Symbolic Search for Oversubscription Planning

David Speck\(^1\) and Michael Katz\(^2\)

\(^1\) University of Freiburg, Freiburg im Breisgau, Germany
\(^2\) IBM T. J. Watson Research Center, Yorktown Heights, NY, USA
speckd@informatik.uni-freiburg.de, michael.katz1@ibm.com

Abstract

The objective of optimal oversubscription planning is to find a plan that satisfies the entire formula. In practice, the situation occurs whenever a large number of possible, often competing goals of varying value exist, or the resources are not sufficient to achieve all goals. In this paper, we investigate the use of symbolic search for optimal oversubscription planning. Specifically, we show how to apply symbolic forward search to oversubscription planning tasks and prove that our approach is sound, complete and optimal. An empirical analysis shows that our symbolic approach favorably competes with explicit state-space heuristic search, the current state of the art for oversubscription planning.

Introduction

The objective of classical planning is to find a plan, i.e., a sequence of actions with low cumulative cost that leads to a goal state. Goal states are usually specified by a goal formula, and most planning algorithms are designed to find a plan that satisfies the entire formula. In practice, however, there can be a large number of possible, often competing goals of varying value, and a system (e.g., Mars rover) might not be able to achieve all these goals with the available resources (e.g., battery power). In such scenarios, it is natural to consider a utility of a state instead and search for states that maximize the overall utility value. If the action costs and state utility values, i.e., solution cost and solution utility, are comparable, the problem of finding a plan is called net-benefit planning (van den Briel et al. 2004). If solution cost and utility are not comparable, the problem is called oversubscription planning (Smith 2004). In other words, oversubscription planning defines a utility function that maps states to the utility values associated with achieving these states, and the objective is to find a plan whose cost does not exceed a specified cost bound, while reaching a state that achieves a high utility. Both net-benefit and oversubscription planning are also known under the common name partial satisfaction planning.

Over the past decades, both classical and partial satisfaction planning have seen a variety of competitive approaches, with the most successful being the explicit state-space heuristic search (Bonet and Geffner 1999) and the symbolic search (Cimatti et al. 1997). For net-benefit in particular, symbolic branch-and-bound search was introduced (Edelkamp and Kissmann 2009; Kissmann 2012), solving a so-called planning with soft-goals problem. More precisely, instead of a general state utility function, a subset of variables, so-called soft goals, is specified, each associated with a certain utility value. An introduction of a compilation of soft-goals into classical planning (Keyder and Geffner 2009), has allowed to apply classical planners to solving net-benefit planning tasks. The underlying idea behind the compilation is that each net-benefit planning task can be reduced to finding a shortest path from a single source node to a single goal node in a graph under a single objective function.

For oversubscription planning, however, unlike for net-benefit planning, there is no simple reduction to classical planning with a single objective function. In fact, the complexity of several fragment of these two problems is different (Katz and Mirkis 2016). The first practically efficient approaches to optimally solving oversubscription planning applied an explicit branch-and-bound search with heuristics adapting the ideas of classical planning to oversubscription planning (Mirkis and Domshlak 2013, 2014; Domshlak and Mirkis 2015; Muller and Karpas 2018). Later, Katz et al. (2019a) introduced reformulations of oversubscription planning as classical planning with two cost functions, but without an utility function. The reformulations allowed to directly adapt classical planning heuristics to oversubscription planning by simply ignoring the secondary cost function, improving the efficiency of branch-and-bound search. Finally, Katz and Keyder (2019) show that these reformulations are useful for search as well, allowing switching to the more efficient A* search (Hart, Nilsson, and Raphael 1968), performed on the reformulated task. The search can then use the aforementioned adapted heuristics from classical planning (Katz and Keyder 2019). However, if such heuristics are used as-is, and not adapted to account for the secondary cost function, their informativeness is not sufficient to compensate for the time spent on the computation. As a result, the informed search is often outperformed by a simple blind search. If, however, the heuristics are adapted to account for the secondary cost function, the performance of the informed search improves significantly (Katz and Keyder 2019).
Definition 1 (OSP Task). An oversubscription planning task (OSP task) is a 6-tuple \( \Pi = (V, O, s_0, c, u, b) \). \( V \) is a finite set of state variables, each associated with a finite domain \( D_v = \{0, \ldots, |D_v| - 1\} \). A fact is a pair \((v, d)\), where \( v \in V \) and \( d \in D_v \). For binary variables we also write \( v \) for \((v, 1)\) and \( \neg v \) for \((v, 0)\). A partial variable assignment \( s \) over \( V \) is a consistent set of facts. If \( s \) assigns a value to each variable \( v \in V \), \( s \) is called a state. With \( S \) we refer to the set of all possible states defined over \( V \). \( O \) is a finite set of operators, where each operator is a pair \( o = (pre_o, eff_o) \) of partial variable assignments, called preconditions and effects. The state \( s_0 \in S \) is called the initial state. Each operator has non-negative cost defined by the function \( c : O \rightarrow \mathbb{N}_0 \). Finally, \( u : S \rightarrow \mathbb{N}_0 \) is an efficiently computable state utility function and \( b \in \mathbb{N}_0 \) specifies a cost bound.

An operator \( o \in O \) is applicable in a state \( s \) iff \( pre_o \) is satisfied in \( s \), i.e., \( pre_o \subseteq s \). Applying operator \( o \) to state \( s \) yields the state \( s' \) where \( s'(v) = eff_o(v) \) for all variables \( v \in V \) for which \( eff_o \) is defined and \( s'(v) = s(v) \) otherwise. A plan in oversubscription planning is defined as follows.

Definition 2 (Plan). A plan \( \pi = (o_0, \ldots, o_{n-1}) \) for an oversubscription planning task \( \Pi \) is a sequence of operators applicable in \( s_0 \) that yield the states \( s_1, \ldots, s_n \), such that the cumulative operator cost \( c(\pi) = \sum_{i=0}^{n-1} c(o_i) \) is smaller or equal to the cost bound \( b \), i.e., \( c(\pi) \leq b \). The utility \( u(\pi) \) of plan \( \pi \) is defined by the utility of the end state \( s_n \) induced by \( \pi \), i.e., \( u(\pi) = u(s_n) \). A plan \( \pi \) is considered to be utility-optimal or simply optimal if there is no other plan \( \pi' \) such that \( u(\pi') > u(\pi) \). A plan \( \pi \) is cheapest utility-optimal if it is cheapest among utility-optimal plans: there is no utility-optimal plan \( \pi' \) such that \( c(\pi') < c(\pi) \).

Since the empty plan is already a valid plan for any OSP task, the main objective of oversubscription planning is to determine a plan with high utility. The search for an optimal plan, i.e., a plan with the highest utility, is called optimal oversubscription planning and is the main focus of this work.

Symbolic Search

Symbolic search is a technique for exploring state spaces that uses efficient data structures to represent and manipulate sets of states (McMillan 1993). Symbolic search algorithms resemble their explicit counterparts, but expand and generate whole sets of states in contrast to individual states. In symbolic planning, a set of states \( S \subseteq S \) is represented by its characteristic function \( \chi_S \), which is a Boolean function \( \chi_S : S \rightarrow \{0, 1\} \). States contained in \( S \) are mapped to 1 and all other to 0, i.e., \( \chi_S(s) = 1 \) if \( s \in S \) and \( \chi_S(s) = 0 \) otherwise. An operator \( o \in O \) can be represented as transition relation (TR) that is defined over sets of state pairs, namely predecessors and successors. A TR \( T_o \) representing an operator \( o \in O \) is a function \( \chi_T : S \times S' \rightarrow \{0, 1\} \) which maps the pairs of states \((s, s')\) to 1 if successor \( s' \) is reachable from predecessor \( s \) by applying operator \( o \in O \). Given a set of states \( S \) and a TR \( T_o \), the image \((S, T_o)\) operation computes all successors of \( S \) with respect to the operator \( o \). Note that a single TR can in general represent multiple operators with the same cost (Torralba 2015). From now on, if it is clear from the context, we sometimes simplify notation and use the same symbol \( S \) for a set of states \( S \) and the corresponding characteristic function \( \chi_S \).

Symbolic representation of a planning task makes it possible to carry out symbolic forward search. The search starts with the characteristic function of the initial state \( \chi(s_0) \) and iteratively computes the successors until a function is found which meets a certain stopping condition. In classical planning, e.g., the stopping condition is an non-empty intersection with the goal. Finally, a plan reconstruction (Torralba 2015) is performed to obtain the final plan. This is necessary because, in contrast to explicit search, the predecessor states of an expanded state are not directly known in symbolic search. The plan is constructed by reconstructing the path from the initial state to the end state using a closed list which provides the perfect heuristic for all expanded states.

Symbolic Data Structures

In symbolic search, characteristic functions are usually represented by compact and efficient symbolic data structures such as (reduced and ordered) Binary Decision Diagrams (BDDs) (Bryant 1986).

Definition 3 (Binary Decision Diagram). A BDD is a directed acyclic graph with a single root node and two terminal nodes: the 0-sink and the 1-sink. Each inner node corresponds to a binary variable \( v \in V \) and has two successors,

\[^1\]Each finite-domain variable \( v \in V \) can be represented by \( \lfloor \log_2 |D_v| \rfloor \) binary variables.
where the low edge represents that variable $v$ is false, while the high edge represents that variable $v$ is true. By traversing the BDD according to a given assignment, the represented function can be evaluated.

Figure 1a visualizes the BDD which represents the characteristic function $\chi_S = x \land \neg y$, i.e., the set of states $S = \{ s \in S \mid s(x) = 1 \land s(y) = 0 \}$.

Algebraic Decision Diagrams (ADDs) (Bahar et al. 1997) and Edge-Valued Multi-Valued Decision Diagrams (EVMDDs) (Lai, Pedram, and Vrudhula 1996; Ciardo and Siminiceanu 2002) have been successfully used to represent numerical functions (Hansen, Zhou, and Feng 2002; Torralba 2015). For example, the numeric function $u = 2x + y$ can be represented as ADD if not stated differently.

**Operations.** The $\max(u)$ operation determines the maximum value of a function $u$, i.e., $\max(u) = \max_{s \in S} u(s)$. Note that the terminal nodes of an ADD $u$ describe all possible values that $u$ can have. This makes it possible to efficiently determine the maximum value of $u$ once the ADD is created. The function shown in Figure 1b has exactly three different values, where $3$ is the maximum value. In addition, an ADD can be disassembled into multiple BDDs, one for each terminal node, in polynomial time and memory with respect to the ADD size (number of nodes) by substituting terminal nodes (Torralba 2015). For example, the ADD shown in Figure 1b can be disassembled into three different BDDs, one for each terminal node. Figure 1a depicts the ADD representing all states for which the evaluation of function $u = 2x + xy$ is 2. The $\arg \max(u)$ operation takes advantage of this property and returns the set of states with the maximum value of a function $u$ as BDD, i.e. $\arg \max(u) = \arg \max_{s \in S} u(s)$. Finally, the intersection $u \land S$ between an ADD $u$ and BDD $S$ results in a new ADD $u_S$, where $u_S(s) = u(s)$ if $s \in S$ and $u_S(s) = -\infty$ otherwise. Intuitively, this operation preserves only the utility values of states in $S$ which makes it for example possible to determine the maximum utility value of a given set of states $S$ by applying $\max(S \land u)$.

**Symbolic Search for OSP**

In this section, we describe and explain how symbolic forward search can be applied to oversubscription planning and prove that our presented algorithm SYM-OSP is sound, complete and optimal. Finally, we discuss the possible application of symbolic backward and bidirectional search to oversubscription planning and the associated challenges and issues. For simplicity, we define SYM-OSP only explicitly for oversubscription planning tasks with non-unit operator cost, i.e., where each operator has an application cost of 1. Symbolic uniform search, also known as Dijkstra’s algorithm (Dijkstra 1959) can be used for non-unit cost oversubscription planning tasks, which is possible by bucketing (disassembling) transition relations, open lists and closed lists into subsets with identical cost values represented as BDDs (Edelkamp and Kissmann 2009; Torralba 2015). Note that the presented algorithm and all presented concepts and theoretical results generalize to oversubscription planning tasks with non-unit open operator cost.

**Algorithm SYM-OSP**

The underlying idea of SYM-OSP (Algorithm 1) is to perform an exhaustive forward search and to explore all states reachable until the cost bound is reached or a state with maximal utility is found. Once one of the termination criteria is satisfied, the plan is reconstructed. Recall, that the plan reconstruction is necessary because, in contrast to explicit search, the predecessor states of an expanded state are not directly known in the symbolic search. SYM-OSP is analogous to symbolic forward search for classical planning, with the difference that in classical planning we terminate as soon as the open list is empty or a goal state is found. Note that all sets of states are represented as BDDs and all numeric functions as ADDs. In detail, SYM-OSP (Algorithm 1) works as follows. First, the open list is initialized as the singleton set containing the initial state, and the closed list is initialized as the empty set of states (line 1). The BDD $\pi_S$ maintains the previously expanded states with the highest utility value found and is thus also initialized as the singleton set containing the initial state (line 2). Similarly, with $\pi_u$ we store the highest utility of all states that have been expanded so far and with $\pi_g$ we store the cheapest cost with which a state with utility $\pi_u$ can be reached. Recall that the empty plan is already a valid, but often not optimal plan for each OSP task. The algorithm continues as long as 1) the open list is not empty, 2) the cost bound has not been exceeded, and 3) no state with the highest possible utility has been found (line 4). In each iteration, the current set of states open is expanded and the already expanded states closed are removed from the resulting set of successors. In line 7 the maximum utility value of all new states open is

---

2Our implementation supports operators with non-unit costs.
Algorithm 1: SYM-OSP for unit cost OSP tasks

Data: OSP Task $\Pi = (V, O, s_0, O \mapsto 1, u, b)$
Result: Cheapest utility-optimal plan $\pi$

1. $open, closed \leftarrow \{s_0\}, \emptyset$
2. $\pi_S, \pi_u, \pi_g \leftarrow \{s_0\}, u(s_0), 0$
3. $g \leftarrow 1$
4. while $open \neq \emptyset$ and $g \leq b$ and $\pi_u < \max(u)$ do
   5. $closed \leftarrow closed \lor open$
   6. $open \leftarrow (\bigcup_{o \in O} \text{image}(open, T_o)) \land \neg closed$
   7. if $\pi_u < \max(open \land u)$ then
      8. $\pi_S \leftarrow \text{arg max}(\max(open \land u))$
      9. $\pi_u \leftarrow \max(open \land u)$
     10. $\pi_g \leftarrow g$
     11. $g \leftarrow g + 1$
12. return $\text{ReconstPlan}(\Pi, open, closed, \pi_S, \pi_u, \pi_g)$

The dominant search strategy of modern symbolic planners (Torralba 2014; Kissmann, Edelkamp, and Hoffmann 2009; Torralba et al. 2017), referred to from now on as SYM-OSP, returns only valid plans, proving that SYM-OSP is sound for oversubscription planning.

Algorithm SYM-OSP terminates if one of the following three cases occurs (line 4): 1) the entire reachable state space has been explored ($open = \emptyset$), 2) there are no more states that can be reached with costs less than the cost bound ($g > b$) or 3) a state with maximum utility value has been found ($\pi_u = \max(u)$). SYM-OSP expands all reachable states with increasing reachability cost $g$ without considering already expanded states. Thus, at some point one of the condition holds and a valid plan is returned, proving that SYM-OSP is complete for oversubscription planning.

Proposition 1. Algorithm SYM-OSP is sound and complete for oversubscription planning.

Proposition 2. SYM-OSP is optimal for oversubscription planning.

Proof. Any sequence of operators $\pi$ that is applicable in the initial state and whose cumulative cost is lower than the reach bound $b$ is a valid plan for an oversubscription planning task. Since SYM-OSP only expands states which are reachable within the cost bound (line 4), and all the resulting sequences of actions are applicable, SYM-OSP therefore returns only valid plans, proving that SYM-OSP is sound for oversubscription planning.

It is possible to show that SYM-OSP is optimal, meaning that SYM-OSP returns utility-optimal plans (Definition 2) for a given oversubscription planning task $\Pi$.

Algorithm SYM-OSP terminates when the set of states $PiS$ is initialized with the initial state $s_0$, which can be reached with cost $\pi_g = 0$ and has the utility value $u(s_0) = 0$ (line 2). The first expansion (line 6) leads to two new states $open = \{s_1, s_2\} = (\neg x \land \neg y, x \land \neg y)$, both of which can be reached with cost $g = 1$. Since the intersection $open \land u$ maps the utility values of states $s \notin open$ to $-\infty$, the result of $\max(open \land u)$ is 2 (line 7 and 8). The operation arg $\max(open \land u)$ returns the set of states with the maximum utility $\pi_S = \{s_2\} = x \land \neg y$ as BDD, which is shown in Figure 1a (line 8). Finally, the plan $\pi_g$, which leads to one of the state in $\pi_S$ (here $s_2$), is reconstructed (line 12), since the cost bound of $b = 1$ is exceeded (line 4).

Theoretical Properties

In the following, we show that SYM-OSP is sound, complete and optimal for oversubscription planning.

Proposition 1. Algorithm SYM-OSP is sound and complete for oversubscription planning.

It is possible to show that SYM-OSP is optimal, meaning that SYM-OSP returns utility-optimal plans (Definition 2) for a given oversubscription planning task $\Pi$.

Proposition 2. Algorithm SYM-OSP is optimal for oversubscription planning.

Proof. Any sequence of operators $\pi$ that is applicable in the initial state and whose cumulative cost is lower than the reach bound $b$ is a valid plan for an oversubscription planning task. Since SYM-OSP only expands states which are reachable within the cost bound (line 4), and all the resulting sequences of actions are applicable, SYM-OSP therefore returns only valid plans, proving that SYM-OSP is sound for oversubscription planning.

Algorithm SYM-OSP terminates if one of the following three cases occurs (line 4): 1) the entire reachable state space has been explored ($open = \emptyset$), 2) there are no more states that can be reached with costs less than the cost bound ($g > b$) or 3) a state with maximum utility value has been found ($\pi_u = \max(u)$). SYM-OSP expands all reachable states with increasing reachability cost $g$ without considering already expanded states. Thus, at some point one of the condition holds and a valid plan is returned, proving that SYM-OSP is complete for oversubscription planning.

It is possible to show that SYM-OSP is optimal, meaning that SYM-OSP returns utility-optimal plans (Definition 2) for a given oversubscription planning task $\Pi$.

Proposition 2. Algorithm SYM-OSP is optimal for oversubscription planning.

Proof. Any sequence of operators $\pi$ that is applicable in the initial state and whose cumulative cost is lower than the reach bound $b$ is a valid plan for an oversubscription planning task. Since SYM-OSP only expands states which are reachable within the cost bound (line 4), and all the resulting sequences of actions are applicable, SYM-OSP therefore returns only valid plans, proving that SYM-OSP is sound for oversubscription planning.

Algorithm SYM-OSP terminates if one of the following three cases occurs (line 4): 1) the entire reachable state space has been explored ($open = \emptyset$), 2) there are no more states that can be reached with costs less than the cost bound ($g > b$) or 3) a state with maximum utility value has been found ($\pi_u = \max(u)$). SYM-OSP expands all reachable states with increasing reachability cost $g$ without considering already expanded states. Thus, at some point one of the condition holds and a valid plan is returned, proving that SYM-OSP is complete for oversubscription planning.

It is possible to show that SYM-OSP is optimal, meaning that SYM-OSP returns utility-optimal plans (Definition 2) for a given oversubscription planning task $\Pi$. 

Proposition 2. Algorithm SYM-OSP is optimal for oversubscription planning.

Proof. Any sequence of operators $\pi$ that is applicable in the initial state and whose cumulative cost is lower than the reach bound $b$ is a valid plan for an oversubscription planning task. Since SYM-OSP only expands states which are reachable within the cost bound (line 4), and all the resulting sequences of actions are applicable, SYM-OSP therefore returns only valid plans, proving that SYM-OSP is sound for oversubscription planning.

Algorithm SYM-OSP terminates if one of the following three cases occurs (line 4): 1) the entire reachable state space has been explored ($open = \emptyset$), 2) there are no more states that can be reached with costs less than the cost bound ($g > b$) or 3) a state with maximum utility value has been found ($\pi_u = \max(u)$). SYM-OSP expands all reachable states with increasing reachability cost $g$ without considering already expanded states. Thus, at some point one of the condition holds and a valid plan is returned, proving that SYM-OSP is complete for oversubscription planning.

It is possible to show that SYM-OSP is optimal, meaning that SYM-OSP returns utility-optimal plans (Definition 2) for a given oversubscription planning task $\Pi$.

Proposition 2. Algorithm SYM-OSP is optimal for oversubscription planning.

Proof. Any sequence of operators $\pi$ that is applicable in the initial state and whose cumulative cost is lower than the reach bound $b$ is a valid plan for an oversubscription planning task. Since SYM-OSP only expands states which are reachable within the cost bound (line 4), and all the resulting sequences of actions are applicable, SYM-OSP therefore returns only valid plans, proving that SYM-OSP is sound for oversubscription planning.

Algorithm SYM-OSP terminates if one of the following three cases occurs (line 4): 1) the entire reachable state space has been explored ($open = \emptyset$), 2) there are no more states that can be reached with costs less than the cost bound ($g > b$) or 3) a state with maximum utility value has been found ($\pi_u = \max(u)$). SYM-OSP expands all reachable states with increasing reachability cost $g$ without considering already expanded states. Thus, at some point one of the condition holds and a valid plan is returned, proving that SYM-OSP is complete for oversubscription planning.

It is possible to show that SYM-OSP is optimal, meaning that SYM-OSP returns utility-optimal plans (Definition 2) for a given oversubscription planning task $\Pi$. 

Proposition 2. Algorithm SYM-OSP is optimal for oversubscription planning.
The presented symbolic approaches to oversubscription planning were implemented\(^3\) in the SYMBA (Torralba et al. 2014) planner, built on top of the FAST DOWNWARD planning system (Helmert 2006). The two approaches, SYM-OSP uBDD and SYM-OSP uADD, perform symbolic forward search (see Algorithm 1), with the difference being in the way the utility of a set of states is evaluated. SYM-OSP uBDD disassembles the utility function into multiple BDDs and performs multiple intersections to determine the maximum utility of a set of states, while SYM-OSP uADD uses a single ADD. In practice, ADDs are often better suited to determine the value of a single state in an explicit search, while a collection of BDDs is considered more suitable for symbolic search (Torralba 2015). In addition, we implemented $A^\star_{\text{uADD}}$ (Hart, Nilsson, and Raphael 1968) as an explicit search, representing the utility function as an ADD to determine the utility values of a single state. Algorithm $A^\star_{\text{uADD}}$ with the blind heuristic $h_{\text{blind}}$ is the explicit counterpart of the algorithms SYM-OSP uBDD and SYM-OSP uADD.

\[^3\]Available online: https://github.com/speckdavid/symbolic-osp

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>SYM-OSP</th>
<th>$A^\star_{\text{uADD}}$</th>
<th>$A^\star_{\text{mc}}$</th>
<th>bNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benchmark Set (# Inst.)</td>
<td>uBDD</td>
<td>uADD</td>
<td>$h_{\text{blind}}$</td>
<td>$h_{\text{blind}}$</td>
</tr>
<tr>
<td>25% Bound (1667)</td>
<td>1271</td>
<td>1274</td>
<td>1165</td>
<td>1197</td>
</tr>
<tr>
<td>50% Bound (1667)</td>
<td>990</td>
<td>993</td>
<td>860</td>
<td>901</td>
</tr>
<tr>
<td>75% Bound (1667)</td>
<td>866</td>
<td>862</td>
<td>718</td>
<td>758</td>
</tr>
<tr>
<td>100% Bound (1667)</td>
<td>802</td>
<td>793</td>
<td>629</td>
<td>668</td>
</tr>
<tr>
<td>OVERALL (6668)</td>
<td>3929</td>
<td>3922</td>
<td>3372</td>
<td>3524</td>
</tr>
</tbody>
</table>

Table 1: Coverage of the presented symbolic algorithms and heuristic search approaches on the oversubscription planning benchmark set which is based on the optimal track from the International Planning Competition 1998-2014 (Katz et al. 2019b). The best coverage for each benchmark set, i.e. the maximum entry of each row, is highlighted in bold.

2014; Speck, Mattmüller, and Nebel 2020) is bidirectional blind search (Torralba, Linares López, and Borrajo 2016; Speck, Geißer, and Mattmüller 2020). However, it is not straightforward to apply regression efficiently to oversubscription planning. In classical planning, the underlying idea of symbolic backward search, i.e., regression, is to start with the set of goal states and regress until the initial state is found. This is possible because the set of goal states is represented as a compact but often under-specified goal formula. However, since there is no goal formula in oversubscription planning, the starting point of the regression is not obvious. Since the objective is to find an end state with maximum utility value, it might be possible to partition all states according to their utility values and represent them as separate BDDs by disassembling the ADD representing the utility function. However, this would require to regress the entire state space by regressing several BDDs with multiple backward searches. All in all, it is not clear how to apply symbolic regression efficiently to oversubscription planning. We leave this task for future work.

**Empirical Evaluation**

The presented symbolic approaches to oversubscription planning were implemented\(^3\) in the SYMBA (Torralba et al. 2014) planner, built on top of the FAST DOWNWARD planning system (Helmert 2006). The two approaches, SYM-OSP uBDD and SYM-OSP uADD, perform symbolic forward search (see Algorithm 1), with the difference being in the way the utility of a set of states is evaluated. SYM-OSP uBDD disassembles the utility function into multiple BDDs and performs multiple intersections to determine the maximum utility of a set of states, while SYM-OSP uADD uses a single ADD. In practice, ADDs are often better suited to determine the value of a single state in an explicit search, while a collection of BDDs is considered more suitable for symbolic search (Torralba 2015). In addition, we implemented $A^\star_{\text{uADD}}$ (Hart, Nilsson, and Raphael 1968) as an explicit search, representing the utility function as an ADD to determine the utility values of a single state. Algorithm $A^\star_{\text{uADD}}$ with the blind heuristic $h_{\text{blind}}$ is the explicit counterpart of the algorithms SYM-OSP uBDD and SYM-OSP uADD.

**Setup**

All experiments are conducted on the oversubscription planning benchmark set, based on the the optimal track from the International Planning Competition (IPC) 1998-2014 (Katz et al. 2019b), where goal facts are replaced with utilities. In total, the benchmark set consists of four parts, each consisting of the same 57 domains. The difference between the four parts is the cost bound, which for each planning instance is set at 25%, 50%, 75% or 100% of the cost of the optimal or best known solution respectively. Note that several of the domains contain operators with non-unit costs.

For comparison, publicly available planners are chosen that support PDDL (McDermott 2000) and are state of the art in oversubscription planning: branch-and-bound search BNN (Katz et al. 2019a) and $A^\star_{\text{mc}}$ search with multiple cost functions (mc) (Katz and Keyder 2019). Both approaches are based on reformulations that result in planning problems with multiple separate cost functions, but no utility function. The branch-and-bound search BNN can use two separate heuristics, one for guidance, i.e., to decide which node to expand next, and one for pruning nodes. Such heuristics $h_{\text{mc}}$ are computed on the reformulated planning problem with multiple cost functions, while BNN is performed on the original problem. We compare our approach against BNN with the blind heuristic for guidance and the blind heuristic $h_{\text{blind}}$ and the LM-cut heuristic $h_{\text{mc}}^{\text{lmcut}}$ (Helmert and Domshlak 2009) for pruning, as proposed by Katz et al. (2019a). Note that the known best performing configuration uses the blind heuristic $h_{\text{blind}}$ for both heuristics (Katz et al. 2019a). $A^\star_{\text{mc}}$ search maintains multiple cost values for each state, one for each cost function. In addition, $A^\star_{\text{mc}}$ search for oversubscription planning can utilize bound-sensitive heuristics which are able to reason about the primary cost of a solution while taking into account secondary cost functions and bounds (Katz and Keyder 2019). We compare with $A^\star_{\text{mc}}$ search using the blind heuristic $h_{\text{blind}}$, the bound-sensitive max heuristic $h_{\max}^b$ (Bonet and Geffner 1999) and the bound sensitive merge-and-shrink heuristic $h_{\text{m&s}}^b$ (Helmert et al. 2014) introduced in Katz and Keyder (2019).

All experiments are run on a compute cluster with nodes equipped with two Intel Xeon Gold 6242 32-core CPUs, 20 MB cache and 188 GB shared RAM running Ubuntu 18.04 LTS 64 bit. The planners are build with 64 bit and run with a time limit of 30 minutes and memory limit of 4 GB.
Experiments

Table 1 shows the coverage (sum of solved instances) of the presented symbolic approaches compared to explicit heuristic search approaches. Overall, as expected, the performance decreases with increasing cost bounds, since a greater cost bound generally requires more states to be considered and expanded to find a plan and prove its optimality in oversubscription planning. Next, we analyze and discuss the presented symbolic approaches in detail before comparing them to the state-of-the-art approaches that are based on an explicit heuristic search. Finally, we perform a per-domain comparison of all approaches.

Symbolic approaches. Comparing the approaches using symbolic data structures, we can observe that symbolic search with a disassembled utility function in multiple BDDs (SYM-OSP uADD) works best overall. The configuration SYM-OSP uADD, which represents the utility function as a single ADD, performs only slightly worse overall and better in the benchmark sets with smaller cost bounds (25% and 50%). The main advantage of evaluating the utility values of the open list with BDDs compared to ADDs is that the decision diagram library CUDD (Somenzi 2015) uses techniques such as complement edges to store BDDs more compactly (Brace, Rudell, and Bryant 1990). However, complement edges are not used to represent ADDs. Therefore, when the utility function is represented as an ADD, it is necessary to transform the open list represented as a BDD with complement edges into an ADD (without complement edges). Depending on the structure of the BDD, this leads to an overhead which can be avoided when representing the utility function as multiple BDDs. This is consistent with the literature, where representation of numeric functions as BDDs was shown in general to be more suitable for symbolic search (Torralba 2015).

Observe that the time required to create the utility function as ADD is negligible in most instances. In over 6300 of 6668 instances it takes less than one second to create the ADD, with a maximum in the remaining instances of about two minutes. Also, the time for disassembling the ADD into several BDDs (which is only carried out by SYM-OSP uBDD) is not a bottleneck in most instances, because in over 5700 instances this process takes less than one second. However, in 60 instances this procedure lasted more than two minutes.

A comparison of symbolic forward search to explicit search, in which the utility function is represented symbolically, i.e., as ADD, reveals a performance gap. Figure 3 compares the run times of SYM-OSP uADD and its explicit counter part A∗_{uADD} with the blind heuristic h_{blind}. Note that symbolic blind search performs better overall, especially with increasing bounds, where the compact representation gets more important. Most instances of the 100% benchmark set are solved faster by symbolic forward search than by explicit forward search. In a few instances, however, explicit blind search outperforms symbolic blind search. Apart from implementation reasons, this has the explanation that in order to carry out symbolic search, all necessary symbolic data structures, such as the transition relations, have to be created first, which in some cases does not pay off in terms of time and memory. However, the dominance of symbolic search can be traced back to theoretical properties of BDDs, such as the fact that the number of nodes required to represent a given set of states, is at most linear in the number of states and variables. Further, it is possible to represent exponentially many states (in the number of variables) with only a polynomial number of nodes. For these reasons, symbolic search is well suited for exhaustive state space exploration.

<table>
<thead>
<tr>
<th>SYM-OSP uADD</th>
<th>SYM-OSP uADD</th>
<th>SYM-OSP uADD</th>
<th>SYM-OSP uADD</th>
<th>SYM-OSP uADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>h_{blind}</td>
<td>h_{blind}</td>
<td>h_{blind}</td>
<td>h_{blind}</td>
<td>h_{blind}</td>
</tr>
<tr>
<td>26</td>
<td>28</td>
<td>-</td>
<td>0</td>
<td>26</td>
</tr>
<tr>
<td>47</td>
<td>47</td>
<td>-</td>
<td>47</td>
<td>62</td>
</tr>
<tr>
<td>53</td>
<td>51</td>
<td>86</td>
<td>56</td>
<td>76</td>
</tr>
<tr>
<td>38</td>
<td>40</td>
<td>43105</td>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td>36</td>
<td>35</td>
<td>48</td>
<td>0</td>
<td>53</td>
</tr>
<tr>
<td>26</td>
<td>25</td>
<td>25</td>
<td>0</td>
<td>43</td>
</tr>
</tbody>
</table>

Table 2: Per-domain coverage comparison of different algorithms on the complete oversubscription planning benchmark set (228 domains) which is based on the optimal track from the IPC 1998-2014. The entry in row r and column c shows the number of domains in which algorithm r solves more tasks than algorithm c. For each pair of algorithms we highlight the maximum of entries (r, c) and (c, r) in bold.
Explicit heuristic approaches. A comparison between symbolic forward search for oversubscription planning and heuristic search algorithms based on reformulations shows an advantage of symbolic search in terms of overall coverage (Table 1). Here as well, the difference in performance between symbolic search and explicit heuristic search increases with an increasing bound and thus increasing plan length. The natural explanation is that heuristics do not pay off, which is also why blind search performs best and, as mentioned above, symbolic blind search is in practice often preferable to explicit blind search. This observation is underlined by Figure 4, which depicts the number of solved instances over time. While symbolic search solves fewer instances in the first seconds, after a short time it dominates all other approaches, i.e., it solves more instances overall and keeps this performance gap over time. Especially for greater cost bounds, the advantage in overall coverage tends to increase over time (Figures 4c and 4d), while it remains the same for smaller cost bounds (Figures 4a and 4b).

Per-domain comparison. Table 2 shows a per-domain comparison of the different approaches to oversubscription planning. As in classical planning, symbolic and explicit approaches shine in different domains. However, there is a clear advantage to symbolic search over explicit heuristic search overall. Symbolic search solves more instances in more than twice as many domains than any explicit search approach. Nevertheless, there are domains where explicit heuristic search performs better than symbolic search. This shows that, similar to classical planning, a potential portfolio planner of these complementary search strategies could result in a state-of-the-art planner that combines both strengths (Sievers et al. 2019).

Conclusion

We presented a novel approach to optimal oversubscription planning, SYM-OSP, based on symbolic forward search and proved that SYM-OSP is sound, complete and optimal. Our empirical evaluation shows that SYM-OSP performs better than other state-of-the-art approaches, for all tested cost bounds. In addition, SYM-OSP scales to larger cost bounds better than other approaches, making SYM-OSP suitable regardless of the cost bound. However, as usual, there are domains, where explicit heuristic search performs better than symbolic search.

For future work, we would like to investigate symbolic backward search for oversubscription planning, a necessary condition for both performing bidirectional symbolic search and a possible application of symbolic abstraction heuristics (Edelkamp 2002) to oversubscription planning. Finally, applying symbolic branch-and-bound search (Edelkamp and Kissmann 2009; Kissmann 2012) to oversubscription planning is an interesting research direction.
Acknowledgments

David Speck was supported by the German Research Foundation (DFG) as part of the project EPSDAC (MA 7790/1-1). We thank the anonymous reviewers for their comments and suggestions.

References


