* (Hoffmann and Nebel 2001), a layered graph of facts and actions, describing action application in the planning task, under value accumulating semantics, i.e., delete-relaxation in the STRIPS formalism. The layers are added until a fixpoint is reached, i.e., until no new fact can be achieved. The first fact layer F_0 thus corresponds to the facts from the initial state, and the last layer is also a fact layer, and it is equal to the preceding fact layer. Each action layer A_i consists of all actions from A that are applicable in F_i, i.e., $A_i = \{a \in A \mid \mathit{pre}(a) \subseteq F_i\}$. The next fact layer F_{i+1} is then constructed by adding to F_i all facts achieved by the actions in A_i, namely $F_{i+1} = F_i \cup \bigcup_{a \in A_i} \mathit{eff}(a)$. Figure 1 (c) shows the fact layers for our example task, without repeating the last equivalent layer. As layers grow monotonically, we only show the newly added facts in each layer. For example, $F_1 = F_0 \cup \{c_1 = D, c_2 = D, c_3 = E, c_3 = F, t = D\}$, with $c_1 = D$ being the effect of the action *drive*-c_1-A-D, whose precondition is in F_0.

Translating STRIPS to SAS$^+$

Finally, the SAS$^+$ representation is often obtained by *translation* from the STRIPS representation (Helmert 2006). The multi-valued SAS$^+$ variables then correspond to invariant groups of pairwise mutually exclusive facts (mutexes), where exactly one such fact is true in any state reachable from the initial state. Each such invariant group over STRIPS facts corresponds to a set of facts *at most one* of which can be true in any reachable state. If there exist such states where no facts are true, then an additional value is added, representing that none of the facts in the invariant group is true. In our example task, the STRIPS representation includes mutually exclusive facts that represent various locations of vehicles, as well as locations/positions of parcels. Exactly one of these facts can be true in any reachable state.

Construction

The aim of this work is to generate planning tasks of a specific causal graph structure. Therefore, our algorithm will receive a graph $G = (\mathcal{V}, E)$ as its input and produce a SAS$^+$ planning task Π with $\mathit{CG}_\Pi = G$.

Planning Task Construction

Given a graph $G = (\mathcal{V}, E)$ and a number of facts $n \geq 2|\mathcal{V}|$, we construct the SAS$^+$ planning task $\Pi = \langle \mathcal{V}, A, s_0, s_* \rangle$ with the causal graph G as follows. First, we choose the domain size $d_v \geq 2$ for each multi-valued variable $v \in \mathcal{V}$ and assume without loss of generality the values to be $dom(v) = \{0, \dots, d_v - 1\}$. The variables represent sets of mutexes, each corresponds to one of the two types: exactly one or at most one of the values is true in all reachable states. We randomly decide which variables belong to which category. For the latter case, the last domain value corresponds to the case when none of the other facts are true. Next, without loss of generality, we assume $s_0[v] = 0$ for all $v \in \mathcal{V}$. Then, we construct the actions, in layers, while in parallel constructing the relaxed planning graph. Finally, the goal is chosen from the last fact layer of the relaxed planning graph, making sure that at most one fact is chosen per variable. We randomly decide whether at least one of the chosen facts is unique to the last fact layer. In what follows, we describe how actions are constructed. Starting with the initial state as the first fact layer F_0, we create actions for an action layer L_i by

(I) selecting a subset of facts from the fact layer F_i, ensuring at most one fact is selected per variable,

(II) partitioning the selected set of facts into prevail condition and non-prevail precondition, and

(III) choosing for all[1] the variables of the precondition facts a different value as its effect.

The constructed action a is checked against the graph G, ensuring that it contributes only edges that exist in G. If not, a is discarded. If a does not contribute any new edges and does not achieve new facts, we randomly decide whether to keep it.

The generic approach to action construction described above can be adapted to enforce particular properties. We discuss three such cases in detail.

(A) Enforce the action to achieve at least one new fact: ensure in steps (I) and (II) that the precondition includes facts for some variables $v \in \mathcal{V}$ that are not fully covered by the fact layer F_i (that is $F_v \setminus F_i \neq \emptyset$), and in step (III) to choose one of these facts $F_v \setminus F_i$. Note that

[1]While SAS$^+$ representation does not require to specify the precondition when the effect is specified, in order to ensure maintaining variables as mutexes of facts, we restrict ourselves here to always specifying the precondition in such cases.

this can be done without covering any new edges to the causal graph, if a single fact is chosen in step (I).

(B) Enforce the preconditions to include atoms from $F_i \setminus F_{i-1}$, enforced in step (I).

(C) Enforce covering another edge $\langle v, v' \rangle$ of G: ensure that $\langle v, \vartheta \rangle$ and $\langle v', \vartheta' \rangle$ are chosen in step (I), and in step (II) at most one of these facts is chosen for the prevail condition.

We randomly and independently decide whether to enforce the options (A)-(C) and whether to cover additional edges of the causal graph. Note that not all combinations are always possible. In such cases, an action is not constructed in that iteration. Each layer is constructed until a sufficient number of new facts $F_{i+1} \setminus F_i$ is added. The construction is stopped when all facts were achieved and all edges from G are reflected in the causal graph of the constructed planning task. The latter is enforced in the last layer. The goal is then randomly chosen from the last layer according to step (I), randomly deciding whether to ensure that at least one of the facts is not achieved before the last layer, analogously to how a precondition of an action is chosen when enforcing the option (B). Algorithm 1 describes the construction of a planning task from the given graph G, where the function CREATEACTION creates a single action, randomly choosing among the options described above. All random choices are made uniformly, with the decisions being made based on algorithm parameters. Given that, we can now show that the algorithm successfully terminates in polynomial time.

Theorem 1 *Given G and n, Algorithm 1 terminates in time polynomial in $|G|$ and n and returns a planning task with the causal graph G.*

Proof: The proof follows from the fact that in line 13 of Algorithm 1, an action that covers a previously not covered edge in E is eventually created, after a polynomial number of invocations. This is due to the property (C) being enforced with probability p_a. Thus, it must eventually hold that $E_a \subseteq E$ and $E_a \setminus E_{A \cup A'} \neq \emptyset$ and the action will be added to the actions layer. Enforcing the property (A) will allow creating actions that add new atoms to the fact layer. Therefore, CREATELAYER terminates and returns a layer with m new facts or, if $m = 0$, with A' such that $E_{A \cup A'} = E$. As there are only a constant number of options, a new fact is achieved or a new causal graph edge is covered in time $O(1)$ and therefore CREATELAYER terminates in time $O(m + |E|)$. Since at least one new fact is added in each layer, Algorithm 1 terminates in time $O(n|E|)$. Since the while loop in line 4 terminates only when $E \setminus E_A = \emptyset$ and actions a are added to A only if $E_a \subseteq E$, when the algorithm terminates, we must have $E_A = E$. Therefore, the causal graph of the returned task Π must be exactly the graph G. $\square$

A feature of our algorithm is that it produces instances that are guaranteed to be delete-relaxed solvable, but not necessarily solvable. But what is the space of all delete-relaxed solvable SAS$^+$ planning tasks that can be produced by our algorithm? As it turns out, any such task can in principle be produced. To see that, observe that, except for the structure-based, any restrictions on action preconditions and effects, as well as the goal are optional. Specifically, the choice of goal to include facts exclusive to the last layer was merely aimed at increasing plan length.

Algorithm 1 Planning task construction.

```
Input: Graph $G = (\mathcal{V}, E)$,   number of facts $n \geq 2|\mathcal{V}|$,
    number of goals $n_g \leq |\mathcal{V}|$,
    maximal number of facts per layer $n_m$
    maximal number of prevail conditions $n_p$,
    maximal number of effects $n_e$,
    probability of goal achieved only in the last layer $p_g$,
    probability of enforcing a new edge $p_a$
Output: A planning task $\Pi$ with the causal graph G
 1: Partition $n$ into $|\mathcal{V}|$ values $d_v \geq 2$ s. t. $\sum_{v \in \mathcal{V}} d_v = n$
 2: $F_v \leftarrow \{\langle v, \vartheta \rangle \mid 0 \leq \vartheta < d_v\}$ for all $v \in \mathcal{V}$
 3: $s_0[v] \leftarrow 0$ for all $v \in \mathcal{V}$,  $k \leftarrow 0$,  $A \leftarrow \emptyset$,  $F_k \leftarrow s_0$
 4: while $|F_k| < n$ or $E \setminus E_A \neq \emptyset$ do
 5:     $m_{k+1} \leftarrow$ number of new facts ($\leq n_m$) for layer $k+1$
 6:     $F_{k+1}, A_k \leftarrow$ CREATELAYER($m_{k+1}$, $F_k$, $A$)
 7:     $A \leftarrow A \cup A_k$,  $k \leftarrow k + 1$
 8: Select $s_* \subseteq F_k$ s. t. $\forall v \in \mathcal{V}$, $|s_* \cap F_v| \leq 1$ and $|s_*| = n_g$
                         and  ($s_* \setminus F_{k-1} \neq \emptyset$ or  Rnd($p_g$))
 9: return $\Pi = \langle \mathcal{V}, A, s_0, s_* \rangle$
10: function CREATELAYER($m$, $F$, $A$)
11:     $A' \leftarrow \emptyset$,  $F' \leftarrow F$
12:     while $|F' \setminus F| < m$ or ($m = 0$ and $E \setminus E_{A \cup A'} \neq \emptyset$) do
13:         $a \leftarrow$ CREATEACTION( )
14:         if $E_a \subseteq E$ and ($E_a \setminus E_{A \cup A'} \neq \emptyset$ or Rnd($p_a$)) then
15:             $A' \leftarrow A' \cup \{a\}$,  $F' \leftarrow F' \cup eff(a)$
16:     return $F'$, $A'$
```

Theorem 2 *Given a delete-relaxed solvable* SAS$^+$ *planning task Π, the probability of Algorithm 1 producing a planning task that is equivalent to Π is greater than 0.*

Proof: The proof is straightforward from the construction. First, all the parameters of the algorithm can be taken from Π. This includes the causal graph, the number of variables and their domains, etc. Let $F_0, \dots, F_n$ and $A_0, \dots, A_{n-1}$ be the fact and action layers of the relaxed planning graph of Π. For simplicity, let us assume that the domain values of the variables in Π are $0, \dots, |dom(v)| - 1$, consistent with the order of the fact layers $F_0, \dots, F_n$. First, note that the first fact layer created by Algorithm 1 is F_0. Starting with the first layer, if the algorithm has produced the fact layers $F_0, \dots, F_i$ and action layers $A_0, \dots, A_{i-1}$, we show that it can produce the action layer A_i. Each action $o \in A_i$ has a non-zero probability of being constructed by our algorithm for creating the first layer. This is due to the fact that the precondition is chosen from F_i and the condition (C) can be enforced. For effects, step (III) does not impose any restrictions. Therefore, there is a non-zero probability of producing exactly the actions in A_i in any order and the algorithm can produce exactly the next fact layer F_{i+1}. $\square$

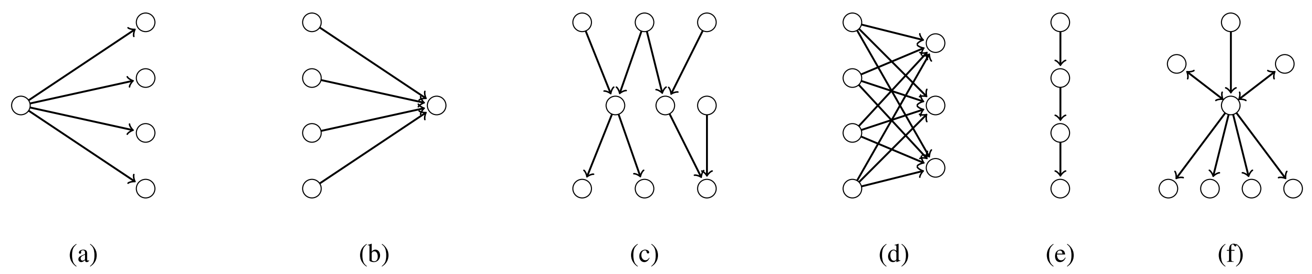


Figure 2: Graph structures: (a) fork, (b) inverted fork, (c) polytree, (d) directed bipartite graph, (e) chain, and (f) star.

Causal Graph Structures

The algorithm described above requires a graph to be provided as an input. As our construction works with any graph, we have implemented generation procedures for a variety of structures. We focus on well-defined and well-explored causal graph structures from the planning literature, rather than structures that occur in IPC domains. In the latter, existing translators mostly produce SAS$^+$ tasks with causal structure that is not easily characterized. However, the translation process can be guided by enforcing particular restrictions to obtain SAS$^+$ tasks with certain properties (Fišer et al. 2021). In principle, it should be possible to impose constraints on the causal graph structure to produce SAS$^+$ tasks with causal graphs from particular families, e.g., DAG.

We focus on the following graphs: *chain*, *directed chain*, *fork*, *inverted fork*, *star*, *bipartite graph*, *directed bipartite graph*, *tree*, *polytree*, *directed acyclic graph*, and *random graph*. For some of these structures, namely *directed chain*, *fork*, *inverted fork*, and *complete graph* the graphs are fully defined by the number of nodes (modulo automorphisms). In other cases, we introduce randomness into the graph construction. In what follows, we describe how we handle these cases.

Directed Bipartite Graph: A full directed bipartite graph is constructed by first randomly partitioning the nodes into left and right and then introducing an edge from each node on the left to each node on the right.

Bipartite Graph: A full undirected bipartite graph is constructed by first randomly partitioning the nodes into left and right and then introducing an edge from each node on the left to each node on the right, and vice versa.

Directed Chain: A *directed* chain of n nodes $v_1, \dots, v_n$ is created by adding the edges (v_i, v_{i+1}) for each $1 \leq i < n$.

Chain: An *undirected* chain of n nodes $v_1, \dots, v_n$ is created as follows. For each $1 \leq i < n$, we randomly decide whether to add an edge (v_i, v_{i+1}), with probability p. If no such edge is added, we add the edge (v_{i+1}, v_i) and if the edge (v_i, v_{i+1}) was added, we decide with probability p whether to add the edge (v_{i+1}, v_i).

Tree: A directed tree of n nodes $v_1, \dots, v_n$ is constructed by choosing for each node v_i a parent randomly out of the nodes $v_1, \dots, v_{i-1}$.

Polytree: For a polytree, we start with a tree constructed as above, and then for each edge switch its direction with probability p.

Directed Acyclic Graph: A directed acyclic graph of n nodes $v_1, \dots, v_n$ is constructed by choosing for each node v_i at least one parent randomly out of the nodes $v_1, \dots, v_{i-1}$. We do that by going over all the preceding nodes and deciding with probability p whether to add an edge from the preceding node to the current node. If no edges were added, we repeat until at least one edge is added for each node (except the first one).

Random Graph: For each pair of nodes v_i and v_j we randomly decide whether to add a directed edge from v_i to v_j.

Fork: A fork is a directed tree with all non-root nodes being leafs, with their parent being the root node. A fork over nodes $v_1, \dots, v_n$ is created by adding the edges (v_1, v_i) for each $1 < i \leq n$.

Inverted Fork: An inverted fork is a directed polytree with one leaf node and all non-leaf nodes being roots, with their only child node being the leaf node. An inverted fork over nodes $v_1, \dots, v_n$ is created by adding the edges (v_i, v_1) for each $1 < i \leq n$.

Star: A star structure has one central node with all other nodes connected with the central node only. A star over nodes $v_1, \dots, v_n$ is created as follows. For each $1 < i \leq n$, we randomly decide whether to add an edge (v_1, v_i), with probability p. If no such edge is added, we add the edge (v_i, v_1) and if the edge (v_1, v_i) was added, we decide with probability p whether to add the edge (v_i, v_1).

Figure 2 exemplifies selected graph structures. Probably the most explored causal graph structure in terms of complexity analysis is the *polytree* (Domshlak and Dinitz 2001; Domshlak and Brafman 2002; Katz and Domshlak 2007, 2008a; Giménez and Jonsson 2008; Bäckström and Jonsson 2013; Aghighi, Jonsson, and Ståhlberg 2015; Bäckström, Jonsson, and Ordyniak 2019). Other explored structures include *chains* (Domshlak and Dinitz 2001; Giménez and Jonsson 2008, 2009), *DAGs* and *directed-path singly connected graphs* (Domshlak and Dinitz 2001; Domshlak and Brafman 2002; Katz and Domshlak 2007, 2008a,b, 2010; Bäckström and Jonsson 2013), as well as (inverted) *forks* and (inverted) *trees* (Domshlak and Dinitz 2001; Domshlak and Brafman 2002; Katz and Domshlak 2007, 2008a,b, 2010; Katz and Keyder 2012). Note that in all these cases, planning remain hard unless additional, often extreme restrictions are imposed (Katz and Domshlak 2007, 2008a,b, 2010).

While our aim in this work is to provide a well-structured benchmarks set, our algorithm is in no way restricted to produce planning tasks based on these particular structures. In fact, even existing SAS$^+$ planning tasks can be used to seed our algorithm. Such tasks can provide all the necessary input: the causal graph and the variables, the number of facts, the number of goals, as well as the maximal number of prevail conditions or effects.

Experimental Evaluation

Our experimental evaluation consists of two parts. The first one focuses on the generation of a new benchmark set. The second one evaluates the generated benchmark set with state-of-the-art planners to ensure that the set is both sufficiently challenging and within the reach of the current state of the art in classical planning. Here, we focus on cost-optimal planning, creating a benchmark set suitable for current state-of-the-art cost-optimal planners.

Benchmark Set for Causal Graph Structures

We start by constructing a new benchmark set from interesting graph families as described in the previous section. The causal structures *star*, *chain*, *directed acyclic graph*, *random*, and *poly-tree*, have the edge probability as a parameter for the graph generator. We use the values 0.1, 0.25, 0.5, and 0.75 as edge probability for the causal structures. For the six causal structures *directed bipartite graph*, *bidirectional bipartite graph*, *directed chain*, *fork*, *inverted fork*, and *tree*, edge probability does not play a role in graph creation. In total, that gives us 26 planning domains. For each of these 26 domains, we have generated a compact and well-balanced set of 30 instances with the help of Autoscale (Torralba, Seipp, and Sievers 2021). Autoscale uses a collection of cost-optimal planners internally to estimate the "hardness" of candidate instances. This hardness is measured in terms of total time until the candidate instance was either solved by a planner or proved to be unsolvable. It is straightforward therefore to create benchmark sets more suitable for current satisficing planners instead, by replacing cost-optimal planners in Autoscale with satisficing ones. In what follows, we refer to this benchmark set as the *structured* benchmark set.

Autoscale takes as input a problem generator and a range of values of its input parameters and generates a set of tasks, aiming at producing tasks whose difficulty scales up smoothly. Here, we varied the number of variables and the average domain size, stabilizing the rest. These values, as well as the distribution of the instance generation parameters can be found in the supplementary material in the code repository. To prevent producing a large number of instances that are easily detected to be unsolvable, we use the h^2 mutex-based preprocessor (Alcázar and Torralba 2015) to quickly test for unsolvability and regenerate an instance with a different random seed.

Evaluating Structured Benchmark Set

To evaluate the performance of the state-of-the-art cost-optimal classical planners on the generated benchmark set, we have selected the top-performing non-portfolio optimal planners from the most recent IPC 2018: Complementary (Franco et al. 2018), Planning-PDBs (Moraru et al. 2018), and Scorpion (Seipp 2018). We excluded the portfolio planner Delfi (Sievers et al. 2019), and included instead its top performing components: the symbolic planner SYMBA* (Torralba et al. 2014) and explicit heuristic search with the LM-cut heuristic (Helmert and Domshlak 2009). The experiments were performed on Intel(R) Xeon(R) CPU E5-2683 v4 @ 2.10GHz machines, with a timeout of 30 minutes and memory limit of 3.5GB per run. Table 1 shows per-collection aggregated coverage comparison of the selected planners. Each task in a collection contributes a value of 1 to the coverage if it was either solved by the planner or the planner was able to prove the task to be unsolvable. Observe that a few domains such as *directed chain* and *random* seem to be easy for these planners, with the coverage reaching or coming close to 30. The rest, however, are quite challenging for these planners. Out of the total 780 tasks, 548 are solved by at least one planner, leaving 232 tasks not being solved by any of these state of the art planners.

Collection	Comp	PDBs	Scorp	SYMBA*	LM-cut
bipartite	5	5	6	**7**	6
bd-bipartite	21	21	**23**	21	**23**
chain 0.1	19	3	**23**	1	7
chain 0.25	22	8	**24**	4	7
chain 0.5	26	16	**28**	8	13
chain 0.75	26	13	**27**	3	1
dag 0.1	12	5	**18**	3	13
dag 0.25	4	3	7	0	**11**
dag 0.5	10	10	12	2	**17**
dag 0.75	7	6	**12**	1	12
d-chain	**29**	**29**	**29**	**29**	26
fork	2	5	**14**	1	6
inverted fork	9	7	11	11	**16**
polytree 0.1	20	14	**29**	9	13
polytree 0.25	10	8	**22**	4	10
polytree 0.5	12	9	**20**	3	13
polytree 0.75	12	8	**20**	3	17
random 0.1	21	21	**28**	12	**28**
random 0.25	6	10	19	6	**23**
random 0.5	26	25	16	25	**29**
random 0.75	**30**	**30**	29	**30**	**30**
star 0.1	6	5	7	0	**11**
star 0.25	5	6	6	0	**12**
star 0.5	10	9	11	1	**12**
star 0.75	6	6	7	0	**10**
tree	15	7	**24**	5	15
Sum (780)	371	289	**472**	189	381

Table 1: Coverage of state-of-the-art planning systems: Complementary (Comp), planning-PDBs (PDBs), Scorpion (Scorp), SYMBA*, and A^* with the LM-cut heuristic on the *structured* benchmark set.

Benchmark Set from IPC Instances

Finally, as mentioned above, our approach allows to generate a planning task that mimics the structure of an input

Collection	Comp	PDBs	Scorp	SymBA*	LM-cut
airport (30)	17	17	**30**	3	28
blocksworld (30)	**18**	**18**	17	11	15
childsnack (30)	**30**	**30**	**30**	23	**30**
depots (30)	**15**	14	13	8	14
driverlog (30)	**30**	**30**	28	29	20
elevators (30)	**29**	28	28	24	23
floortile (30)	**9**	**9**	7	8	5
freecell (30)	**30**	**30**	**30**	22	29
gripper (30)	**23**	**23**	18	22	15
logistics (30)	**29**	**29**	27	26	28
miconic (30)	**30**	**30**	28	**30**	26
mprime (30)	**24**	**24**	**24**	19	21
nomystery (30)	28	**30**	24	25	23
openstacks (30)	12	9	**13**	3	9
organic-s-sp (30)	13	13	**27**	3	9
parcprinter (30)	21	20	**25**	15	18
parking (30)	**14**	13	**14**	7	11
pipes-tank (30)	24	23	24	19	**25**
rovers (30)	**30**	**30**	28	**30**	27
satellite (30)	**30**	**30**	17	**30**	24
snake (30)	**26**	21	25	2	25
sokoban (30)	**21**	**21**	17	15	18
storage (30)	14	14	**15**	7	14
termes (30)	**23**	**23**	22	**23**	22
tetris (30)	10	11	**27**	12	15
thoughtful (30)	**29**	28	**29**	25	**29**
tidybot (30)	29	29	**30**	18	**30**
tpp (30)	20	11	**22**	8	10
transport (30)	**30**	29	28	**30**	25
visitall (29)	22	22	17	**25**	11
woodworking (30)	14	10	**17**	5	12
zenotravel (30)	29	**30**	29	29	27
Sum other (245)	245	245	245	245	245
Sum (1204)	968	944	**975**	801	883
Autoscale (1260)	415	**420**	339	385	241

Table 2: Coverage of state-of-the-art planning systems: Complementary (Comp), planning-PDBs (PDBs), Scorpion (Scorp), SymBA*, and A^* with the LM-cut heuristic on the *IPC-based* benchmark set.

planning task in SAS$^+$. We use the recently introduced Autoscale IPC benchmark set[2] (Torralba, Seipp, and Sievers 2021) as the source for the input parameters, generating a planning task for each task in that set. The choice of the benchmark set was primarily guided by its better reflection of what instances are hard for the current state of the art in classical planning. Here as well we use the h^2 mutex-based preprocessor (Alcázar and Torralba 2015) to test for unsolvability and regenerate the instance using a different random seed, making up to 1000 attempts. We have created 1204 out of possible 1260 instances in 42 domains. We refer to this benchmark set as *IPC-based*. Table 2 shows the per-domain coverage for the same state-of-the-art cost-optimal planners on these domains. There are 245 instances in 10 domains solved by all planners, not shown in the table. There is still a significant number of instances not solved by any planner, however, not surprisingly, not as much as for the actual Autoscale IPC benchmark set. For comparison, the last row in Table 2 shows how the selected planners perform on the actual Autoscale IPC benchmark set. Looking at the three benchmark sets, the structural one, the IPC-based one, and the Autoscale one, the selected planners' relative performance differs quite significantly from one to another. Scorpion, ranking first on both randomly generated ones, is ranked fourth out of five on Autoscale. Planning-PDBs is ranked first on Autoscale but ranked fourth and third on structural and IPC-based benchmark sets, respectively. LM-cut is ranked second on the structural benchmark set, fourth on IPC-based, and last on Autoscale. While this is not surprising, as it is well known that the choice of planning domains heavily influences planner rankings, it strengthens the motivation of additional benchmarks for classical planning. Here as well, 153 out if 1260 tasks are not solved by any of these planners.

[2]An existing publicly available benchmark generated from IPC domains using the Autoscale tool.

Generation of Solvable and Unsolvable Instances

Both of our generated benchmark sets contain a mixture of solvable and unsolvable planning tasks. We see this as a feature of our method rather than a limitation. Recall that the coverage indicates whether a planner was able to either solve the task or prove it to be unsolvable. The latter is especially challenging if there is no a priori knowledge whether the task is actually unsolvable, which is achievable only in such mixed collections of solvable and unsolvable tasks.

Translating SAS$^+$ to PDDL

While, to the best of our knowledge, most if not all modern competitive classical planners use SAS$^+$ representation internally (Helmert 2006; Ramirez, Lipovetzky, and Muise 2015), for the sake of completeness we also provide our benchmark set in PDDL. To create a PDDL task, the SAS$^+$ task is translated to the STRIPS fragment of PDDL, ignoring the facts that represent the at-most-one case. PDDL preconditions are taken from SAS$^+$ preconditions, add effects are taken from SAS$^+$ effects, and delete effects are taken from non-prevail preconditions. Note that a translation from STRIPS back to SAS$^+$ is not unique and the causal graph structure is not necessarily preserved by translating the PDDL representation to STRIPS and back to SAS$^+$. Therefore, we keep both the SAS$^+$ representation, as well as the translated PDDL. That way, planners that use SAS$^+$ representation can use the provided one, while planners that do not use SAS$^+$ can still benefit from our tasks, using the PDDL input format. Since such planners do not use SAS$^+$ and therefore causal graphs, it is less important for them that the SAS$^+$ structure might not be preserved.

Related Work

The idea of generating domain models as well as specific planning tasks has been explored in the planning community, with a major focus on learning domain models from traces, for classical planning (e.g., Yang, Wu, and Jiang 2007; Zhuo et al. 2010; Tian, Zhuo, and Kambhampati 2016) and HTN planning (e.g., Hogg, Muñoz-Avila, and Kuter 2008; Hogg, Kuter, and Muñoz-Avila 2010; Hogg, Muñoz-Avila, and Kuter 2016). The work often assumes an existence of a complete model the plan traces are generated from. Some aspects of these domain models are then learned or reconstructed from successful plan traces. Some examples include learning action preconditions (Zhuo et al. 2009), or refine incomplete action descriptions (Zhuo, Nguyen, and Kambhampati 2013).

Probably more related to our work is the work on generating problem instances for CSP/SAT problems (e.g., Achlioptas et al. 2000; Xu et al. 2005). There are several online tools/services such as the "Tough SAT Project" or "SATLIB" that generate CNF formulas encoding "difficult" problems (e.g., Yuen and Bebel 2017; Hoos and Stützle 2000). Producing hard satisfiable instances has advanced the research field in SAT/CSP. These instances can be polynomial-time reduced to STRIPS in theory, but also in practice (Porco, Machado, and Bonet 2011). The authors provide a tool to translate multiple NP-complete computational problem instances (including SAT, CLIQUE, DirectedHamiltonianPath, etc.) into an NP-Complete fragment of STRIPS that they call STRIPS-1. In that fragment, the actions are either delete-free or can be applied at most once. While the fragment is somewhat limited, the approach can be used for creating additional benchmark sets for planning. Unfortunately, the work has not yet received the attention it deserves, and the instances or the tools are not currently widely used. It is worth mentioning that our suggested approach to generating random PDDL instances is a somewhat different task than generating random CNF formula, and then translating to STRIPS. Our focus is on being able to control the causal structure of the generated problem, which is not possible with the aforementioned methods.

Highly related is the work on random planning tasks generation for analyzing phase transition in classical planning (Bylander 1996; Rintanen 2004). The authors propose a variety of models for sampling the space of STRIPS planning problem instances, exploring the possibility of phase transition at some constant ratio of the number of actions to the number of state variables. These models correspond to a constrained set of problem instances, restricting the sizes of preconditions and effects, and reducing the chances of generating trivially unsolvable tasks. Unfortunately, the proposed methods for generating tasks do not yield tasks of a desired structure and it is not clear what additional restrictions can be imposed in order to obtain such tasks.

Discussion and Future Work

We have presented a method to generate planning tasks of specific causal graph structure. We also cast these tasks into a STRIPS fragment of PDDL, allowing using as an input to any PDDL planner. With the help of AutoScale we generated a benchmark set of 26 task collections for various causal graph structures, with 30 tasks in each collection. Based on a recently proposed Autoscale IPC benchmark set, we generate a benchmark set mimicking the structure of existing planning tasks. Our experimental evaluation clearly shows that the two generated benchmark sets are challenging for the state-of-the-art in classical optimal planning. In the hope to facilitate further research and enable better comparison of planning tools, we make our tool and the benchmark sets publicly available.

The paper opens up a variety of avenues for future work. First, we would like to investigate the phase transition in planning according to structural characterization of planning tasks. We conjecture that phase transition might appear at different number of actions to state variables ratios for different causal graph structures. Additionally, we would like to explore the usage of generating planning tasks for the purpose of learning a planner selection strategy. Third, we would like to explore the possibility of generating lifted PDDL tasks of a meaningful causal structure. Further, we have explored so far causal graphs as the main structural characterization of generated planning tasks. We would like to explore additional structural restrictions of planning tasks, such as, e.g., k-dependence (Katz and Domshlak 2008a).

References

Achlioptas, D.; Gomes, C. P.; Kautz, H. A.; and Selman, B. 2000. Generating Satisfiable Problem Instances. In *Proc. AAAI 2000*, 256–261.

Aghighi, M.; Jonsson, P.; and Ståhlberg, S. 2015. Tractable Cost-Optimal Planning Over Restricted Polytree Causal Graphs. In *Proc. AAAI 2015*, 3225–3231.

Alcázar, V.; and Torralba, Á. 2015. A Reminder about the Importance of Computing and Exploiting Invariants in Planning. In *Proc. ICAPS 2015*, 2–6.

Bäckström, C.; and Jonsson, P. 2013. A Refined View of Causal Graphs and Component Sizes: SP-Closed Graph Classes and Beyond. *JAIR*, 47: 575–611.

Bäckström, C.; Jonsson, P.; and Ordyniak, S. 2019. A Refined Understanding of Cost-optimal Planning with Polytree Causal Graphs. In *Proc. IJCAI 2019*, 6126–6130.

Bäckström, C.; and Nebel, B. 1995. Complexity Results for SAS$^+$ Planning. *Computational Intelligence*, 11(4): 625–655.

Bonet, B.; Palacios, H.; and Geffner, H. 2009. Automatic Derivation of Memoryless Policies and Finite-state Controllers Using Classical Planners. In *Proc. ICAPS 2009*, 34–41.

Bylander, T. 1996. A Probabilistic Analysis of Propositional STRIPS Planning. *AIJ*, 81(1–2): 241–271.

Domshlak, C.; and Brafman, R. I. 2002. Structure and Complexity in Planning with Unary Operators. In *Proc. AIPS 2002*, 34–43.

Domshlak, C.; and Dinitz, Y. 2001. Multi-Agent Coordination Off-Line: Structure and Complexity. In *Proc. ECP 2001*, 69–76.

Edelkamp, S. 2001. Planning with Pattern Databases. In *Proc. ECP 2001*, 84–90.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *AIJ*, 2: 189–208.

Fišer, D.; Gnad, D.; Katz, M.; and Hoffmann, J. 2021. Custom-Design of FDR Encodings: The Case of Red-Black Planning. In *Proc. IJCAI 2021*, 4054–4061.

Franco, S.; Lelis, L. H. S.; Barley, M.; Edelkamp, S.; Martines, M.; and Moraru, I. 2018. The Complementary1 Planner in the IPC 2018. In *IPC-9 planner abstracts*, 28–31.

Giménez, O.; and Jonsson, A. 2008. The Complexity of Planning Problems With Simple Causal Graphs. *JAIR*, 31: 319–351.

Giménez, O.; and Jonsson, A. 2009. Planning over Chain Causal Graphs for Variables with Domains of Size 5 is NP-Hard. *JAIR*, 34: 675–706.

Giménez, O.; and Jonsson, A. 2012. The influence of k-dependence on the complexity of planning. *AIJ*, 177: 25–45.

Grastien, A.; and Scala, E. 2018. Sampling Strategies for Conformant Planning. In *Proc. ICAPS 2018*, 97–105.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning. In *Proc. AAAI 2007*, 1007–1012.

Helmert, M. 2004. A Planning Heuristic Based on Causal Graph Analysis. In *Proc. ICAPS 2004*, 161–170.

Helmert, M. 2006. The Fast Downward Planning System. *JAIR*, 26: 191–246.

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *Proc. ICAPS 2009*, 162–169.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible Abstraction Heuristics for Optimal Sequential Planning. In *Proc. ICAPS 2007*, 176–183.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *JAIR*, 14: 253–302.

Hogg, C.; Kuter, U.; and Muñoz-Avila, H. 2010. Learning Methods to Generate Good Plans: Integrating HTN Learning and Reinforcement Learning. In *Proc. AAAI 2010*, 1530–1535.

Hogg, C.; Muñoz-Avila, H.; and Kuter, U. 2008. HTN-MAKER: Learning HTNs with Minimal Additional Knowledge Engineering Required. In *Proc. AAAI 2008*, 950–956.

Hogg, C.; Muñoz-Avila, H.; and Kuter, U. 2016. Learning Hierarchical Task Models from Input Traces. *Computational Intelligence*, 32(1): 3–48.

Hoos, H.; and Stützle, T. 2000. SATLIB: An Online Resource for Research on SAT. 283–292.

Katz, M.; and Domshlak, C. 2007. Structural Patterns of Tractable Sequentially-Optimal Planning. In *Proc. ICAPS 2007*, 200–207.

Katz, M.; and Domshlak, C. 2008a. New Islands of Tractability of Cost-Optimal Planning. *JAIR*, 32: 203–288.

Katz, M.; and Domshlak, C. 2008b. Structural Patterns Heuristics via Fork Decomposition. In *Proc. ICAPS 2008*, 182–189.

Katz, M.; and Domshlak, C. 2010. Implicit Abstraction Heuristics. *JAIR*, 39: 51–126.

Katz, M.; and Keyder, E. 2012. Structural Patterns Beyond Forks: Extending the Complexity Boundaries of Classical Planning. In *Proc. AAAI 2012*, 1779–1785.

McDermott, D. 2000. The 1998 AI Planning Systems Competition. *AI Magazine*, 21(2): 35–55.

Moraru, I.; Edelkamp, S.; Martinez, M.; and Franco, S. 2018. Planning-PDBs Planner. In *IPC-9 planner abstracts*, 69–73.

Palacios, H.; and Geffner, H. 2009. Compiling Uncertainty Away in Conformant Planning Problems with Bounded Width. *JAIR*, 35: 623–675.

Porco, A.; Machado, A.; and Bonet, B. 2011. Automatic polytime reductions of NP problems into a fragment of STRIPS. In *Proc. ICAPS 2011*, 178–185.

Ramirez, M.; Lipovetzky, N.; and Muise, C. 2015. Lightweight Automated Planning ToolKiT. http://lapkt.org/. Accessed: 2020-01-01.

Rintanen, J. 2004. Phase Transitions in Classical Planning: an Experimental Study. In *Proc. ICAPS 2004*, 101–110.

Seipp, J. 2018. Fast Downward Scorpion. In *IPC-9 planner abstracts*, 77–79.

Sievers, S.; Katz, M.; Sohrabi, S.; Samulowitz, H.; and Ferber, P. 2019. Deep Learning for Cost-Optimal Planning: Task-Dependent Planner Selection. In *Proc. AAAI 2019*, 7715–7723.

Sohrabi, S.; Riabov, A. V.; Katz, M.; and Udrea, O. 2018. An AI Planning Solution to Scenario Generation for Enterprise Risk Management. In *Proc. AAAI 2018*, 160–167.

Tian, X.; Zhuo, H. H.; and Kambhampati, S. 2016. Discovering Underlying Plans Based on Distributed Representations of Actions. In *Proc. AAMAS 2016*, 1135–1143.

Torralba, Á.; Alcázar, V.; Borrajo, D.; Kissmann, P.; and Edelkamp, S. 2014. SymBA*: A Symbolic Bidirectional A* Planner. In *IPC-8 planner abstracts*, 105–109.

Torralba, Á.; Seipp, J.; and Sievers, S. 2021. Automatic Instance Generation for Classical Planning. In *Proc. ICAPS 2021*, 376–384.

Xu, K.; Boussemart, F.; Hemery, F.; and Lecoutre, C. 2005. A Simple Model to Generate Hard Satisfiable Instances. In *Proc. ICAPS 2005*, 337–342.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning Action Models from Plan Examples Using Weighted MAX-SAT. *AIJ*, 171: 107–143.

Yuen, H.; and Bebel, J. 2017. Tough SAT Project. https://toughsat.appspot.com/. Accessed: 2015-05-07.

Zhuo, H. H.; Hu, D. H.; Hogg, C.; Yang, Q.; and Muñoz-Avila, H. 2009. Learning HTN Method Preconditions and Action Models from Partial Observations. In *Proc. IJCAI 2009*, 1804–1810.

Zhuo, H. H.; Nguyen, T.; and Kambhampati, S. 2013. Refining Incomplete Planning Domain Models Through Plan Traces. In *Proc. IJCAI 2013*, 2451–2457.

Zhuo, H. H.; Yang, Q.; Hu, D. H.; and Li, L. 2010. Learning Complex Action Models with Quantifiers and Logical Implications. *AIJ*, 174(18): 1540–1569.