Symbolic Top-k Planning

David Speck and **Robert Mattmüller** and **Bernhard Nebel**
University of Freiburg, Germany
{speckd, mattmuel, nebel}@informatik.uni-freiburg.de

Abstract

The objective of top-k planning is to determine a set of k different plans with lowest cost for a given planning task. In practice, such a set of best plans can be preferred to a single best plan generated by ordinary optimal planners, as it allows the user to choose between different alternatives and thus take into account preferences that may be difficult to model. In this paper we show that, in general, the decision problem version of top-k planning is PSPACE-complete, as is the decision problem version of ordinary classical planning. This does not hold for polynomially bounded plans for which the decision problem turns out to be PP-hard, while the ordinary case is NP-hard. We present a novel approach to top-k planning, called SYM-K, which is based on symbolic search, and prove that SYM-K is sound and complete. Our empirical analysis shows that SYM-K exceeds the current state of the art for both small and large k.

Introduction

The objective of cost-optimal planning is to find a single plan, which is a sequence of actions that leads to a goal state with minimal cumulative costs. The problem of determining not only a single best plan, but a set of the k best plans is called top-k planning (Katz et al. 2018). While an optimal plan may be sufficient in some cases, in practice it is often better to have many good alternative plans. A variety of good alternative plans makes it possible to take into account user preferences and environmental influences that are difficult to model or may have changed at the time the plan is executed. A top-k planner also allows to "generate and test" high quality plans, which is relevant for various areas, such as goal recognition (Sohrabi, Riabov, and Udrea 2016), diverse planning (Katz and Sohrabi 2019), morally permissible planning (Lindner, Mattmüller, and Nebel 2019), or explanation generation (Eifler et al. 2019). In addition, collections of plans for planning tasks can serve as practical training sets for machine learning algorithms (Toyer et al. 2018; Gnad et al. 2019) and enable empirical studies on properties of different planning tasks (Corraya et al. 2019).

Although the problem of determining k *shortest* paths for a given graph is a well studied topic going back to 1957 (Bock, Kanter, and Haynes 1957), the problem of determining k *cheapest* plans for a given planning task has more recently been raised (Riabov, Sohrabi, and Udrea 2014). There are several approaches to the k shortest paths problem, such as a generalization of A* (Hart, Nilsson, and Raphael 1968), called K* (Aljazzar and Leue 2011), which partially generates and processes parts of the input graph. For top-k planning, there are currently two different approaches: K* and FORBID-K (Katz et al. 2018). While K* can be applied to top-k planning without major changes, FORBID-K is a novel iterative approach based on a replanning loop that forbids already found plans and preserves all other plans. It turns out that FORBID-K dominates K* if only a small number of plans are desired. However, the replanning is too expensive if a larger number of plans is desired, which leads to a large decrease in performance. This raises the question whether there are approaches that are efficient for small k and also scale well to larger k.

Over the past decade, symbolic search has proven to be a competitive approach to cost-optimal planning (Edelkamp, Kissmann, and Torralba 2015). In contrast to explicit search, in which individual states are generated and expanded, symbolic search operates on whole state sets. Interestingly, symbolic planning often finds not only one (optimal) plan, but several at once. However, only one of these plans is reported and all other plans are ignored (Torralba 2015). To the best of the authors' knowledge, symbolic planning has never before been used for top-k planning. However, Günther, Schuster, and Siegle (2010) proposed a symbolic search based on Binary Decision Diagrams (Bryant 1986) to determine the most probable paths of a given graph with transition probabilities. The approach of Günther, Schuster, and Siegle (2010) explores the entire state space to determine these paths and is closely related to the k shortest path problem of graphs.

This paper has a theoretical and a practical contribution to top-k planning. On the theoretical side, we answer the question whether top-k planning is computationally harder than ordinary planning. On the practical side, we introduce a new symbolic approach to top-k planning and prove soundness and completeness. An empirical study on various planning domains shows that the algorithm presented is a top-k planner that exceeds the current state of the art for both small and large values of k.

Preliminaries

We consider classical planning tasks that are characterized by the SAS$^+$ formalism (Bäckström and Nebel 1995).

Definition 1 (Classical planning task). A *classical planning task* is a tuple $\Pi = \langle \mathcal{V}, \mathcal{I}, \mathcal{O}, \mathcal{G} \rangle$ consisting of four components. $\mathcal{V}$ is a finite set of state variables, each associated with a finite domain D_v. A fact is a pair (v, d), where $v \in \mathcal{V}$ and $d \in D_v$, and a partial variable assignments over $\mathcal{V}$ is a consistent set of facts. If s assigns a value to each $v \in \mathcal{V}$, s is called a state. States and partial variable assignments are functions which map variables to values, i.e., $s(v)$ is the value of variable v in state s (analogous for partial variable assignments). $\mathcal{O}$ is a set of operators, where an operator is a pair $o = \langle pre_o, \mathit{eff}_o \rangle$ of partial variable assignments called preconditions and effects, respectively. Each operator has non-negative cost $c_o \in \mathbb{N}_0$. The state $\mathcal{I}$ is called the initial state and the partial variable assignment $\mathcal{G}$ specifies the goal condition, which defines all possible goal states $S_\star$. With $\mathcal{S}$ we refer to the set of all states defined over $\mathcal{V}$, and with $|\Pi|$ we refer to the size of planning task Π, i.e., the number of operators and facts.

We call an operator $o \in \mathcal{O}$ applicable in state s iff pre_o is satisfied in s, i.e., $s \models pre_o$. Applying operator o in state s results in a state s' where $s'(v) = \mathit{eff}_o(v)$ for all variables $v \in \mathcal{V}$ for which eff_o is defined and $s'(v) = s(v)$ for all other variables. We also write $s[o]$ for s'. The objective of ordinary classical planning is to determine a plan, which is defined as follows.

Definition 2 (Plan). A *plan* $\pi = \langle o_0, \ldots, o_{n-1} \rangle$ for planning task Π is a sequence of applicable operators which generates a sequence of states $s_0, \ldots, s_n$, where $s_0 = \mathcal{I}$, $s_n \in S_\star$ is a goal state and $s_{i+1} = s_i[o_i]$ for all $i = 0, \ldots, n-1$. The cost of plan π is the sum of its operator costs. Plan π is optimal if there is no cheaper plan. With P_Π we refer to the (possibly infinite) set of all plans for a planning task Π.

Top-k Planning

The objective of top-k planning is to determine a set of k different plans $P \subseteq P_\Pi$ with lowest costs for a planning task Π. It is important that there exists no plan $\pi \in P_\Pi$ for Π that is not included in P and has lower costs than some plan in P. We want to point out that ordinary classical planning is an important special case of top-k planning ($k = 1$). Similar to Katz et al. (2018), we formally define top-k planning as follows.

Definition 3 (Top-k planning). Given a planning task Π and a natural number k, top-k planning is the problem of determining a set of plans $P \subseteq P_\Pi$ such that:

1. there exists no plan $\pi' \in P_\Pi$ with $\pi' \notin P$ that is cheaper than some plan $\pi \in P$, and
2. $|P| = k$ if $|P_\Pi| \geq k$, and $|P| = |P_\Pi|$, otherwise.

We call an algorithm A sound for top-k planning iff algorithm A reports only valid plans and satisfies condition (1.) of Definition 3. Note that this definition of soundness already implies optimality. Intuitively this makes sense, because a top-k planner should only report *top* plans. Alternatively, it would be possible to define soundness without optimality to describe algorithms which find k different but not necessarily optimal plans. Similarly, we say that an algorithm A is complete for top-k planning iff algorithm A terminates and satisfies condition (2.) of Definition 3.

Symbolic Search

Symbolic versions of search algorithms resemble their explicit counterparts, but expand and generate whole sets of states in contrast to individual states. In recent years, symbolic planning has become a competitive alternative to explicit optimal classical planning (Edelkamp, Kissmann, and Torralba 2015). In symbolic planning, a set of states $S \subseteq \mathcal{S}$ is represented by its corresponding characteristic function χ_S, which is a Boolean function $\chi_S : \mathcal{S} \mapsto \{\top, \bot\}$. More precisely, states contained in S are mapped to $\top$ and all others to $\bot$, i.e., $\chi_S(s) = \top$ if $s \in S$ and $\chi_S(s) = \bot$ otherwise. Usually such functions are represented by compact and efficient data structures such as Binary Decision Diagrams, also known as BDDs (Bryant 1986). Operators can be represented as transition relations (TRs) which are defined over sets of states. A set of operators $O \subseteq \mathcal{O}$ can be represented as a TR containing the set of all state pairs (s, s') such that s' is reachable from s by applying an operator $o \in O$. For a given set of states S and a TR T, the image/preimage operation computes all successors/predecessors of S with respect to the operators represented by T.

Symbolic Dijkstra search (Dijkstra 1959) in forward direction (progression) starts with the characteristic function of the initial state $\chi_\mathcal{I}$ and iteratively computes the successors until a function is found that is consistent with the goal condition. Technically, a BDD at the beginning represents the initial state, and image operations are applied iteratively until a BDD with a non-empty intersection with the BDD representing the goal conditions/states is found. Symbolic backward search (regression) can be realized by starting with the goal states, applying the preimage operation until the initial state is found. The combination of these two searches forms a bidirectional search used by most modern symbolic planners (Torralba 2015).

Computational Complexity

One interesting question is, of course, whether top-k planning is computationally harder than ordinary planning. It turns out that the short answer is: it depends. Let us first introduce two *decision problem* versions of the planning problem, which correspond to generating optimal plans.

Definition 4 (Bounded plan existence). BOUNDED PLAN EXISTENCE is the problem of deciding for a given planning task Π and a natural number ℓ, whether there exists a plan of length ℓ or less. POLYNOMIALLY BOUNDED PLAN EXISTENCE is the same problem where ℓ is bounded by some polynomial p in the size of Π, i.e., $\ell \leq p(|\Pi|)$.

Bylander (1994) has shown that BOUNDED PLAN EXISTENCE is a PSPACE-complete problem. Hardness has been shown using a generic reduction from a Turing machine. From that it follows straightforwardly that the more restricted version POLYNOMIALLY BOUNDED PLAN EXIS-

TENCE is NP-complete. Parallel to the planning problems we now introduce the top-k-existence problems.

Definition 5 (Bounded top-k-existence). BOUNDED TOP-K-EXISTENCE is the following decision problem: Given a planning task Π and two natural numbers ℓ and k, are there at least k different plans of length at most ℓ? POLYNOMIALLY BOUNDED TOP-K-EXISTENCE is the decision problem with $\ell \leq p(|\Pi|)$ for some polynomial p.

For the analysis of the new problems we need to introduce a few less well-known complexity classes: FP, FPSPACE, #P (Valiant 1979), #PSPACE (Ladner 1989), and PP (Gill 1977). FP is the set of functions computable in polynomial time. Similarly, FPSPACE is the class of functions computable in polynomial space. The space of the output tape is not taken into account, i.e., the result of a function can have a size exponential in the instance size. #P is the set of functions f such that there exists a non-determinstic polynomial-time Turing machine where the result of f is equal to the number of accepting computations. #PSPACE is a similar class where we allow for polynomial space. Finally, PP is the decision problem version of #P. It is the class of decision problems such that there exists a non-deterministic polynomial-time Turing machine where the majority of the computations are accepting.

Short Plans

Now, it is natural to consider the counting problem #POLYNOMIALLY BOUNDED PLAN EXISTENCE, i.e., the problem of determining the number of plans with a plan length less than or equal to ℓ. This problem is by definition in the complexity class #P. It is also hard for this class using Bylander's (1994) above mentioned generic reduction, because the reduction has the property that the number of successful plans and the number of accepting runs are identical.

Proposition 1. *#POLYNOMIALLY BOUNDED PLAN EXISTENCE is #P-complete.*

This does not tell us whether POLYNOMIALLY BOUNDED TOP-K-EXISTENCE is harder than POLYNOMIALLY BOUNDED PLAN EXISTENCE, but it gives an indication that the problem is indeed difficult. The natural decision problem counterpart to #P is PP, i.e., the set of problems that can be solved by a nondeterministic Turing machine in polynomial time where the acceptance condition is that a majority of computation paths accept (Gill 1977). Obviously, the majority condition is a special case of asking for a particular k.

Proposition 2. *POLYNOMIALLY BOUNDED TOP-K-EXISTENCE is PP-hard.*

Using Toda's (1991) Theorem, which shows that the entire polynomial hierarchy is included in P^{PP}, demonstrates that POLYNOMIALLY BOUNDED TOP-K-EXISTENCE appears indeed to be much harder than POLYNOMIALLY BOUNDED PLAN EXISTENCE. From the assumption that POLYNOMIALLY BOUNDED TOP-K-EXISTENCE is of the same complexity as POLYNOMIALLY BOUNDED PLAN EXISTENCE, i.e., to be a member of NP, it would follow that the polynomial hierarchy collapses at P^{NP}, which is considered to be very unlikely.

The General Case

When we consider the unrestricted problem, then one does not see a difference in complexity between BOUNDED PLAN EXISTENCE and BOUNDED TOP-K-EXISTENCE. As we will show, BOUNDED TOP-K-EXISTENCE is a PSPACE-complete problem. With the same arguments as above, we arrive at the following characterization of #BOUNDED PLAN EXISTENCE.

Proposition 3. *#BOUNDED PLAN EXISTENCE is #PSPACE-complete.*

One should note that the number of plans can be exponential in ℓ, which itself is not any longer restricted to be polynomial in $|\Pi|$. So it seems unclear whether BOUNDED TOP-K-EXISTENCE could still be decided in polynomial space. As the next theorem shows, we need indeed only polynomial space.

Theorem 4. *BOUNDED TOP-K-EXISTENCE is PSPACE-complete.*

Proof. Hardness follows for $k = 1$ from the complexity of bounded plan existence. Membership follows because of Proposition 3, the fact that $\#\mathsf{PSPACE} = \mathsf{FPSPACE}$ (Ladner 1989), and the fact that BOUNDED TOP-K-EXISTENCE is a member of $\mathsf{P}^{\mathsf{FPSPACE}}$. In order to see that the latter claim is true, let us assume the Turing machine for #BOUNDED TOP-K-EXISTENCE returns a very large number k' that is exponential in ℓ. However, we only need to read the $\lfloor \log_2 k \rfloor + 2$ least significant bits of k' in order to decide whether $k' \geq k$. This, can be done in time polynomial in the representation of k (and hence the instance). Finally, since $\mathsf{P}^{\mathsf{FPSPACE}} \subseteq \mathsf{PSPACE}$ by definition, membership of BOUNDED TOP-K-EXISTENCE in PSPACE follows. □

A Symbolic Algorithm for Top-k Planning

In contrast to explicit state space search, symbolic search expands sets of states rather than single states. One consequence of this is that often not only one (optimal) plan but several are found at once. Clearly, if multiple goal states are expanded, also multiple plans are found. However, even if only one goal state is expanded it is possible that different plans are found which lead to this particular goal state. However, in the literature so far, only one of these plans is reconstructed and all other plans are ignored.

In general, symbolic planning is split in two parts. In the beginning, all reachable states are generated until a goal state is found, followed by a plan reconstruction phase which regresses the search to reconstruct a goal path and the corresponding plan. Interestingly, the generalization of the Dijkstra algorithm for top-k planning and K* also contains two phases: a) search for reachable states and b) a plan construction phase using an additional data structure to store paths. For the sake of simplicity, we first describe SYM-K for symbolic forward search (progression) and ignore actions with zero cost. Backward search (regression), bidirectional search and actions with zero cost are addressed afterwards.

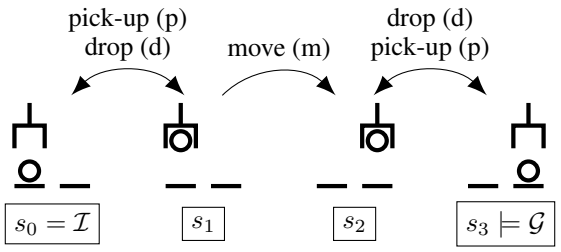

Figure 1: Visualization of a planning task based on the GRIPPER domain (International Planning Competition 1998).

Similar to modern symbolic planning systems (Torralba et al. 2014; Speck, Geißer, and Mattmüller 2018b) SYM-K performs a variant of symbolic Dijkstra search (SYM-DIJ). However, SYM-K differs from normal SYM-DIJ in three aspects:

1. once a goal state is expanded, all plans leading to that goal state are reconstructed,
2. states are not closed, and
3. SYM-K terminates if either k plans are found or the open list contains only states that have already been expanded at least once and are not part of a goal path induced by a plan already found.

We explain the functionality of SYM-K in detail below, give an example, and prove its soundness and completeness.

Symbolic Top-k Planning

In each step of SYM-K, all states S_c that appear most promising, i.e., currently have the lowest reachability cost c, are extracted from the open list and expanded. If goal states are contained in S_c, the (modified) plan reconstruction is performed and all new plans are added to the set of already found plans P. However, if during this process the desired number of plans k is found, the search terminates and returns the set of plans P. Otherwise, all newly generated states with their corresponding reachability costs are included in the open list. Note that in ordinary SYM-DIJ all expanded states are stored in a closed list and all newly generated states are filtered using the closed list. This filtering is not performed in SYM-K to prevent the loss of suboptimal plans. However, as in ordinary symbolic planning, the closed list is maintained and factorized by the costs of the states it contains. The latter is necessary to allow for plan reconstruction, which is described in detail afterwards. Finally, the termination criteria need to be adapted. Usually, SYM-DIJ is terminated when a plan is found or the open list is empty. SYM-K terminates if k plans are found or the open list contains only "known" states that do not lead to a goal state. The second condition ensures termination when no more plans can be found. Note that SYM-K has an anytime behavior, since the actual search is independent of the desired number of plans.

Plan Reconstruction

In explicit search, each state keeps track of its predecessor state, making it easy to construct a plan when a goal state is found. In symbolic search, however, the predecessor states are not directly known, but all predecessors are stored in the closed list with their reachability costs. Therefore, it is possible to perform a greedy backward search with the perfect heuristic obtained by the closed list. More precisely, the plan reconstruction iterates over all operators (descending cost) and selects an explicit predecessor contained in the closed list. The latter process is repeated until the initial state is reached. Note that a greedy search in combination with the perfect heuristic leads the search directly from a goal state to the initial state. To reconstruct all found plans, we perform an *exhaustive* greedy backward search using the provided perfect heuristic (again without closing states). Thus, we do not stop if the initial state was found once, but continue the search until we have found enough plans or we have explored the entire state space represented by the closed list. In this way, all possible plans that lead from the initial state via any expanded states included in the closed list to the detected goal states are reconstructed and added to the set of plans P. Although this exhaustive search may appear expensive, the perfect heuristics makes it goal-driven, and each time the initial state is reached, a new plan is created.

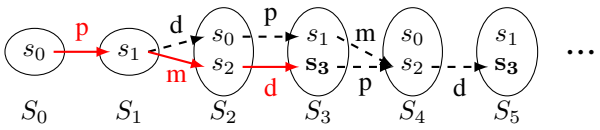

Figure 2: Visualization of SYM-K on the basis of the GRIPPER example of Figure 1.

Example 1. Consider a planning task with two rooms, a ball and a robot with a gripper (Figure 1). The robot can only move from room A to room B if it carries the ball and can never return to room A again if it has moved to room B. Furthermore, suppose the desired number of plans is three, i.e., $k = 3$. Figure 2 depicts the functioning of SYM-K. First all states S_0, which can be reached with costs 0, are extracted from the open list and expanded. Obviously S_0 contains only the initial state, which is not a goal state. The only state reachable from s_0 is s_1 with a cost of 1. Also, s_1 is not a goal state and is expanded, which leads to a set of two reachable states $S_2 = \{s_0, s_2\}$, whose cost is 2. Note that s_0 is included again, although it was expanded previously and would therefore no longer be considered in ordinary symbolic planning. Next, the set of states S_2 is expanded, resulting in the set $S_3 = \{s_1, s_3\}$ with cost of 3. Now S_3 is extracted, which contains the goal state s_3. Therefore, the plan reconstruction procedure is executed. The exhaustive greedy backward search results in exactly one plan $\langle$pick-up, move, drop$\rangle$ visualized in red in Figure 2. Since three plans are desired and we have only found one, the set of states S_3 is expanded, which leads to S_4 and then to S_5, which again contains a goal state, namely s_3. The plan reconstruction returns two plans for s_3 with a cost of 5 each, which terminates the algorithm because the desired number of plans is found. If more plans are desired, the search will continue.

Actions with Zero Costs

Up to this point, we have only considered actions with non-zero costs. In symbolic search, before applying non-zero cost actions, it is necessary to perform an additional SYM-DIJ with zero cost actions in order to obtain all states reachable with certain costs (Torralba 2015). We will illustrate this with Example 1 depicted in Figure 1, where this time the pickup and drop actions cost 0 and the move action costs 1. In this example, a SYM-DIJ initialized with s_0, which only considers zero cost actions and closes states results in $S_0 = \{s_0, s_1\}$. In other words, only s_0 and s_1 are reachable with a cost of 0. These sets of states are partitioned according to plan length in order to preserve the predecessor relationships and enable plan reconstruction. SYM-K performs exactly the same search (with closing states) within any set of states S_c. Note that SYM-DIJ without closing states, can have an infinite execution if the transition system induced by zero cost actions contains loops. In Example 1 exactly this is the case, because a SYM-DIJ without a closed list would alternately add s_0 and s_1 to the open list and the search would never get past this point. However, as explained in Example 1, closing states can discard plans. To solve the explained issue, the plan reconstruction in SYM-K considers all states reachable with the same cost c, i.e., those contained in S_c, as possible predecessors with a tiebreaking for states closer to the initial state. The following example illustrates this in more detail.

Example 2. Consider Example 1 where the pickup and drop actions cost 0 and the move action costs 1. The search starts with the initial state s_0 and the first SYM-DIJ, which considers only zero cost actions, determines that s_0 and s_1 are reachable with cost 0, i.e., $S_0 = \{s_0, s_1\}$. From S_0, only s_2 is reachable with a non-zero cost action (= move). Initialized with s_2, the subsequent zero cost SYM-DIJ determines that s_2 and s_3 are reachable with cost 1, i.e., $S_1 = \{s_2, s_3\}$. Now the goal state s_3 is expanded and the plan reconstruction starts with s_3 and finds the unique predecessor s_2. Next, there are two possible predecessors of s_2, namely s_1 and s_3. Note that s_3 is only considered as possible predecessor because it is reachable with the same cost as s_2, namely 1. Predecessor s_1, however, was reachable with cost 0 and is thus closer to the initial state and processed next. The only predecessor of s_1 is s_0. Now, s_0 is processed and because it is the initial state, we found the first plan $\langle \text{pick-up}, \text{move}, \text{drop} \rangle$. In addition, s_0 has the unique predecessor s_1, which again is considered as predecessor because it is reachable with the same cost as s_0. Finally, s_1 is again processed which forms an infinite loop until k plans are found and returned.

Theoretical Properties

In the following, we show that SYM-K is sound and complete for top-k planning.

Theorem 5. SYM-K *is sound for top-k planning.*

Proof. By construction, SYM-K expands all reachable states with increasing costs. The plan reconstruction performs an exhaustive greedy best-first search on the "induced reachability graph", which therefore finds all plans generated by SYM-DIJ. Also, all subplans with cost zero are found within a partitioned state set S_c, since all states contained in S_c are considered as possible predecessors. This and the fact that states are expanded with increasing costs proves that SYM-K is sound for top-k planning. □

Theorem 6. SYM-K *is complete for top-k planning.*

Proof. SYM-K terminates either when a) k plans are found or b) the open list contains only states that have already been expanded and are not part of a goal path induced by a plan already found. In case a), with the same argument as in the proof of Theorem 5, we know that SYM-K finds all existing plans with increasing costs. Thus, if at least k plans exist, SYM-K finds k plans at some point in time. In case b), since all states S contained in the open list have already been expanded, they can again only lead to already expanded states. Thus, S together with all reachable states from S forms a fixpoint. If a goal state $s_\star$ can be reached from a state $s \in S$, s must be part of a goal path induced by a plan that has not yet been found. If this is the case, at least one state of this path between s and $s_\star$ must be included in the open list and never expanded before. This is a contradiction that shows that SYM-K always terminates, either by a) reporting k plans or b) by reporting $k' < k$ plans, if only k' plans exist. □

Finally, we want to mention that SYM-K is strongly optimal for planning task without zero cost actions, i.e., SYM-K reports plans in increasing length among the cheapest plans. In general, this does not hold if zero cost actions exist, because once a plan is found which contains a loop consisting of actions with zero costs, that loop is extended until enough plans are found and other, perhaps shorter, plans are ignored.

Bidirectional Search

A strength of symbolic planning is that regression can be easily realized by exchanging the initial state with the goal states and using the preimage operation instead of the image operation. This enables a bidirectional search, which is used by modern symbolic planning systems (Torralba et al. 2014; Speck, Geißer, and Mattmüller 2018b). It is also possible to perform symbolic bidirectional search for top-k planning without significant modifications of the SYM-K algorithm. Both search directions maintain their own open and closed lists and only close states within a partitioned state set S_c. Plan reconstruction is again a greedy best-first search, but is performed twice, always opposing the actual search direction. More specifically, both plan reconstructions are initialized with the meeting point and one search is a regression to the initial state, while the other search is a progression to the goal states. Each combination of those two subplans results in a new final plan.

Further Extensions

In the past, symbolic planning was generalized to support conditional effects (Kissmann, Edelkamp, and Hoffmann 2014), state-dependent actions cost (Speck, Geißer, and Mattmüller 2018a) and axioms (Speck et al. 2019). Such extensions to classical planning can be combined, resulting in a widely applicable and competitive planning strategy.

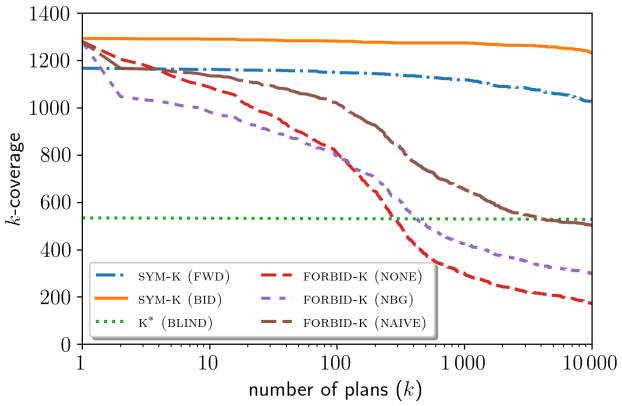

Figure 3: The k-coverage of the presented symbolic top-k planner SYM-K with forward and bidirectional search in comparision to K* and FORBID-K with different plan reordering strategies. The benchmark set consists of 73 domains (without axioms) of the optimal track from the International Planning Competition 1998-2018.

This also applies to the presented symbolic top-k planner SYM-K. In contrast, other approaches to top-k planning such as K* or FORBID-K are based on heuristics that often do not support concepts such as conditional effects or axioms.

Empirical Evaluation

The presented symbolic top-k algorithm SYM-K[1] is implemented in the SYMBA* (Torralba et al. 2014) planner, which is built on top of the FAST DOWNWARD planning system (Helmert 2006). We evaluated the performance in terms of the k-coverage, i.e., the sum of instances for which a planner reports a set of k best plans or reports only $k' < k$ plans but proves that only k' plans exist (see Definition 3). We compared our approach with planners that support PDDL (McDermott 2000) and are currently state of the art in top-k planning (Katz et al. 2018): K* and FORBID-K. K* (Aljazzar and Leue 2011; Katz et al. 2018) is a top-k planning approach that generates and processes parts of the implicit search tree as needed. The search of K* can be enhanced by a heuristic. Note that the search has no anytime behaviour, i.e., the search is influenced by the desired number of plans k. FORBID-K (Katz et al. 2018) is an iterative approach that searches for additional plans through a replanning loop that forbids already found plans and preserves all other plans. The underlying search is an orbit space search with structural symmetries (Alkhazraji et al. 2014; Domshlak, Katz, and Shleyfman 2015) and a version of the LM-cut heuristic (Helmert and Domshlak 2009) supporting conditional effects. Between the replanning steps, different reordering strategies (naive and neighbour) can be used to generate several plans from a single plan. The benchmark set we use consists of 73 planning domains from the optimal tracks of the International Planning Competitions 1998-2018. Only domains containing axioms were not considered, because the FORBID-K planner does not support these. Furthermore, we evaluated K* only with the BLIND heuristic, because planning-specific versions of K* require a consistent heuristic, but no modern heuristic that is consistent supports conditional effects. However, K* is very memory intensive, which is why the differences between K* with the blind heuristic and K* with, e.g., the iPDB heuristic (Haslum et al. 2007), on all domains without conditional effects are minor anyway ($< 2\%$ difference in coverage). For the experiment we use a time limit of 30 minutes and a 4 GB memory limit for translation, preprocessing, search and plan reports.

Figure 3 shows the k-coverage of SYM-K compared to K* and FORBID-K. The x-axis indicates the number of desired plans k, while the y-axis represents the number of instances in which the corresponding k-coverage has been reached. Informally, the higher and flatter a line, the better the k-coverage and the better is the scaling of the corresponding algorithm. Since K* has no anytime behavior, we evaluated K* for $k = 1$ and $k = 10\,000$ separately and used a linear interpolation to estimate the k-coverage between these extremes. K* however only solved 6 instances more for $k = 1$ than for $k = 10\,000$ due to the high memory consumption that forms the bottleneck of the approach. As Katz et al. (2018) have already shown, FORBID-K dominates K* for small k. But for larger k it turns out that replanning is too extensive, so K* works just as well as the best version of FORBID-K for larger k. SYM-K for $k = 1$ is only slightly better than FORBID-K, but already for $k = 2$, SYM-K with bidirectional search clearly has the highest k-coverage. Similar to ordinary symbolic planning, forward search is dominated by bidirectional search (Torralba 2015). However, SYM-K with forward search also performs better than FORBID-K and K* when five or more plans are requested. The most important observation is that SYM-K also scales to larger k where the replanning approach of FORBID-K shows a decrease in performance. If we compare the performance of SYM-K and FORBID-K for $k = 1$ and $k = 10\,000$, we can see that SYM-K with bidirectional search has less than 5% performance decrease in terms of k-coverage, while the best scaling FORBID-K configuration (naive) has more than 60% performance decrease. Furthermore, we expect that SYM-K will dominate K* and FORBID-K even for very large k until the high number of plan reports is the limiting factor, i.e., the limiting factor is to write the plans to the disk.

Table 1 shows a domainwise comparison of SYM-K with bidirectional search and FORBID-K with naive plan construction. As usual, symbolic and explicit approaches shine in different domains. However, it can be assumed that it is more difficult to find $10\,000$ plans than only one plan. But in 52 of 73 domains, the same number of instances is solved by SYM-K for $k = 1$ and $k = 10\,000$. This raises the question in how many instances there are $10\,000$ plans at all. In other words, is the main challenge of this benchmark set to prove that only $k' < k$ plans exist or to find and report k different plans? It turns out that all algorithms together can only show in 41 different instances that less than $10\,000$ plans exist. For example, the first seven instances of the PEGSOL^{08} domain have only between 4 and $2\,678$ plans. Intuitively this makes

[1] Available online: https://github.com/speckdavid/symk

Algorithm	SYM-K (bidirectional)			FORBID-K (naive)		
Domain (#Tasks) / k	1	100	10 000	1	100	10 000
AGRICOLA (20)	**6**	**6**	**6**	0	0	0
AIRPORT (50)	23	23	21	**28**	**26**	**26**
BARMAN^{11} (20)	**8**	**8**	**8**	**8**	**8**	0
BARMAN^{14} (14)	**6**	**6**	**6**	3	3	3
BLOCKS (35)	**30**	**30**	**30**	28	16	0
CALDERA-SPLIT (20)	**11**	**11**	**11**	10	10	3
CAVEDIVING (20)	**7**	**7**	**7**	**7**	0	0
CHILDSNACK (20)	1	1	1	**6**	**6**	**6**
CITYCAR (20)	15	15	15	**18**	**18**	**18**
DATA-NET (20)	**13**	**13**	**13**	12	9	1
DEPOT (22)	5	5	5	**9**	**9**	**7**
DRIVERLOG (20)	12	12	**12**	**13**	**13**	7
ELEVATORS^{08} (30)	**24**	**24**	**22**	22	19	14
ELEVATORS^{11} (20)	**19**	**19**	**18**	18	15	12
FLOORTILE^{11} (20)	**14**	**14**	**14**	8	8	8
FLOORTILE^{14} (20)	**20**	**20**	**20**	8	8	8
FREECELL (80)	**21**	**21**	**21**	15	10	0
GED (20)	**15**	**15**	**7**	**15**	5	0
GRID (5)	**2**	**2**	**2**	**2**	1	0
GRIPPER (20)	**20**	**20**	**20**	**20**	**20**	**20**
HIKING (20)	**15**	**15**	**15**	13	13	11
LOGISTICS^{98} (35)	5	5	**5**	**6**	**6**	**5**
LOGISTICS^{00} (28)	**20**	**20**	**20**	**20**	**20**	17
MAINTENANCE (5)	**5**	**5**	**5**	**5**	**5**	2
MICONIC (150)	116	**116**	**114**	**142**	95	36
MICONIC-SA (150)	**150**	**150**	**150**	143	91	4
MOVIE (30)	**30**	**30**	**30**	**30**	**30**	**30**
MPRIME (35)	18	**16**	**14**	**23**	13	0
MYSTERY (30)	**25**	**23**	**21**	21	15	4
NOMYSTERY (20)	14	14	**14**	**15**	**15**	1
NURIKABE (20)	**11**	**11**	**11**	10	6	0
OPENSTACKS (30)	**20**	**20**	**20**	7	6	5
OPENSTACKS^{08} (30)	**30**	**30**	**29**	24	24	22
OPENSTACKS^{11} (20)	**20**	**20**	**20**	19	19	19
OPENSTACKS^{14} (20)	**13**	**13**	**8**	5	5	5
ORGANIC-SPLIT (20)	13	12	**12**	**19**	**15**	10
PARCPRINTER^{08} (30)	18	18	**18**	**19**	**19**	16
PARCPRINTER^{11} (20)	13	13	**13**	**14**	**14**	12
PARKING^{11} (20)	0	**0**	**0**	**2**	**0**	**0**
PARKING^{14} (20)	0	**0**	**0**	**3**	**0**	**0**
PATHWAYS (30)	**5**	**5**	**5**	**5**	0	0
PATHWAYS-NN (30)	**5**	**5**	**5**	**5**	0	0
PEGSOL^{08} (30)	**29**	**29**	**29**	**29**	21	5
PEGSOL^{11} (20)	**19**	**19**	**19**	**19**	10	0
PETRI-NET (20)	**17**	**17**	**10**	9	9	9
PIPESWORLD-NT (50)	15	15	**13**	**21**	**19**	6
PIPESWORLD-T (50)	13	13	**12**	**16**	**15**	7
PSR-SMALL (50)	**50**	**50**	**50**	**50**	49	11
ROVERS (40)	**14**	**14**	**14**	9	9	5
SATELLITE (36)	11	11	**11**	**14**	**14**	9
SCANALYZER^{08} (30)	12	12	**10**	**17**	**16**	8
SCANALYZER^{11} (20)	9	9	7	**14**	**13**	**8**
SCHEDULE (150)	**46**	**46**	**46**	44	44	25
SETTLERS (20)	6	6	**6**	**9**	**9**	**6**
SNAKE (20)	3	**0**	**0**	**6**	**0**	**0**
SOKOBAN^{08} (30)	28	**28**	**25**	**30**	23	10
SOKOBAN^{11} (20)	**20**	**20**	**19**	**20**	14	5
SPIDER (20)	6	**3**	**0**	**11**	2	**0**
STORAGE (30)	14	14	**14**	**17**	**17**	4
TERMES (20)	**16**	**16**	**16**	6	2	0
TETRIS (17)	**9**	**9**	**9**	8	8	6
TIDYBOT^{11} (20)	12	12	**10**	**14**	**14**	0
TIDYBOT^{14} (20)	4	4	**2**	**8**	**8**	1
TPP (30)	**8**	**8**	**8**	**8**	**8**	5
TRANSPORTER^{08} (30)	**11**	**11**	**11**	**11**	7	1
TRANSPORTER^{11} (20)	**7**	**7**	**7**	**7**	3	1
TRANSPORTER^{14} (20)	**6**	**6**	**6**	**6**	2	1
TRUCKS (30)	**12**	**12**	**12**	**12**	**12**	5
VISITALL^{11} (20)	**12**	**12**	**12**	**12**	9	0
VISITALL^{14} (20)	**6**	**6**	**6**	**6**	1	0
WOODWORK^{08} (30)	**30**	**30**	**30**	20	20	15
WOODWORK^{11} (20)	**20**	**20**	**20**	14	14	14
ZENOTRAVEL (20)	10	10	**10**	**13**	**13**	4
$\sum$ (2242)	**1293**	**1282**	**1232**	1280	1020	503

Table 1: A domainwise comparison of SYM-K with bidirectional search and FORBID-K with naive plan reordering for different numbers of desired plans k.

sense, because with each action the number of pegs on the board is monotonically reduced and so there are only a finite number of plans. In contrast, there are domains in which there exist an infinite number of plans. Interestingly, in some domains there are even infinitely many optimal plans. For example, in the ELEVATOR domain, it is possible to board and unboard passengers without any costs. This leads to infinitely many (similar) plans by simply adding these two actions to a valid plan infinitely often. These plans can be very lengthy and expensive to report. The shortest plan found by SYM-K in the first instance of the ELEVATORS^{08} domain has 14 actions, while the longest plan consists of $20\,012$ actions (both with cumulative cost of 42). Nevertheless, the domainwise comparison shows that for $k = 10\,000$, SYM-K has more than twice the k-coverage of FORBID-K and K*. For $k = 10\,000$, SYM-K dominates FORBID-K in 60 domains, while FORBID-K dominates SYM-K only in 5 domains.

Conclusion

On the theoretical side, we have proven that the BOUNDED TOP-K-EXISTENCE problem, i.e., the problem of answering whether there are at least k different plans of length at most ℓ for a given planning task, is PSPACE-complete. This result is surprising, because the ordinary BOUNDED PLAN EXISTENCE problem, i.e., answering the question if only one such plan exists, is also PSPACE-complete (Bylander 1994). However, if the plan length ℓ is polynomially bounded by the input, the problem of deciding whether k different plans exist is PP-hard and thus very likely much more difficult than deciding whether a single polynomially bounded plan exists which is known to be NP-complete.

On the practical side, we presented a novel approach to top-k planning, SYM-K, based on symbolic search, and proved that SYM-K is a sound and complete top-k planning algorithm. We have empirically shown that SYM-K performs better than other state-of-the-art approaches even for a small number of desired plans k. In addition, SYM-K scales to larger k better than all other approaches, making SYM-K suitable regardless of the number of desired plans.

For future work, we want to evaluate how well top-k planning performs in scenarios where complex concepts for plans are required, such as preferences (Ceriani and Gerevini 2015), state-trajectory constraints (Wright, Mattmüller, and Nebel 2018) or even moral permissibility (Lindner, Mattmüller, and Nebel 2019). In addition, we plan to generalize SYM-K to search for diverse plans (Katz and Sohrabi 2019), since in some cases it may be desirable to find plans that differ according to a certain specification in order to avoid plans that are, e.g., different orders of the same actions. In general, it is possible to create plans until any requirement is met. SYM-K can, however, also be used to meet requirements more efficiently by restricting the plan reconstruction to ensure such constraints.

Acknowledgments. David Speck was supported by the German Research Foundation (DFG) as part of the project EPSDAC (MA 7790/1-1).

References

Aljazzar, H., and Leue, S. 2011. K*: A heuristic search algorithm for finding the k shortest paths. *AIJ* 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 *IPC-8 planner abstracts*, 88–92.

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

Bock, F.; Kanter, H.; and Haynes, J. 1957. *An algorithm (the r-th best path algorithm) for finding and ranking paths through a network*. Armour Research Foundation.

Bryant, R. E. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* 35(8):677–691.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69(1–2):165–204.

Ceriani, L., and Gerevini, A. E. 2015. Planning with always preferences by compilation into STRIPS with action costs. In *Proc. SoCS 2015*, 161–165.

Corraya, S.; Geier, F.; Speck, D.; and Mattmüller, R. 2019. An empirical study of the usefulness of state-dependent action costs in planning. In *Proc. KI 2019*, 123–130.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1:269–271.

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.

Edelkamp, S.; Kissmann, P.; and Torralba, Á. 2015. BDDs strike back (in AI planning). In *Proc. AAAI 2015*, 4320–4321.

Eifler, R.; Cashmore, M.; Hoffmann, J.; Magazzeni, D.; and Steinmetz, M. 2019. Explaining the space of plans through plan-property dependencies. In *ICAPS 2019 Workshop on Explainable Planning*.

Gill, J. 1977. Computational complexity of probabilistic turing machines. *SIAM J. Comput.* 6(4):675–695.

Gnad, D.; Torralba, Á.; Domínguez, M.; Areces, C.; and Bustos, F. 2019. Learning how to ground a plan – partial grounding in classical planning. In *Proc. AAAI 2019*, 7602–7609.

Günther, M.; Schuster, J.; and Siegle, M. 2010. Symbolic calculation of k-shortest paths and related measures with the stochastic process algebra tool Caspa. In *Proceedings of the First Workshop on DYnamic Aspects in DEpendability Models for Fault-Tolerant Systems*, 13–18. ACM.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

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., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. ICAPS 2009*, 162–169.

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

Katz, M., and Sohrabi, S. 2019. Reshaping diverse planning: Let there be light! In *ICAPS 2019 Workshop on Heuristics and Search for Domain-independent Planning*.

Katz, M.; Sohrabi, S.; Udrea, O.; and Winterer, D. 2018. A novel iterative approach to top-k planning. In *Proc. ICAPS 2018*.

Kissmann, P.; Edelkamp, S.; and Hoffmann, J. 2014. Gamer and dynamic-gamer – symbolic search at ipc 2014. In *IPC-8 planner abstracts*, 77–84.

Ladner, R. E. 1989. Polynomial space counting problems. *SIAM J. Comput.* 18(6):1087–1097.

Lindner, F.; Mattmüller, R.; and Nebel, B. 2019. Moral permissibility of action plans. In *Proc. AAAI 2019*.

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

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.

Sohrabi, S.; Riabov, A. V.; and Udrea, O. 2016. Plan recognition as planning revisited. In *Proc. IJCAI 2016*, 3258–3264.

Speck, D.; Geißer, F.; Mattmüller, R.; and Torralba, Á. 2019. Symbolic planning with axioms. In *Proc. ICAPS 2019*, 464–572.

Speck, D.; Geißer, F.; and Mattmüller, R. 2018a. Symbolic planning with edge-valued multi-valued decision diagrams. In *Proc. ICAPS 2018*, 250–258.

Speck, D.; Geißer, F.; and Mattmüller, R. 2018b. SYMPLE: Symbolic Planning based on EVMDDs. In *IPC-9 planner abstracts*, 91–94.

Toda, S. 1991. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.* 20(5):865–877.

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. 2015. *Symbolic Search and Abstraction Heuristics for Cost-Optimal Planning*. Ph.D. Dissertation, Universidad Carlos III de Madrid.

Toyer, S.; Trevizan, F.; Thibaux, S.; and Xie, L. 2018. Action schema networks: Generalised policies with deep learning. In *Proc. AAAI 2018*, 6294–6301.

Valiant, L. G. 1979. The complexity of computing the permanent. *Theoretical Computer Science* 8:189–201.

Wright, B.; Mattmüller; and Nebel, B. 2018. Compiling away soft trajectory constraints in planning. In *Proc. KR 2018*, 474–482.