A Novel Iterative Approach to Top-k Planning

Michael Katz, Shirin Sohrabi, Octavian Udrea
IBM Research
Yorktown Heights, NY, USA

Dominik Winterer*
University of Freiburg
Germany

Abstract

While cost-optimal planning aims at finding one best quality plan, top-k planning deals with finding a set of solutions, such that no better quality solution exists outside that set. We propose a novel iterative approach to top-k planning, exploiting any cost-optimal planner and reformulating a planning task to forbid exactly the given set of solutions. In addition, to compare to existing approaches to finding top-k solutions, we implement the K^* algorithm in an existing PDDL planner, creating the first K^* based solver for PDDL planning tasks. We empirically show that the iterative approach performs better for up to a large required size solution sets (thousands), while K^* based approach excels on extremely large ones.

Introduction

Cost-optimal planning is the problem of finding one goal-achieving sequence of actions or a plan of a minimal summed up cost. Such plans are of interest in many applications, where the quality of solution is of extreme importance. In particular, in problems where preferences or likelihoods are encoded as cost of the actions (e.g., (Keyder and Geffner 2009; Sohrabi, Udrea, and Riabov 2013)). In many cases, one best solution is not sufficient, and the desire is to obtain a set of solutions of a high quality. Top-k planning is one way of obtaining such a set. It is the problem of finding a set of solutions of size k, such that no better solution exists outside the set. While finding a set of plans is motivated by several applications including plan repair, often within the context of diverse planning (e.g., (Fox et al. 2006; Bryce 2014; Coman and Muñoz-Avila 2011)), other applications including risk management, hypothesis and explanation generation require to focus on high-quality plans instead, as the underlying approach of plan-recognition-as-planning works best with such plans (Sohrabi, Udrea, and Riabov 2013; Sohrabi, Riabov, and Udrea 2016; Sohrabi et al. 2018).

Recently, Sohrabi et al. (2016) proposed the use of a k-shortest path algorithm called K^* by Aljazzar and Leue (2011) to address the top-k planning problem. Their experimental results have shown that the planning time to compute top-k plans is comparable to finding the optimal plan; in addition, the quality of the solutions found is much higher compared to the use of the diverse version of the LPG planner (Nguyen et al. 2012) which computes a set of diverse plans. This is not surprising, as the focus of diverse planning is not on plan quality, making top-k planning the more suitable approach to the applications above.

One major limitation of K^* is the consistency requirement for the heuristic function. This poses a restriction on the usability of the approach, disallowing the use of one of the most successful heuristics to date (Helmert and Domshlak 2009). Further, all plans are found almost simultaneously, requiring long time until the first solution is found. Moreover, the existing implementation of K^* for planning is in a planner that supports Stream Processing Planning Language (SPPL) (Riabov and Liu 2005), and not in PDDL. Thus, there are no experimental results for K^* on the standard planning benchmarks. To alleviate these problems, Riabov, Sohrabi, and Udrea (2014) suggest an iterative approach to top-k planning, given a solution to a planning task, encode a set of new planning tasks which, cumulatively, preserve all solutions of the original task, except for the given one. Then, a search is performed on a tree of reformulations, invoking an existing planner in each node. As the number of successors of each node is the number of actions in the found plan, the clear down side of such an approach is the large number of invocations of the underlying planner. On the positive side, the approach exhibits an anytime behavior, with the first plan found rather quickly.

In this work, we propose an alternative iterative approach to top-k planning, finding additional solutions by reformulating the planning task at hand into a single task, preserving all solutions except for the given one. To this end, we formally define such reformulations and present one such instance. We then suggest additional ways of deriving solutions from the previously found ones, further reducing the number of invocations of the underlying cost-optimal planner. For that, we extend our reformulation to forbid multiple plans at once, alleviating the increase in task formulation size. Additionally, we implement the K^* algorithm on top of the Fast Downward planning system (Helmert 2006), allowing us to perform the long overdue experimental evaluation on International Planning Competition (IPC) benchmarks, comparing our new iterative approach to K^*. Our results show that K^* works better for extremely large values of k, while the iterative approach excels otherwise.

*The work was performed during an internship at IBM

Background

We consider classical planning tasks as captured by the well-known SAS$^+$ formalism (Bäckström and Nebel 1995), extended with action costs. In such a *planning task* $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star, cost \rangle$, $\mathcal{V}$ is a finite set of finite-domain *state variables*. Each variable $v \in \mathcal{V}$ is associated with a finite domain $\mathcal{D}(v)$ of variable values. A *partial assignment* p maps a subset of variables $vars(p) \subseteq \mathcal{V}$ to values in their domains. For a variable $v \in \mathcal{V}$ and partial assignment p, the value of v in p is denoted by $p[v]$ if $v \in vars(p)$ and we say $p[v]$ is *undefined* if $v \notin vars(p)$. A partial assignment s with $vars(s) = \mathcal{V}$, is called a *state*. State s is *consistent* with partial assignment p if they agree on all variables in $vars(p)$, shortly denoted by $p \subseteq s$. The product $\mathcal{S} = \prod_{v \in \mathcal{V}} \mathcal{D}(v)$ is called the *state space* of planning task Π. The state s_0 is called *initial state* of Π and the partial assignment $s_\star$ is called the *goal* of Π. A state s is called a *goal state* if $s_\star \subseteq s$ and the set of all goal states is denoted by $\mathcal{S}_{s_\star}$. The finite set $\mathcal{O}$ is a set of *actions*, each action is a pair $\langle pre, \mathit{eff} \rangle$ where pre is a partial assignment called *precondition* and eff is a partial assignment called *effect*. Further, each action o has an associated natural number $cost(o)$, called *cost*. An action $o = \langle pre, \mathit{eff} \rangle$ is applicable in state s if $pre \subseteq s$. Applying action o in state s results in a state denoted by $s[\![o]\!]$ where $s[\![o]\!][v] = \mathit{eff}[v]$ for all $v \in vars(\mathit{eff})$ and $= s[\![o]\!][v] = s[v]$ for all other variables. An action sequence $\pi = \langle o_1, \cdots, o_n \rangle$ is applicable in state s if there are states $s_0, \cdots, s_n$ such that o_i is applicable in s_{i-1} and $s_{i-1}[\![o_i]\!] = s_i$ for $0 \leq i \leq n$. We denote s_n by $s[\![\pi]\!]$. For convenience we often write $o_1, \cdots, o_n$ instead of $\langle o_1, \cdots, o_n \rangle$. An action sequence with $s[\![\pi]\!] \in \mathcal{S}_{s_\star}$ is called a *plan*. The cost of a plan π, denoted by $cost(\pi)$ is the summed cost of the actions in the plan. For a planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star, cost \rangle$, the set of all plans is denoted by $\mathcal{P}_\Pi$. A plan π is *optimal* if its cost is minimal among all plans in $\mathcal{P}_\Pi$. Let Π, Π' be two planning tasks with actions $\mathcal{O}$ and $\mathcal{O}'$ respectively. A total function $r : \mathcal{O}' \mapsto \mathcal{O}$ is called an *action mapping*. We call the mapping $r' : \mathcal{O}'^n \mapsto \mathcal{O}^n$ for a non-negative integer n, the *extension of action mapping* r if for every action sequence $\pi' = \langle o'_1, \cdots, o'_n \rangle$ with $o_i \in \mathcal{O}'$ for $0 \leq i \leq n$ there is an action sequence $\pi = \langle r(o'_1), \cdots, r(o'_n) \rangle$ and $r'(\pi') = \pi$.

Having provided the necessary background on classical planning, let us define the *top-k planning problem* (Sohrabi et al. 2016) that is the subject of interest in this paper.

Definition 1 (top-k planning problem) *Given a planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star, cost \rangle$ and a natural number k, find a set of plans $P \subseteq \mathcal{P}_\Pi$ such that:*

(i) for all plans $\pi \in P$, if there exists a plan π' for Π with $cost(\pi') < cost(\pi)$, then $\pi' \in P$, and

(ii) $|P| \leq k$, with $|P| < k$ implying $P = \mathcal{P}_\Pi$.

An instance of the top-k planning problem $\langle \Pi, k \rangle$, is called solvable if $|P| = k$ and unsolvable if $|P| < k$.

The objective of top-k planning is finding the k-plans of lowest costs for a planning task Π and thus optimal planning is the special case of top-1 planning. Our approach to solving the top-k planning problem is the topic of the next section.

Repeatedly Forbidding Plans

We now introduce an algorithmic scheme for solving a top-k planning problem $\langle \Pi, k \rangle$. The key idea of this scheme is an iteration of the following steps: (1) Find an optimal plan π for planning task Π. (2) Reformulate Π to a planning task Π' with the same set of plans but excluding π. (3) Repeat (1) with $\Pi = \Pi'$ and $\pi = \pi'$ unless either k solutions have been found or the Π' is provably unsolvable. The scheme is summarized below.

Algorithm 1: IterativeTopK($\langle \Pi, k \rangle$)

```
P ← ∅,  π ← ∅,  r ← id
while |P| < k and no failure occurs do
    Π, r' ← PlanForbidReformulation(Π, π)
    r ← r ∘ r'
    π ← GetOptimalPlan(Π)
    if GetOptimalPlan(Π) reports UNSOLVABLE
     then
        return P
    end
    P ← P ∪ {r(π)}
end
return P
```

The key challenge of turning this algorithmic scheme into a concrete algorithm lies in the reformulation. More concretely, in how to *forbid* exactly one plan while at the same time preserving every other plan. Next, we state a necessary condition for such a reformulation.

Definition 2 (plan forbidding reformulation) *Let Π be a planning task over actions $\mathcal{O}$ and π be a plan for Π. A planning task Π' over actions $\mathcal{O}'$ is a* **plan forbidding reformulation** *of Π if there exists an action mapping $r : \mathcal{O}' \mapsto \mathcal{O}$ and its extension r' such that π' is a plan for Π' iff $r'(\pi')$ is a plan for Π with $cost'(\pi') = cost(r'(\pi'))$ and $r'(\pi') \neq \pi$.*

We proceed by proving soundness and completeness of our scheme given some plan forbidding reformulation. Later we will show how to construct a specific such reformulation.

Theorem 1 *The algorithm* IterativeTopK *is sound and complete.*

Proof: Let P be the set of plans returned by the algorithm, $\pi_1 \dots \pi_m$ be the ordering in which the plans were found and let $\Pi_1 \dots \Pi_m$ be the sequence of task reformulations constructed by the algorithm such that π_i is the optimal plan found for Π_i. Then $cost(\pi_1) \leq cost(\pi_2) \leq \cdots \leq cost(\pi_m)$. If there exists a plan π for Π such that $cost(\pi) < cost(\pi_i)$ for some (assume W.L.O.G. smallest such) i, and $\pi \neq \pi_j$ for $j < i$, then from Definition 2, there exists a plan π' for Π_i such that $r(\pi') = \pi$ and $cost(\pi') = cost(\pi) < cost(\pi_i)$, contradicting the optimality of the plan π_i for Π_i. If $m < k$, then the planning task Π_{m+1} is unsolvable, and therefore no other solution exists for Π_m and thus for Π. □

Theorem 1 opens the door to a novel family of algorithms for top-k planning. Yet, what is missing is to provide a concrete plan forbidding reformulation Π^-_π for planning task Π and a plan $\pi \in \mathcal{P}_\Pi$. As mentioned earlier the idea here is that planning task Π^-_π forbids the sequence of actions π from being a plan, accepting all other plans of Π.

Definition 3 *Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star, cost \rangle$ be a planning task and $\pi = o_1 \dots o_n$ be some plan. The task $\Pi^-_\pi = \langle \mathcal{V}', \mathcal{O}', s'_0, s'_\star, cost' \rangle$ is defined as follows.*

- $\mathcal{V}' = \mathcal{V} \cup \{\overline{v}, \overline{v}_0, \dots \overline{v}_n\}$*, with $\overline{v}_i$ being binary variables,*
- $\mathcal{O}' = \{o^e \mid o \in \mathcal{O}, o \notin \pi\} \cup \{o^1, o^2 \mid o \in \pi\} \cup \bigcup_{i=1}^n \{o^3_i\}$*, where*

$$o^e = \langle pre(o), \mathit{eff}(o) \cup \{\langle \overline{v}, 0 \rangle\} \rangle,$$
$$o^1 = \langle pre(o) \cup \{\langle \overline{v}, 0 \rangle\}, \mathit{eff}(o) \rangle,$$
$$o^2 = \langle pre(o) \cup \{\langle \overline{v}, 1 \rangle\} \cup \bigcup_{o=o_i, 1 \le i \le n} \{\langle \overline{v}_{i-1}, 0 \rangle\}, \mathit{eff}(o) \cup \{\langle \overline{v}, 0 \rangle\} \rangle,$$
$$o^3_i = \langle pre(o_i) \cup \{\langle \overline{v}, 1 \rangle, \langle \overline{v}_{i-1}, 1 \rangle\}, \mathit{eff}(o_i) \cup \{\langle \overline{v}_{i-1}, 0 \rangle, \langle \overline{v}_i, 1 \rangle\} \rangle, \text{and}$$

  $cost'(o^e) = cost'(o^1) = cost'(o^2) = cost'(o^3) = cost(o)$,

- $s'_0[v] = s_0[v]$ *for all* $v \in \mathcal{V}$, $s'_0[\overline{v}] = 1$, $s'_0[\overline{v}_0] = 1$, *and* $s'_0[\overline{v}_i] = 0$ *for all* $1 \le i \le n$, *and*
- $s'_\star[v] = s_\star[v]$ *for all* $v \in \mathcal{V}$ *s.t.* $s_\star[v]$ *defined, and* $s'_\star[\overline{v}] = 0$.

Let us explain the semantics of the reformulation in Definition 3. The variable $\overline{v}$ starts from the value 1 and switches to 0 when the sequence of actions applied is not a prefix of π. Once a value 0 is reached indicating a deviating from plan π, it cannot be switched back to 1. Variables $\overline{v}_0, \dots, \overline{v}_n$ encode the progress along the plan π, before deviating from it. The actions o^1 and o^2 are the copies of actions in π for the cases when π is already discarded from consideration (variable $\overline{v}$ has switched its value to 0) and for discarding π from consideration (switching $\overline{v}$ to 0). In case there are multiple appearances of an action o on a plan, o^2 requires all the corresponding variables $\overline{v}_{i-1}$ for each $o_i = o$ to have the value 0. Thus, o^2 will be applicable only in states that are different from the states resulting from applying a prefix of π up to some o's representative in the initial state. o^3_i are copies of actions along π, with a separate copy for each appearance of the same action o. These actions are responsible for following the sequence π and are applicable only while the sequence is still followed.

In the following theorem, we show that the planning task in Definitions 3 is indeed a plan-forbidding reformulation.

Theorem 2 *Let Π be a planning task and π be its plan. The task Π^-_π is a plan forbidding reformulation of Π and π.*

Proof: Let $r : \mathcal{O}' \mapsto \mathcal{O}$ be the mapping defined by $r(o^e) = o$ and $r(o^1) = r(o^2) = r(o^3_i) = o$ for all $1 \le i \le n$. Note that Π^-_π restricted to the variables $\mathcal{V}$ equals to the task Π, modulo the three equal instances of the actions in π. Thus, for each plan π' for Π^-_π, $r(\pi)$ is a plan for Π.

For the second direction, since for each $o \in \pi$ at most one of the actions o^1, o^2, o^3 is applicable in each state s of Π^-_π, given a sequence of actions ρ applicable in the initial state of Π, it can be mapped to an applicable in the initial state of Π^-_π sequence of action ρ' such that $r(\rho') = \rho$, by choosing in each state the relevant representative out of o^e, o^1, o^2, and o^3. In other words, r restricted to applicable in the initial state sequences of actions is invertible, and we denote its inverse mapping described above by r^{-1}.

First, let $\pi' = r^{-1}(\pi)$ be the inverse of the plan $\pi = o_1 \dots o_n$ for Π. Then $\pi' = o^3_1 \dots o^3_n$, since at step i we have $\overline{v}_0 = 1$ and $\overline{v}_{i-1} = 1$. Thus, after applying π', the value of the variable $\overline{v}$ remains 1, and thus π' is not a plan for Π^-_π.

Now, let ρ be a plan for Π such that $\rho \neq \pi$. Let o be the first action on ρ that differs from the corresponding action of π. In other words, there exists a prefix $\rho' = o_1 \dots o_m$ of ρ such that (i) $\rho' o$ is a prefix of ρ, (ii) ρ' is a prefix of π, and (iii) $\rho' o$ is not a prefix of π. Then we have $r^{-1}(\rho') = o^3_1 \dots o^3_m$, and since $o \neq o_{m+1}$, the next action on $r^{-1}(\rho)$ will not be o^3_{m+1}. If $o \in \pi$, then the next action will be o^2 (which we next show applicability of), and otherwise it will be o^e, in both cases setting the value of $\overline{v}$ to 0. Thus, all the following actions o' are mapped to either o'^e or o'^1, and the preconditions of these actions are restricted to $\mathcal{V}$ and $\overline{v} = 0$, the sequence of actions $r^{-1}(\rho)$ achieves the goal values on $\mathcal{V}$ and thus is a plan.

We finalize the proof by showing the applicability of o^2 in the state $s'_m := s'_0[\![o^3_1 \dots o^3_m]\!]$ for $o \in \pi$ such that $o \neq o_{m+1}$. Naturally, $pre(o)$ holds in s'_m. Further, since $s'_0[\overline{v}] = 1$, $s'_0[\overline{v}_0] = 1$, and $s'_0[\overline{v}_i] = 0$ for all $1 \le i \le n$, after applying $o^3_1 \dots o^3_m$ we have $s'_m[\overline{v}] = 1$, $s'_m[\overline{v}_m] = 1$, and $s'_m[\overline{v}_i] = 0$ for all $0 \le i \le n$, $i \neq m$. Since $o \neq o_{m+1}$, for each $1 \le i \le n$ such that $o = o_i$, we have $s'_m[\overline{v}_{i-1}] = 0$, and thus o^2 is applicable in s'_m. □

Devising Additional Plans

The top-k planning approach proposed in the previous section reformulates a planning task of a solvable top-k planning instance exactly k times. While in each iteration, the reformulated planning task grows only linearly in the plan size, for larger k, such an approach is prohibitively expensive for anything but small tasks. How can we bypass this problem? Given an optimal plan π for Π, it is often possible to infer additional optimal plans for Π from the structure of the planning task Π. Our idea is to forbid in each iteration a set of plans instead of a single plan, decreasing the number of reformulations needed. Before we introduce an algorithm for this approach, we focus on characteristics of a graph $G(P)$ representing such a set of plans.

First, given two plans π_1 and π_2, if these plans intersect, i.e., pass through the same state s, then additional plans may be devised out of these two by following one of the plans until the state s and the other plan from the state s onwards. In general, a set of plans P induces a directed graph $G(P)$ over the states of Π with edges annotated by the actions on the plans. Each path in $G(P)$ from the initial state to some goal state is a plan for Π. Formally, $G(P) = (N, E)$, where $N = \{s \in \mathcal{S} \mid o_1 \dots o_n \in P, s = s_0[\![o_1 \dots o_i]\!], 0 \le i \le n\}$

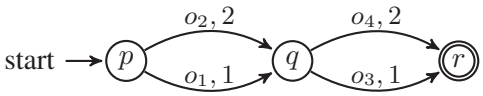

Figure 1: An example task with one variable and four actions changing its value from the initial value p to q and to the goal value r. Edges are labelled with actions, costs.

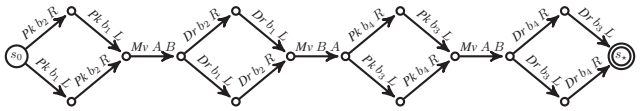

Figure 2: An order reduction of the example plan for the GRIPPER task with 4 balls.

are all the states encountered by the plans in P and $E = \{(s,t) \mid s,t \in \mathcal{S}, o_1 \dots o_n \in P, s = s_0[\![o_1 \dots o_{i-1}]\!], t = s[\![o_i]\!], 1 \leq i \leq n\}$ its edges. Each edge (s,t) is labelled with the action $o \in \pi \in P$ inducing it.

$G(P)$ can be viewed as a compact representation for a set of plans P of a planning task Π. Hence, often more plans are represented by $G(P)$ as compared to P. Let us now proof the correspondence of paths in $G(P)$ and plans for Π.

Lemma 1 *Let Π be a planning task and P be a set of plans for Π. Then, any path in $G(P)$ from s_0 to some goal state of Π corresponds to a plan for Π.*

Proof: Let $s_0, s_1, \dots, s_n$ with $s_n \in \mathcal{S}_{s_\star}$ be some path in $G(P)$. Each edge (s_{i-1}, s_i) corresponds to some action o_i on a plan in P, and thus o_i is applicable in s_{i-1}, giving us $o_1 \dots o_n$ being a plan for Π. □

Theorem 3 *Let Π be a planning task and P be a set of optimal plans for Π. Then, any path in $G(P)$ from s_0 to some goal state of Π corresponds to an optimal plan for Π.*

Proof: Let $s_0, s_1, \dots, s_n$ with $s_n \in \mathcal{S}_{s_\star}$ be some path in $G(P)$. From Lemma 1 we have that it corresponds to some plan $o_1 \dots o_n$ for Π, where each edge (s_{i-1}, s_i) corresponds to an action o_i on some optimal plan in P. Therefore, $h^*(s_{i-1}) = h^*(s_i) + cost(o_i)$ or $cost(o_i) = h^*(s_{i-1}) - h^*(s_i)$. Summing over the actions in the plan we get $\sum_{i=1}^{n} cost(o_i) = \sum_{i=1}^{n} h^*(s_{i-1}) - h^*(s_i) = h^*(s_0)$. □

If not all plans in P are optimal, we may get plans from $G(P)$ with costs larger than of any plan in P. A simple example for that is described in Figure 1. There is one optimal plan $\pi_1 = o_1 o_3$ with the cost 2, there are two plans with the cost 3, $\pi_2 = o_1 o_4$ and $\pi_3 = o_2 o_3$. If $P = \{\pi_1, \pi_2, \pi_3\}$, then there is also a path in $G(P)$ that corresponds to a plan $o_2 o_4$ with the cost 4, which is strictly larger than of any of the plans in P. An algorithm for extracting plans out of $G(P)$ must therefore, be able to extract paths of a bounded cost from $G(P)$. One approach is a simple traversal of $G(P)$, starting from the node s_0. Given a bound b on the total plan cost, a plan can be incrementally constructed by moving along an edge (x,y) that corresponds to an action o only if for the plan prefix constructed so far π that leads from s_0 to x holds $cost(\pi) + cost(o) + w^*(y) \leq b$, where $w^*(y)$ is the cost of the cheapest path from y to some goal node $t \in \mathcal{S}_{s_\star}$. In other words, we can extend π with o if there is a possibility to reach the goal under the bound.

Partial Orders on Plans

One simple way to derive additional plans from an existing one is by reordering the actions along the plan. A sequential plan corresponds to a total order over the actions in the plan. It can be reduced to a partial order, which corresponds to multiple total orders, and thus to multiple plans. A minimal partial order can be obtained in polynomial time (Bäckström 1998). The suggested procedure greedily attempts to remove an ordering between two plan actions, while maintaining a valid partial order plan[1]. However, the best order of removal is not clear, and thus particular implementations may vary significantly in their performance.

In what follows, we propose a slightly different approach, exploiting the notion of *independence* between actions (Wehrle and Helmert 2012). The procedure simply follows the order of the actions in the plan, gathering (pairwise) independent actions into the set as long as possible. A new set is started every time a non-independent action is met. In what follows, we refer to this procedure as *Indep*. The complexity of this procedure is exponential in the size of the largest independent set of actions (Wehrle and Helmert 2012). Figure 2 depicts the graph constructed from a plan $\pi_0 = \text{pick}(b_1, L)\ \text{pick}(b_2, R)\ \text{move}(A, B)\ \text{drop}(b_1, L)\ \text{drop}(b_2, R)\ \text{move}(B, A)\ \text{pick}(b_3, L)\ \text{pick}(b_4, R)\ \text{move}(A, B)\ \text{drop}(b_3, L)\ \text{drop}(b_4, R)$ for the GRIPPER planning task with 4 balls, when reducing orders. The graph encodes 16 different plans obtained by reducing the order between $\text{pick}(b_1, L)$ and $\text{pick}(b_2, R)$, between $\text{drop}(b_1, L)$ and $\text{drop}(b_2, R)$, between $\text{pick}(b_3, L)$ and $\text{pick}(b_4, R)$, and between $\text{drop}(b_3, L)$ and $\text{drop}(b_4, R)$.

Note that some other valid plan reorderings, such as $\text{pick}(b_3, L)\ \text{pick}(b_4, R)\ \text{move}(A, B)\ \text{drop}(b_3, L)\ \text{drop}(b_4, R)\ \text{move}(B, A)\ \text{pick}(b_1, L)\ \text{pick}(b_2, R)\ \text{move}(A, B)\ \text{drop}(b_1, L)\ \text{drop}(b_2, R)$ are not obtained this way. In order to obtain all possible reorderings of a plan, one can use, e.g., a traversal procedure. In such a case, for a set P of partial order plans, the graph $G(P)$ is induced by possible sequentializations of the plans in P. Note that this can be exponential in the plan length and thus might turn out too expensive in practice. Since we are interested in a bounded number of solutions, which might be significantly smaller than the number of valid reorderings (e.g., in GRIPPER domain), in our experiments a depth first search traversal without duplicate detection was chosen. The algorithm is bound to apply each of the actions on the plan exactly once, and therefore complete. In addition, a cycle detection is performed on each path. The number of times the algorithm

[1] A partial order plan is valid if all its total order plans are valid.

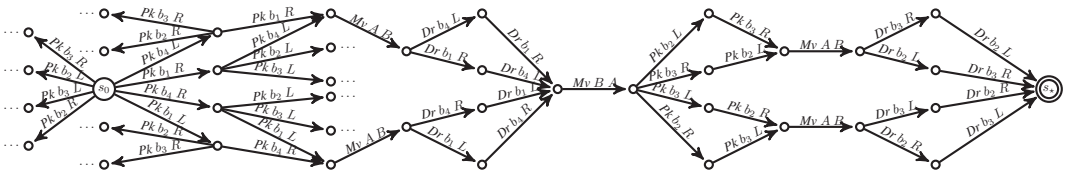

Figure 3: An extension with symmetries of the example plan for the GRIPPER task with 4 balls (a fragment).

has reached the goal is then used in the stopping criteria. In what follows, we refer to this procedure as *Naive*.

Symmetric Plans

One limitation of deriving plans based on reducing orders between actions is the restriction to the actions of the original plan. In what follows, we show how such a restriction can be alleviated. An additional way of deriving plans from already existing plans is by extending the graph $G(P)$, adding its symmetric counterparts in the state transition graph. This can be done using structural symmetries (Shleyfman et al. 2015). Structural symmetries are permutations of facts and actions that induce automorphisms of the state transition graph. Here, we present the definition of structural symmetries for SAS$^+$ as was given by Sievers et al. (2017).

Definition 4 (structural symmetry) *For a* SAS$^+$ *planning task* $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star, cost \rangle$*, let* F *be the set of* Π*'s facts, i.e., pairs* $\langle v, d \rangle$ *with* $v \in \mathcal{V}$*,* $d \in \mathcal{D}(v)$*. A* structural symmetry *for* Π *is a permutation* $\sigma : \mathcal{V} \cup F \cup \mathcal{O} \to \mathcal{V} \cup F \cup \mathcal{O}$*, where:*

1. $\sigma(\mathcal{V}) = \mathcal{V}$ *and* $\sigma(F) = F$ *such that* $\sigma(\langle v, d \rangle) = \langle v', d' \rangle$ *implies* $v' = \sigma(v)$*;*
2. $\sigma(\mathcal{O}) = \mathcal{O}$ *such that for* $o \in \mathcal{O}$*,* $\sigma(pre(o)) = pre(\sigma(o))$*,* $\sigma(\mathit{eff}(o)) = \mathit{eff}(\sigma(o))$*,* $cost(\sigma(o)) = cost(o)$*;*
3. $\sigma(s_\star) = s_\star$*;*

where $\sigma(\{x_1, \ldots, x_n\}) := \{\sigma(x_1), \ldots, \sigma(x_n)\}$*, and for a partial state* s*,* $s' := \sigma(s)$ *is the partial state obtained from* s *s.t. for all* $v \in \mathcal{V}(s)$*,* $\sigma(\langle v, s[v] \rangle) = \langle v', d' \rangle$ *implies* $s'[v'] = d'$*.*

We say that a structural symmetry σ *stabilizes a state* s if $\sigma(s) = s$. The set of all structural symmetries Γ_Π of a planning task Π forms a group under the composition operation. In what follows, for simplicity, by a symmetry group Γ we refer to a subgroup of the symmetry group Γ_Π of the planning task Π. In practice, a set of structural symmetries of a planning task (that stabilize a given state) can be efficiently computed using off-the-shelf tools for discovery of automorphisms in explicit graphs (Shleyfman et al. 2015).

We extend structural symmetries to sequences of actions in a natural way. Let σ be a structural symmetry that stabilizes the initial state s_0. Then, for a plan π of Π we have $\sigma(\pi)$ being a plan of Π with $cost(\sigma(\pi)) = cost(\pi)$. By P^Γ we denote the closure of P under Γ, i.e., $P^\Gamma = \{\sigma(\pi) \mid \sigma \in \Gamma, \pi \in P\}$.

Theorem 4 *Let* Π *be a planning task,* Γ *be a symmetry group of* Π *and* P *be a set of* Π*'s plans. If* P *is a solution to the top-*$|P|$ *planning problem, then* P^Γ *is a solution to the top-*$|P^\Gamma|$ *planning problem.*

Proof: Let $\pi \in P^\Gamma$ be some plan for Π. Since Γ is a group, there exists $\sigma \in \Gamma$ such that $\sigma(\pi) \in P$. Since P is a solution to the top-$|P|$ planning problem, if there exists a plan π' for Π with $cost(\pi') < cost(\pi) = cost(\sigma(\pi))$, then $\pi' \in P$ and thus $\pi' \in P^\Gamma$. □

Similarly, but more generally, we define the *closure of a graph* $G = (N, E)$ under Γ as a graph $G^\Gamma = (N', E')$, where $N' = N^\Gamma = \{\sigma(n) \mid \sigma \in \Gamma, n \in N\}$ the closure of N under Γ and $E' = \{(\sigma(n), \sigma(n')) \mid \sigma \in \Gamma, (n, n') \in E\}$.

Theorem 5 *Let* Π *be a planning task,* Γ *be a symmetry group of* Π *and* P *be a set of* Π*'s plans. Then* $G(P)^\Gamma = G(P^\Gamma)$*.*

Proof: First, we show that $G(P)^\Gamma \subseteq G(P^\Gamma)$. Let (u, v) be some edge in $G(P)^\Gamma$. Thus $u = \sigma(u')$ and $v = \sigma(v')$ for some $\sigma \in \Gamma$ and edge $(u', v') \in G(P)$. Let $\pi' \in P$ be some plan that traverses the edge (u', v'). Then $\pi = \sigma(\pi')$ is a plan, and it traverses (u, v). Further, π is in P^Γ, and thus $(u, v) \in G(P^\Gamma)$. For the other direction, let (u, v) be some edge in $G(P^\Gamma)$. Then, (u, v) is traversed by some plan $\pi \in P^\Gamma$. Thus, there exist $\sigma \in \Gamma$ and $\pi' \in P$ such that $\pi = \sigma(\pi')$. Thus, $(\sigma^{-1}(u), \sigma^{-1}(v))$ is traversed by the plan π', giving us $(\sigma^{-1}(u), \sigma^{-1}(v)) \in G(P)$. Therefore, by the definition of $G(P)^\Gamma$ we have $(u, v) \in G(P)^\Gamma$. □

Theorems 4 and 5 above allow us to safely extend the graph $G(P)$ with symmetries by taking its closure under the symmetries that stabilize the initial state. That way, methods extending a set of plans can include a reduction of plan actions order or a symmetry based extension or both.

Figure 3 depicts (part of) the graph obtained by extending the example gripper task plan with structural symmetries. The structural symmetries found on this task are between balls and between arms, i.e., all balls are symmetric to each other, and the two arms are symmetric. Note though, that not all optimal plans are symmetric to each other. For instance, the example plan π_0 = pick(b_1, L) pick(b_2, R) move(A, B) drop(b_1, L) drop(b_2, R) move(B, A) pick(b_3, L) pick(b_4, R) move(A, B) drop(b_3, L) drop(b_4, R) is not symmetric to the plan π_1 = pick(b_1, L) pick(b_2, R) move(A, B) drop(b_1, L) drop(b_2, R) move(B, A) pick(b_3, L) pick(b_4, R) move(A, B) drop(b_4, R) drop(b_3, L) (the last two actions reordered), since there is no composition of symmetries that permute balls and arms that can map between the two plans. However, when permuting

ball3 with ball4 and arms together, we obtain the plan $\pi_2 = \text{pick}(b_1, R)\ \text{pick}(b_2, L)\ \text{move}(A, B)\ \text{drop}(b_1, R)\ \text{drop}(b_2, L)\ \text{move}(B, A)\ \text{pick}(b_4, R)\ \text{pick}(b_3, L)\ \text{move}(A, B)\ \text{drop}(b_4, R)\ \text{drop}(b_3, L)$. Both this π_2 and π_0, if followed through up to the last two actions, end up in the same state. Thus, adding both π_0 and π_2 to $G(P)$, allows us to extract π_1, as it follows π_0 for the first nine actions and π_2 for the last two. All optimal plans for the example task can be obtained this way and therefore the graph encodes all optimal plans for this task.

Repeatedly Forbidding Multiple Plans

Having introduced the graph $G(P)$ as compact representation of a set of plans P and having clarified how to extract plans from $G(P)$, we now devise a reformulation that forbids all plans represented by $G(P)$.

Definition 5 ($G(P)$-forbidding reformulation) *Let Π be a planning task over actions $\mathcal{O}$ and P be some set of plans for Π. A task Π' over actions $\mathcal{O}'$ is a $G(P)$-forbidding reformulation of Π if there exists a mapping of actions $r : \mathcal{O}' \mapsto \mathcal{O}$, and its extension r' such that π' is a plan for Π' iff $r'(\pi')$ is a plan for Π with $cost'(\pi') = cost(r'(\pi'))$ and $r'(\pi') \notin G(P)$.*

In what follows, we sometimes abuse the notation and treat $G(P)$ as the set of plans that can be derived from $G(P)$, denoting such plans by $\pi \in G(P)$. Further, by $\mathcal{O}(G(P))$ we denote the set of all actions in P. These are exactly the actions labeling the edges of $G(P)$. Finally, by E^o we denote the subset of edges that are induced by the action o, and each action instance that corresponds to the edge (s, t) is denoted by $o_{(s,t)}$. We now extend the plan forbidding reformulation presented in the previous section to forbid all plans in $G(P)$.

Definition 6 *Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star, cost\rangle$ be a planning task, P be some set of plans, and $G := G(P) = (N, E)$. The task $\Pi^-_G = \langle \mathcal{V}', \mathcal{O}', s'_0, s'_\star, cost'\rangle$ is defined as follows.*

- $\mathcal{V}' = \mathcal{V} \cup \{\overline{v}\} \cup \{\overline{v}_s \mid s \in N\}$*, with $\overline{v}_s$ being binary variables,*
- $\mathcal{O}' = \{o^e \mid o \in \mathcal{O} \setminus \mathcal{O}(G)\} \cup \bigcup_{o \in \mathcal{O}(G)} \{o^1, o^2\} \cup \bigcup_{(s,t) \in E} \{o^3_{(s,t)}\}$,

  where

  $o^e = \langle pre(o), \mathit{eff}(o) \cup \{\langle \overline{v}, 0\rangle\}\rangle$

  $o^1 = \langle pre(o) \cup \{\langle \overline{v}, 0\rangle\}, \mathit{eff}(o)\rangle$

  $o^2 = \langle pre(o) \cup \{\langle \overline{v}, 1\rangle\} \cup \{\langle \overline{v}_s, 0\rangle \mid (s, t) \in E^o\}, \mathit{eff}(o) \cup \{\langle \overline{v}, 0\rangle\}\rangle$

  $o^3_{(s,t)} = \langle pre(o_{(s,t)}) \cup \{\langle \overline{v}, 1\rangle, \langle \overline{v}_s, 1\rangle\}, \mathit{eff}(o_{(s,t)}) \cup \{\langle \overline{v}_s, 0\rangle, \langle \overline{v}_t, 1\rangle\}\rangle$, *and*

  $cost'(o^e) = cost'(o^1) = cost'(o^2) = cost'(o^3) = cost(o)$,
- $s'_0[v] = s_0[v]$ *for all* $v \in \mathcal{V}$, $s'_0[\overline{v}] = 1$, $s'_0[\overline{v}_{s_0}] = 1$, *and* $s'_0[\overline{v}_s] = 0$ *for all* $s \in N \setminus \{s_0\}$, *and*
- $s'_\star[v] = s_\star[v]$ *for all* $v \in \mathcal{V}$ *s.t.* $s_\star[v]$ *defined, and* $s'_\star[\overline{v}] = 0$.

The planning task Π^-_G forbids all sequences of actions $\pi \in G$ from being a plan, accepting all other plans of Π. Similarly to the case of a single plan, the variable $\overline{v}$ starts from the value 1 and switches to 0 when the sequence of actions applied is not a prefix of π. Once a value 0 is reached, it cannot be switched back to 1. The actions o^1 and o^2 are the copies of actions in π for the cases when π is already discarded from consideration (variable $\overline{v}$ has already switched its value to 0) and for discarding π from consideration (switching $\overline{v}$ to 0). In case there are multiple appearances of an action o in G, o^2 requires all the corresponding variables $\overline{v}_s$ for each edge (s, t) in E that is induced by o to have the value 0. Thus, o^2 will be applicable only in states that are different from the states resulting from applying the prefix of some plan $\pi \in G$ up to o in the initial state. $o^3_{(s,t)}$ are copies of the corresponding action o along G, with a separate copy for each appearance of the same action o. These actions are responsible for following sequences $\pi \in G$ and are applicable while some such sequence is still followed.

Theorem 6 *Let Π be a planning task, P be some set of plans, and $G := G(P) = (N, E)$. The task Π^-_G is a G-forbidding reformulation of Π.*

Proof: Let $r : \mathcal{O}' \mapsto \mathcal{O}$ be the mapping defined by $r(o^e) = o$ and $r(o^1) = r(o^2) = r(o^3) = o$. Note that Π^-_G restricted to the variables $\mathcal{V}$ equals to the task Π, modulo the three equal instances of the actions in G. Thus, for each plan π for Π^-_G, $r(\pi)$ is a plan for Π.

For the second direction, since for each $o \in G$ at most one of the actions o^1, o^2, o^3 is applicable in each state s of Π^-_G, given a sequence of actions ρ applicable in the initial state of Π, it can be mapped to an applicable in the initial state of Π^-_G sequence of action ρ' such that $r(\rho') = \rho$, by choosing in each state the relevant representative out of o^e, o^1, o^2, and o^3. In other words, r restricted to applicable in the initial state sequences of actions is invertible, and we denote its inverse mapping described above by r^{-1}.

First, let $\pi = o_1 \dots o_n$ be some plan in G and let $\pi' = r^{-1}(\pi)$ be the inverse of π. Then $\pi' = s'_0 o^3_1 s_1 \dots s_{n-1} o^3_n s_n$, since at step i we have $\overline{v} = 1$ and $\overline{v}_{s_{i-1}} = 1$. Thus, after applying π', the value of the variable $\overline{v}$ in s_n remains 1, and thus π' is not a plan for Π^-_G.

Now, let ρ be a plan for Π such that $\rho \notin G$. Let o be the first action on ρ that does not follow a path in G. In other words, there exists a prefix $\rho' = s_0 o_1 s_1 \dots s_{m-1} o_m s_m$ of ρ such that (i) $\rho' o$ is a prefix of ρ, (ii) ρ' is a path in G, and (iii) $\rho' o$ is not a path in G. Then we have $r^{-1}(\rho') = o^3_1 \dots o^3_m$, and since $o \neq o_{m+1}$, the next action on $r^{-1}(\rho)$ will not be o^3_{m+1}. If $o \in G$, then the next action will be o^2 (which we next show applicability of), and otherwise it will be o^e, in both cases setting the value of $\overline{v}$ to 0. Thus, all the following actions o' are mapped to either o'^e or o'^1, and the preconditions of these actions are restricted to $\mathcal{V}$ and $\overline{v} = 0$, the sequence of actions $r^{-1}(\rho)$ achieves the goal values on $\mathcal{V}$ and thus is a plan.

We finalize the proof by showing the applicability of o^2 in the state $s'_m := s'_0[\![o^3_1 \dots o^3_m]\!]$ for $o \in G$ such that o does not correspond to any edge (s_m, t). Naturally, $pre(o)$ holds in

Algorithm 2: IterativeTopKMultiple($\langle \Pi, k \rangle$)

$P \leftarrow \emptyset$, $T \leftarrow \emptyset$, $B \leftarrow \emptyset$, $r \leftarrow id$
while $|T| < k$ *and no failure occurs* **do**
  $\Pi, r' \leftarrow$ GRAPHFORBIDREFORMULATION($\Pi, G(P)$)
  $r \leftarrow r \circ r'$
  $\pi \leftarrow$ GETOPTIMALPLAN(Π)
  if GETOPTIMALPLAN(Π) *reports* UNSOLVABLE **then**
    return SETTOPELEMENTS($T \cup B, k$)
  end
  $P \leftarrow$ EXTENDPLAN(Π, π)
  $T_P \leftarrow \{r(\pi') \mid \pi' \in G(P), cost(\pi') = cost(\pi)\}$
  $B_P \leftarrow \{r(\pi') \mid \pi' \in G(P), cost(\pi') > cost(\pi)\}$
  $T \leftarrow T \cup T_P \cup \{\pi' \in B \mid cost(\pi') = cost(\pi)\}$
  $B \leftarrow B \setminus T \cup B_P$
end
return SETTOPELEMENTS(T, k)

s'_m. Further, since $s'_0[\overline{v}] = 1$, $s'_0[\overline{v}_{s_0}] = 1$, and $s'_0[\overline{v}_s] = 0$ for all $s \in N \setminus \{s_0\}$, after applying $o^3_1 \dots o^3_m$ we have $s'_m[\overline{v}] = 1$, $s'_m[\overline{v}_{s_m}] = 1$, and $s'_m[\overline{v}_s] = 0$ for all $s \in N \setminus \{s_m\}$. Since o does not correspond to any edge (s_m, t), for each $(s, t) \in E^o$ we have $s'_m[\overline{v}_s] = 0$, and thus o^2 is applicable in s'_m. □

The scheme, depicted in Algorithm 2, works as follows. Once a plan is found, it is extended to a set of plans P and then to the graph $G(P)$, which is forbidden in the next iteration. Further, the plans encoded by $G(P)$ are extracted and partitioned into two sets, optimal plans T and non-optimal ones B. In the next iterations, the set T is extended with optimal plans T' from that iteration, as well as all plans of the same cost as those in T' from the set B. The algorithm is thus iterating until the set T consists of at least k plans or no more plans exist. In practice, however, the algorithm can be simplified if the graph $G(P)$ encodes only plans of the same cost.

Experimental Evaluation

In order to empirically evaluate the feasibility of our approach to finding top-k plans, we implemented Algorithm 2 on top of the Fast Downward planning system (Helmert 2006), extended with the support for structural symmetries and the orbit space search algorithm (Domshlak, Katz, and Shleyfman 2015; Alkhazraji et al. 2014). As an underlying classical planner we used an orbit space search, not stabilizing the initial state, with the LM-cut heuristic (Helmert and Domshlak 2009), a state-of-the-art heuristic search planner. A natural candidate as a baseline for the comparison would be the K^* algorithm. Unfortunately, the only existing (to our knowledge) implementation of the K^* algorithm for classical planning supports only the SPPL language. Therefore, in order to be able to compare to an existing method for deriving top-k plans, we also implemented K^* search within the Fast Downward planning system. As K^* requires a consistent heuristic, we used the iPDB heuristic (Haslum et al. 2007). Since iPDB may require a long pre-search computation time, we also experimented with the blind heuristic.

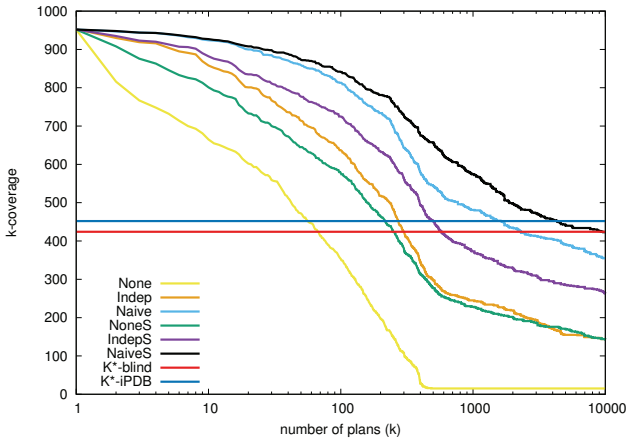

Figure 4: The k-coverage as a function of the restriction on the number of required plans, for $k \leq 10000$.

Our benchmark suite includes all 1667 tasks from the *optimal* IPC STRIPS benchmarks. The experiments were performed on Intel(R) Xeon(R) CPU E7-8837 2.67GHz, with time (memory) limit of 30min (2GB).

To measure the effect of various techniques for extending the set of existing plans, we switched the extension by symmetry on and off (adding "S" to the configuration name if symmetry is switched on), and experimented with the two aforementioned approaches for plan reordering (Indep and Naive), as well as not reordering at all (None), giving us in total six configurations for the iterative approach and two configurations for the K^* approach. The experiments were performed with a large k value, namely $k = 10000$, aiming at checking the feasibility of the suggested approaches for finding a large set of plans.

For a given task and a natural number k, the k-coverage is a value in $\{0, 1\}$ assigning the value 1 if a solution to the top-k planning problem was found (or proven unsolvable), according to Definition 1, and 0 otherwise. In order to measure the effect of the number of required plans k, we computed the k-coverage for any value of k in $[1, 10000]$, without rerunning the experiments for that value of k. We also experimented with a small value of $k = 10$, obtaining similar results for all configurations. We also note that for the iterative configurations, there are only up to 10 tasks for each configuration that fail on memory, with most failures being due to timeouts. For K^* based configurations, the opposite is true, with 12 and 68 timeouts for blind heuristic and iPDB, respectively.

Figure 4 depicts the results for all configurations, summed over all tasks in our benchmark suite. The horizontal lines correspond to K^* with iPDB heuristic (top) and the blind heuristic (bottom). Both K^* configurations found all required plans within a short time window, and thus always either fail to find any plan or find all required ones. In contrast, the iterative approaches exhibit an anytime behaviour. Further, since the iterative configurations exploit a better underlying classical planner, the k-coverage for smaller values of k is significantly higher for the iterative approaches than

	k=10		k=100		k=1000		k=10000			
Coverage	IndepS	NaiveS	IndepS	NaiveS	IndepS	NaiveS	IndepS	NaiveS	K^*_{bl}	K^*_{iPDB}
airport	28 (2)	28 (1)	22 (18)	26 (1)	14 (1)	26 (1)	11 (1)	26 (1)	16	11
barman11	8 (1)	8 (1)	8 (1)	8 (1)	1 (1)	8 (1)	0 (0)	0 (0)	4	4
barman14	3 (1)	3 (1)	3 (1)	3 (1)	3 (1)	3 (1)	0 (0)	3 (1)	0	0
blocks	22 (9)	27 (5)	13 (99)	16 (78)	0 (0)	0 (0)	0 (0)	0 (0)	18	18
childsnack14	6 (1)	6 (1)	6 (1)	6 (1)	6 (1)	6 (1)	6 (1)	6 (1)	0	0
depot	9 (1)	9 (1)	6 (20)	9 (9)	3 (1)	8 (34)	2 (1)	7 (1)	3	3
driverlog	13 (1)	13 (1)	13 (15)	13 (6)	2 (1)	11 (2)	0 (0)	7 (1)	4	4
elevators08	22 (2)	22 (1)	8 (18)	19 (6)	3 (2)	18 (1)	0 (0)	14 (1)	4	4
elevators11	18 (2)	18 (1)	6 (7)	15 (1)	3 (2)	15 (1)	0 (0)	12 (1)	2	2
floortile11	8 (1)	8 (1)	8 (1)	8 (1)	4 (1)	8 (1)	0 (0)	8 (1)	0	0
floortile14	8 (1)	8 (1)	8 (1)	8 (1)	6 (1)	8 (1)	0 (0)	8 (1)	0	0
freecell	13 (2)	14 (1)	6 (75)	11 (36)	0 (0)	2 (1)	0 (0)	0 (0)	13	13
ged14	13 (8)	13 (8)	5 (96)	5 (94)	0 (0)	0 (0)	0 (0)	0 (0)	10	10
grid	1 (9)	1 (9)	1 (99)	1 (99)	0 (0)	0 (0)	0 (0)	0 (0)	0	0
gripper	20 (1)	20 (1)	20 (1)	20 (1)	20 (1)	20 (1)	20 (2)	20 (1)	6	6
hiking14	13 (1)	13 (1)	13 (1)	13 (1)	13 (8)	13 (1)	6 (1)	11 (1)	6	6
logistics00	20 (1)	20 (1)	20 (5)	20 (1)	17 (1)	20 (11)	16 (1)	17 (1)	10	10
logistics98	6 (1)	6 (1)	6 (13)	6 (1)	2 (1)	6 (1)	2 (1)	5 (1)	1	1
miconic	142 (1)	142 (1)	142 (13)	141 (13)	102 (6)	79 (5)	78 (2)	37 (4)	40	40
movie	30 (1)	30 (1)	30 (1)	30 (1)	30 (4)	30 (1)	30 (76)	30 (6)	30	30
mprime	22 (7)	22 (6)	13 (88)	13 (81)	0 (0)	0 (0)	0 (0)	0 (0)	1	1
mystery	19 (6)	19 (6)	15 (61)	15 (58)	4 (1)	4 (1)	4 (1)	4 (1)	5	8
nomystery11	15 (1)	15 (1)	15 (14)	15 (12)	3 (26)	5 (6)	1 (1)	1 (1)	7	9
openstacks08	24 (1)	24 (1)	24 (2)	24 (1)	23 (2)	23 (1)	18 (1)	22 (1)	14	17
openstacks11	19 (1)	19 (1)	19 (1)	19 (1)	19 (1)	19 (1)	16 (1)	19 (1)	9	12
openstacks14	5 (1)	5 (1)	5 (1)	5 (1)	4 (1)	5 (1)	4 (1)	5 (1)	1	1
openstacks	5 (1)	7 (1)	5 (1)	6 (1)	1 (10)	5 (1)	0 (0)	5 (1)	7	7
parcprinter08	18 (2)	18 (1)	17 (30)	18 (6)	5 (2)	17 (1)	4 (2)	16 (2)	7	13
parcprinter11	13 (2)	13 (1)	12 (30)	13 (1)	3 (1)	13 (1)	2 (1)	12 (2)	3	9
parking11	1 (9)	1 (9)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0	0
parking14	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0	0
pathways-nn	5 (1)	5 (1)	5 (14)	5 (2)	4 (42)	5 (28)	2 (1)	3 (1)	3	3
pegsol08	28 (7)	28 (4)	15 (66)	20 (47)	5 (73)	7 (51)	5 (73)	5 (72)	26	26
pegsol11	18 (6)	18 (3)	5 (44)	9 (17)	0 (0)	3 (1)	0 (0)	0 (0)	16	16
pipes-notank	20 (2)	21 (1)	17 (18)	19 (8)	1 (87)	12 (16)	0 (0)	6 (1)	6	6
pipes-tank	16 (2)	16 (1)	14 (16)	15 (15)	11 (10)	12 (8)	5 (6)	8 (6)	6	6
psr-small	49 (3)	49 (2)	43 (42)	49 (28)	7 (4)	21 (2)	1 (1)	11 (1)	43	43
rovers	7 (1)	7 (1)	6 (30)	7 (3)	0 (0)	5 (84)	0 (0)	3 (1)	4	4
satellite	13 (1)	13 (1)	11 (18)	13 (12)	4 (1)	8 (1)	3 (1)	8 (1)	4	4
scanalyzer08	16 (2)	16 (1)	16 (21)	16 (16)	9 (1)	13 (5)	7 (1)	8 (1)	6	6
scanalyzer11	13 (1)	13 (1)	13 (11)	13 (7)	8 (1)	12 (5)	7 (1)	8 (1)	3	3
sokoban08	14 (6)	29 (2)	6 (75)	23 (20)	0 (0)	11 (1)	0 (0)	10 (1)	15	21
sokoban11	9 (6)	19 (2)	3 (52)	14 (11)	0 (0)	6 (1)	0 (0)	5 (1)	12	17
storage	17 (2)	17 (2)	17 (23)	17 (20)	3 (19)	7 (16)	0 (0)	5 (2)	11	11
tetris14	8 (1)	8 (1)	5 (22)	8 (6)	0 (0)	7 (1)	0 (0)	6 (1)	5	1
tidybot11	11 (1)	14 (1)	3 (68)	14 (5)	0 (0)	10 (1)	0 (0)	0 (0)	1	1
tidybot14	4 (1)	9 (1)	0 (0)	9 (1)	0 (0)	6 (1)	0 (0)	1 (1)	0	0
tpp	7 (2)	7 (2)	7 (19)	7 (18)	5 (7)	5 (8)	4 (1)	4 (1)	5	5
transport08	9 (5)	9 (4)	5 (58)	7 (32)	0 (0)	3 (7)	0 (0)	1 (1)	8	8
transport11	4 (6)	5 (4)	0 (0)	3 (1)	0 (0)	3 (1)	0 (0)	1 (1)	3	3
transport14	2 (5)	4 (3)	1 (22)	2 (1)	0 (0)	2 (1)	0 (0)	1 (1)	3	3
trucks	12 (1)	12 (1)	12 (14)	12 (10)	5 (1)	8 (1)	1 (1)	5 (1)	4	4
visitall11	9 (6)	9 (6)	9 (68)	9 (68)	1 (124)	1 (124)	0 (0)	0 (0)	8	8
visitall14	4 (7)	4 (7)	1 (13)	1 (13)	0 (0)	0 (0)	0 (0)	0 (0)	2	2
woodwork08	18 (1)	18 (1)	18 (15)	18 (3)	8 (10)	16 (1)	4 (1)	13 (1)	4	6
woodwork11	12 (1)	12 (1)	12 (4)	12 (1)	7 (11)	12 (1)	3 (1)	12 (1)	0	1
zenotravel	13 (3)	13 (3)	12 (43)	13 (39)	4 (57)	7 (33)	1 (1)	4 (1)	5	5
Sum	882	927	725	841	373	574	263	423	424	452

Table 1: Per-domain k-coverage for selected values of k. Average number of iterations in parentheses.

for the K^* ones. Looking at the overall k-coverage, we can see a clear dominance of *NaiveS* over all other configurations, up to $k = 4320$. For $k > 4320$, K^* with iPDB has the best overall k-coverage. For specific domains, however, the picture might differ substantially.

Table 1 shows the per-domain k-coverage results on selected configurations, namely *IndepS* and *NaiveS* for the iterative approach and both K^* configurations for three values of k, namely $k = 10, 100, 1000, 10000$. The first two columns depict the k-coverage of the two iterative configurations for $k = 10$, the next two for $k = 100$ and the next two for $k = 1000$. The last block of four columns shows the k-coverage of all four selected configurations for $k = 10000$. As mentioned above, the K^* configurations (two rightmost columns) perform similarly with all values of $k \leq 10000$ and thus are shown once.

First, looking at the rightmost part, corresponding to $k = 10000$, note that there is no clear advantage to either of the approaches across the domains. K^* achieves better performance in 23 domains out of 57, while the iterative approach performs better in 28 domains. Further, there is often a large difference in coverage between the two approaches, to one side and to another, making them complementary. Within each approach, there is a clear advantage to one of the configurations. For the iterative approach, with the exception of the MICONIC domain, *NaiveS* performs at least as good as *IndepS*, performing strictly better in 37 domains. This dominance is preserved for smaller k values, with strict dominance in 39 domains for $k = 1000$, in 25 domains for $k = 100$, and 10 for $k = 10$. *NaiveS* loses to *IndepS* only in MICONIC domain, for $k > 10$. For the K^* based approach, there are two domains where the blind search performs better than iPDB, namely AIRPORT and TETRIS, and performs strictly worse in 10 domains. Note that there are 45 domains with equal k-coverage for the two K^* based approaches.

Moving on to smaller k values, note that for $k = 1000$ the dominance shifts further towards the iterative approach, with 34 domains vs. 17 domains for K^*. For $k = 100$ it becomes 43 domains vs. 10 domains. For $k = 10$, the iterative approach always performs at least as good as K^*, achieving better performance on 54 out of 57 domains.

Looking at some non-IPC domains (Sohrabi, Udrea, and Riabov 2013; Riabov, Sohrabi, and Udrea 2014; Sohrabi et al. 2018), we note that these were modelled with planner efficiency in mind. For the iterative approach, the number of plans found and forbidden per iteration is typically 1, and therefore the number of plans found within the time and memory restrictions is typically low. K^*, on the other hand, even with the blind heuristic performs extremely well on these domains, solving all tasks for $k = 10000$ in all but the risk management domain (Sohrabi et al. 2018). On this domain, it finds all 10000 requested top plans in 133 out of 200 tasks, failing to find any plans on the other 67 tasks.

Conclusions and Future Work

We have presented a novel approach to the problem of top-k planning, based on iterative computation of optimal solutions, exploiting existing optimal planners. We proposed two techniques for deriving additional solutions from previously found ones. To empirically evaluate the feasibility of our suggested approach, we have implemented an existing approach to top-k planning, K^*, on top of a state-of-the-art planner. We show that these techniques are complementary in their performance, with K^* being more beneficial for extremely large values of k.

In the future work, we intend to focus on both approaches. For the iterative approach, other methods for deriving additional plans given already existing plans can lead to considerable further performance improvement. For the K^* based approach, we intend to explore several directions. First, to investigate the adaptation of the search algorithm to work with state-of-the-art non-consistent admissible heuristics, such as LM-cut. Second, we would like to explore the usage of search pruning techniques, such as partial order reduction and symmetry reduction for top-k planning, ensuring

that relevant solutions are not pruned. Finally, some effort should be invested in exploring the high memory consumption of K^* and ways to overcome it.

References

Aljazzar, H., and Leue, S. 2011. K*: A heuristic search algorithm for finding the k shortest paths. *Artificial Intelligence* 175(18):2129–2154.

Alkhazraji, Y.; Katz, M.; Mattmüller, R.; Pommerening, F.; Shleyfman, A.; and Wehrle, M. 2014. Metis: Arming Fast Downward with pruning and incremental computation. In *the 8th International Planning Competition (IPC-8): planner abstracts*, 88–92.

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

Bäckström, C. 1998. Computational aspects of reordering plans. *Journal of Artificial Intelligence Research* 9:99–137.

Bryce, D. 2014. Landmark-based plan distance measures for diverse planning. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 56–64.

Coman, A., and Muñoz-Avila, H. 2011. Generating diverse plans using quantitative and qualitative plan distance metrics. In *Proceedings of the 25th National Conference on Artificial Intelligence (AAAI)*, 946–951.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2015. Symmetry breaking in deterministic planning as forward search: Orbit space search algorithm. Technical Report IS/IE-2015-03, Technion, Haifa.

Fox, M.; Gerevini, A.; Long, D.; and Serina, I. 2006. Plan stability: Replanning versus plan repair. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*, 212–221.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI 2007)*, 1007–1012.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Keyder, E., and Geffner, H. 2009. Soft Goals Can Be Compiled Away. *Journal of Artificial Intelligence Research* 36:547–556.

Nguyen, T.; Do, M.; Gerevini, A.; Serina, I.; Srivastava, B.; and Kambhampati, S. 2012. Generating diverse plans to handle unknown and partially known user preferences. *Artificial Intelligence* 190:1–31.

Riabov, A., and Liu, Z. 2005. Planning for stream processing systems. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, 1205–1210.

Riabov, A. V.; Sohrabi, S.; and Udrea, O. 2014. New algorithms for the top-k planning problem. In *ICAPS 2014 Scheduling and Planning Applications woRKshop*, 10–16.

Shleyfman, A.; Katz, M.; Helmert, M.; Sievers, S.; and Wehrle, M. 2015. Heuristics and symmetries in classical planning. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3371–3377.

Sievers, S.; Wehrle, M.; Helmert, M.; and Katz, M. 2017. Strengthening canonical pattern databases with structural symmetries. In *Proceedings of the 10th Annual Symposium on Combinatorial Search (SoCS 2017)*, 91–99.

Sohrabi, S.; Riabov, A.; Udrea, O.; and Hassanzadeh, O. 2016. Finding diverse high-quality plans for hypothesis generation. In *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI)*, 1581–1582.

Sohrabi, S.; Riabov, A. V.; Katz, M.; and Udrea, O. 2018. An AI planning solution to scenario generation for enterprise risk management. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018)*.

Sohrabi, S.; Riabov, A.; and Udrea, O. 2016. Plan recognition as planning revisited. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, 3258–3264.

Sohrabi, S.; Udrea, O.; and Riabov, A. 2013. Hypothesis exploration for malware detection using planning. In *Proceedings of the 27th National Conference on Artificial Intelligence (AAAI)*, 883–889.

Wehrle, M., and Helmert, M. 2012. About partial order reduction in planning and computer aided verification. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 297–305.