Value Driven Landmarks for Oversubscription Planning

Daniel Muller and Erez Karpas
Technion - Israel Institute of Technology
Haifa, Israel

Abstract

Oversubscription planning is the problem of choosing an action sequence which reaches a state with a high utility, given a budget for total action cost. Most previous work on oversubscription planning was restricted to only non-negative utility functions and 0-binary utility functions. While this restriction allows using techniques similar to partial satisfaction planning, it limits the expressivity of the formalism. In this paper, we address oversubscription planning with general additive utility functions over a finite-domain representation. We introduce the notions of net utility of an action, and of a gross positive action. Using these notions, we prove several properties about the structure of an optimal plan, which are then compiled into a classical planning problem. The landmarks of this classical planning problem are value driven landmarks of the original oversubscription problem, that is, they must occur in any action sequence which improves utility. An empirical evaluation demonstrates that these landmarks are more informative than previous state-of-the-art methods for landmark discovery for oversubscription planning, and lead to better planning performance.

Introduction

Oversubscription planning (Smith 2004), hereinafter referred to as OSP, deals with achieving a state with a high utility value, given a budget on total action cost. This formulation allows us to handle situations with under-constrained resources, which do not allow us to achieve all possible goal facts. This is in contrast to classical planning, in which the objective is to find a cheap plan which achieves all goal facts. In optimal OSP and optimal classical planning, the tasks are further constrained to finding a path which achieves a state with maximal utility, and, to finding the cheapest cost path which achieves the goal, respectively.

Over the years, the theory and practice of classical planning have been studied and advanced much more intensively compared to OSP. Recent work (Mirks and Domshlak 2013; 2014; Domshlak and Mirks 2015) made several contributions aiming at improving the scalability of OSP solvers. In particular, they developed a planner which exploits standard landmark discovery tools of classical planning, as well as abstractions for solving OSP problems. In this paper, we propose a technique which allows us to discover more informative landmarks than previous methods. Furthermore, our technique is applicable for general additive utility OSP tasks. More specifically, we introduce the notion of the net utility of an action, which is the net change in utility after applying this action. We prove that any optimal plan must end with an action which has positive net utility. We also introduce the notion of a gross positive action, which improves the utility for at least one of the state variables, although possibly at the cost of a lower utility for other variables. We then combine these and prove that any optimal plan must contain a gross positive action (which might have negative net utility), which achieves some fact that is maintained until the end of the plan, which must be an action with positive net utility. This approach of improving differs from previous approaches which attempted to collect valuable facts and allow us to handle general additive utility functions, rather than only non-negative utilities.

To leverage these properties, we introduce a compilation of an OSP task to classical planning, which forces a plan to conform to these properties. The landmarks of this classical planning task constitute value driven landmarks (called ε-landmarks by Domshlak and Mirks (2015)), that is, landmarks which must occur in every plan that improves over the utility of the initial state. We also introduce a slight modification to the inc-compile-and-BFBB planner (Domshlak and Mirks 2015), which leverages our definitions of net positive action and gross positive actions and our observations on these definitions. An empirical evaluation shows that the landmarks we discover are more informative than those of Domshlak and Mirks (2015), and that our modified planner outperforms state-of-the-art OSP planners.

Preliminaries

Some auxiliary notation first. For $k \in \mathbb{N}^+$, by $[k]$ we denote the set $\{1, 2, \ldots, k\}$. An assignment of a variable $v$ to value $d$ is denoted by $\langle v/d \rangle$ and referred as a proposition or fact. For a partial assignment $p$ to $V$, let $\mathcal{V}(p) \subseteq V$ denote the subset of variables instantiated by $p$, and, for $v \in \mathcal{V}(p)$, $p[v]$ denote the value provided by $p$ to the variable $v$.

Planning Framework. In what follows we adopt the OSP model representation of Domshlak and Mirks (2015), adopting a language close to the SAS+ language for classical
planning (Bäckström and Klein 1991; Bäckström and Nebel 1995). A deterministic over specification planning (OSP) task is given by a sextuple \( \Pi = (V, s_0, o; O, c, b) \), where \( V = \{v_1, \ldots, v_n\} \) is a finite set of finite-domain state variables, with each complete assignment to \( V \) representing a state, and \( S = \text{do}(v_1) \times \cdots \times \text{do}(v_n) \) being the state space of the task; \( s_0 \in S \) is a designated initial state; \( u \) is an efficiently computable state utility function \( u: S \rightarrow \mathbb{R}; O \) is a finite set of actions, with each action \( o \in O \) being represented by a pair \( \langle \text{pre}(o), \text{eff}(o) \rangle \) of partial assignments to \( V \), called preconditions and effects of \( o \), respectively; \( c: O \rightarrow \mathbb{R}^{\geq 0} \) is an action cost function; \( b \in \mathbb{R}^{\geq 0} \) is a cost budget allowed for the task. Action \( o \) is applicable in a state \( s \) if \( s[v] = \text{pre}(o)[v] \) for all \( v \in \mathcal{V}(\text{pre}(o)) \). Applying \( o \) changes the value of each \( v \in \mathcal{V}(\text{eff}(o)) \) to \( \text{eff}(o)[v] \), and the resulting state is denoted by \( s[o] \). Sequential application of actions \( \langle o_1, \ldots, o_m \rangle \) denoted by \( \pi \), called a plan for \( s \) if it is applicable in \( s \) and \( c(\pi) \leq b \). We assume a general additive utility function with multi-valued variables, defined as \( u(s) = \sum \{v/d \in u_v(d), \text{with } u_v(d) \in \mathbb{R} \text{ for all variable-value pairs } v/d \} \). That is, negative utility values are allowed and each variable can get many value assignment with different utility available for each variable-value pair.

A deterministic classical planning task, is given by \( \Pi = (V, s_0, G; O, c) \), where the utility function and cost budget are replaced with a hard goal constraint which is a partial assignment on state variables \( G \subseteq V \).

**Disjunctive Action Landmarks.** Fact landmark are propositions that must be true at some point in every solution plan for a given planning task (Hoffmann, Porteous, and Sebastia 2004). Similarly to landmarks over facts, disjunctive action landmarks (Vidal and Geffner 2006; Zhu and Givan 2004), correspond to a set of actions such that every plan contains at least one action from that set. While landmarks are widely used in (both satisficing and optimal) classical planning (Karpas and Domshlak 2009; Helmert and Domshlak 2009; Coles and Coles 2011; Domshlak, Katz, and Lefler 2012; Bonet and Helmert 2010; Pommerening and Helmert 2013), the first framework for their exploitation in the context of OSP has been introduced by Domshlak and Mirkis (2015) only recently. This framework basic construct is the notion of \( \varepsilon \)-landmarks.

An \( \varepsilon \)-landmark for a state \( s \) is a set of actions that is applied in every plan \( \pi \) that achieves something valuable, that is, by every plan having a positive value. For instance, with the disjunctive action landmarks we use here, if \( L \subseteq O \) is an \( \varepsilon \)-landmark for \( s \), then every plan having a positive value end-state contains an action from \( L \).

\( \varepsilon \)-landmarks can be used to construct a budget reduction compilation (Domshlak and Mirkis 2015), as follows:

- **Given an OSP task \( \Pi \), we generate the \( \varepsilon \)-compilation \( \Pi_{\varepsilon} \) of \( \Pi \), where \( \Pi_{\varepsilon} \) is a classical planning task that simply extends the structure of \( \Pi \) with a set of zero-cost actions such that applying any of them corresponds to verifying that a positive value can be achieved in \( \Pi \).

- Using classical planning tools, a set of landmarks \( \mathcal{L} \) is generated for \( \Pi_{\varepsilon} \), along with an admissible landmark cost function \( l\text{cost}: \mathcal{L} \rightarrow \mathbb{R}^{\geq 0} \) where \( \sum_{L \in \mathcal{L}} l\text{cost}(L) \leq h^*(s) \) (Katz and Domshlak 2010; Karpas and Domshlak 2009; Helmert and Domshlak 2009).

- Performing budget reducing compilation by compiling \( (\mathcal{L}, l\text{cost}) \) into \( \Pi \), obtaining an OSP task \( \Pi_{\varepsilon} \), where \( \Pi_{\varepsilon} \) extends the structure of \( \Pi \) by mirroring the actions of each \( \varepsilon \)-landmark \( L_i \) with their “cheaper by \( l\text{cost}(L_i) \)” versions. To compensate for the discounted actions, budget \( b \) is reduced by \( \sum_{i=1}^{|\mathcal{L}|} l\text{cost}(L_i) \). Domshlak and Mirkis (2015) devised two versions of polynomial and sound budget reducing compilation; A basic compilation \( \Pi_{\varepsilon} \) applied on a pairwise disjoint landmarks and extended with auxiliary control structure, a generalized budget reducing compilation applied on an arbitrary (non-disjoint) sets of \( \varepsilon \)-landmarks. This compilation can then be used within a heuristic search algorithm, as described next.

**Heuristic Search for OSP.** Best-First-Branch-and-Bound (BFBB) for OSP must rely on admissible utility-upper-bounding heuristic function (within budget \( b \) restrictions) \( h: S \times \mathbb{R}^{\geq 0} \rightarrow \mathbb{R}^{\geq 0} \) to estimate the true utility \( h^*(s, b) \).

BFBB maintains a queue in decreasing order of \( h(s(n), b - g(n)) \) and the best solution \( n^* \) found so far. All generated nodes \( n \) evaluated no higher than \( h(n) \) of nodes with cost-so-far \( g(n) \) higher than the problem’s budget \( b \) are pruned. When the node-list becomes empty, or the node selected from the list promises less than the lower bound, the plan associated with the best solution \( n^* \) is returned. If the heuristic \( h \) is admissible, then the returned plan is guaranteed to be optimal.

inc-compile-and-BFBB (Domshlak and Mirkis 2015) is an incremental version of BFBB. Additionally to the best solution so far \( n^* \), inc-compile-and-BFBB maintains a set of reference states \( S_{\text{ref}} \), which is updated with any new, best-do-far state which has reached \( n^* \). For each \( s_i \in S_{\text{ref}} \) and each valuable fact \( g \in V \setminus s_i \) a zero cost action is added to \( O \) which indicates for a valuable fact that is not in \( s_i \). Applying all these zero cost actions verifies an achievement of subset of good facts that included in no state from \( S_{\text{ref}} \). With each update of \( S_{\text{ref}} \), BFBB is restarted with a new set of \( \varepsilon \)-landmarks created on the basis of the current \( S_{\text{ref}} \), that is derived concerning achieving something that has not yet been seen so far. These new, more informed \( \varepsilon \)-landmarks are compiled back into the original task for next iteration.

**The Properties of Optimal Plans**

In this section, we present several properties of the structure of optimal plans. In the following section, we present a full compilation and a solver exploiting these properties. We will refer to a state that improves over the utility of the initial state as a valuable state, and to a sequence of actions reaching such state as a valuable plan.

**The Utility of Actions**

Considering negative utility values in OSP, because of the dependencies between propositions, it is possible that collecting positive utility values might require collecting negative utility values on the way. In particular, it is possible that the same action achieving positive value propositions also achieves some negative value propositions. For
an OSP action \( o \), the total outcome utility \( u^{\text{out}}(o) = \sum_{v \in \text{V}(\text{eff}(o))} u_{\text{v}}(\text{eff}(o)[v]) \), capture these inter-state variables dependencies in a wider context.

Negative interactions are not only confined to problems with variables of negative utility. Even when variables take a non-negative form, an action achieving valuable propositions might lose more valuable propositions that have been held at the origin state. The net utility of an action definition provides a wider context of achievement evaluation, by taking in account both target state and its origin, and captures the intra-state dependencies, in addition to the inter-state dependencies (which captured with the total outcome utility). For clarity of presentation, we present the net utility of an action in SAS representation, under the assumption that for every effect variable a precondition is defined, so the notation is well defined. Later, we relax our simplifying assumption with the selective action split. The net utility of an action \( o \) is defined as follows:

**Definition 1.** For an OSP planning task action \( o \), the net utility of \( o \) is \( u(o) = \sum_{v \in \text{V}(\text{eff}(o))} [u(\text{eff}(o)[v]) - u(\text{pre}(o)[v])] \).

The definition of net utility will help us to derive more informative \( \varepsilon \)-landmarks, and further reduce the search space.

**High Level Overview**

In what follows, we present an interrelated set of properties of optimal plans, which allow us to define a pattern of valuable plans in OSP tasks. Our landmark generation method exploits these optimal plan properties on the level of actions and sequences of actions, in contrast to the traditional “collecting” approach which generates landmarks to valuable facts, which based on properties of the effect list of a single action. These properties are used to generate refined and lengthier landmarks which are the input for Mirkis and Domshlak budget reduction procedure and improve the pruning mechanism which is based on these landmarks. A running example in Figure 1 shows all plans for a given OSP task and will be referred to demonstrate these properties. In this OSP task an action utility function is defined over three state variables with the same domain \( \text{dom} = \{ A, B, C, D \} \), by \( u(A) = 0 \), \( u(B) = 1 \), \( u(C) = 2 \) and \( u(D) = 3 \) for each variable.

In Lemma 1, we prove that any valuable plan terminates with a net positive utility action. The colored in blue arrows illustrated in Figure 1(b) indicate such net positive utility action candidates to finish the search with an optimal state in hand. The green colored actions depicted in Figure 1(c) are gross positive actions, whose effects improve over at least one of the variables in the initial state, and may have total negative utility by achieving worse utility for some other variables. We prove in Lemma 2 that such actions must occur in any optimal plan, prior to a net positive action application which terminates the search.

A special case is an action that is both gross positive and has positive net utility, for example, action \( o_{10} \) in Figure 1(d). In a strictly positive utility setting with an 0-binary utility function and an initial state with utility zero, the net positive actions are also gross positive. However, in general additive utility settings, these actions are likely to be distinct. For example, the net positive utility actions \( o_{5} \) and \( o_{9} \) in our running example lead to states with lower utility compared to the initial state. A maintained achievement property, which we prove in Lemma 3 complements Lemma 2 and ensures that an achieved effect of a gross positive action remains in hand when net positive actions are examined as candidates to terminate the search.

Finally, in Theorem 4, we combine all these claims and define a so-called ‘window of opportunity’ to terminate a plan with a valuable state in hand, as illustrated in Figure 1(d). We construct a classical planning task, which encodes the above mentioned constraints on the ‘window of opportunity’. The landmarks for this classical planning task, are value landmarks which guide us towards a valuable state in the original OSP task.

**Net Positive Actions**

At first glance, the net utility value of an action does not have a clear role in our reasoning; It is very much possible that optimal plans will contain many actions with negative net utility since it allows for achieving valuable facts that
outbalance the respective loss. However, Lemma 1 shows that 
actions with positive net value do play a characteristic role in optimal plans. The proof of Lemma 1 is rather simple and, in plain words, the lemma states that, for each plan \( \pi \), there is a plan \( \pi' \) that: (i) ends with a positive net utility action, (ii) is at most as costly as \( \pi \), and (iii) is at least as valuable as \( \pi \).

**Lemma 1.** Given an OSP task \( \Pi \) with a general additive utility function \( u \), for any plan \( \pi \) for \( \Pi \) such that \( u(s[\pi]) \) > \( u(s_0) \), there is a prefix \( \pi' \) of \( \pi \) such that: 1. \( u(s[\pi']) \leq u(s[\pi]) \), and 2. for the last action \( o_{last} \) along \( \pi' \), we have \( u(o_{last}) > 0 \).

**Proof.** The proof is by induction on the plan length \( n \). For \( n = 1 \), we have \( \pi = (o_1) \), and since \( u(s[\pi]) > u(s_0) \), the operator \( o_1 \) has a positive net utility value. Hence, \( \pi' = \pi \) satisfies the claims. Assuming that the claim holds for \( n \geq 1 \), we now prove it for \( n + 1 \).

Considering a plan \( \pi = (o_1, \ldots, o_{n+1}) \), for \( i \in [n+1] \), let \( \pi_i \) denote the prefix of \( \pi \) consisting of its first \( i \) operators. If the last operator \( o_{n+1} \) along \( \pi \) has a positive net utility value, then we are done with \( \pi' = \pi \). Otherwise, if \( o_{n+1} \) has either negative or zero net utility value, then \( u(s[\pi]) \) > \( u(s_0) \) in particular implies \( u(s[\pi_i]) \) > \( u(s_0) \). If \( \pi' \) is a prefix of \( \pi_n \) that satisfies the lemma by our assumption of induction, then \( \pi' \) also satisfies the lemma with respect to \( \pi \) since \( u(\pi') > u(s[\pi]) \).

Relying on Lemma 1, we can now extract \( \varepsilon \)-landmarks which focus on actions with net positive utility value as candidates to end the search process. Specifically, we can devise a polynomial time compilation to classical planning, that extends the structure of \( \Pi \) with a goal variable \( g \), and for any net positive action \( o \), the effects are extended with the proposition \( \langle g/1 \rangle \). Applying any of those actions indicates a positive value improvement in \( \Pi \). With this compilation we can use any method for classical planning landmark extraction to derive more informative \( \varepsilon \)-landmarks for \( \Pi \), due to a refinement of goal-achieving actions.

**Gross Positive Actions**

Considering tasks with a valuable initial state \( s_0 \), a net positive utility action might be applied during the search, leading to a valuable state \( s \) with lower utility \( u(s) < u(s_0) \). Since landmarks hold for any valuable plan, they must hold for a plan reaching state \( s \).

We are no longer interested in a search of plans that “achieve something”, but in search for plans that achieve an improvement over the initial state \( s_0 \).

**Definition 2.** Given an OSP task \( \Pi = \langle V, s_0, u; O, c, b \rangle \) and a state \( s \), let \( \text{improved}(s) \) be the set of all propositions \( \langle v/d \rangle \in D \) such that \( u_v(d) > u_v(s[v]) \).

Based on the definition of \( \text{improved}(s) \) we define a gross positive action, which achieves an improvement over at least one variable from the initial state.

**Definition 3.** Given an OSP task \( \Pi = \langle V, s_0, u; O, c, b \rangle \) and a state \( s \), the gross positive actions relative to \( s \) \( O_{\text{gPos}}(s) \subseteq O \) of \( \Pi \) are \( O_{\text{gPos}}(s) = \{ o \in O, \text{eff}(o) \cap \text{improved}(s) \neq \emptyset \} \).

For now, we will use the initial state \( s_0 \) as the reference state to improve over. Lemma 2 is based on the definition of the gross positive actions and shows that gross positive actions play a characteristic role in optimal plans. The proof of Lemma 2, in plain words, claims that, along any valuable plan \( \pi \) for an OSP task \( \Pi \) that improves over the given state, there is at least one gross positive action, which improves the utility for at least one of the state variables (although possibly at the cost of a lower utility for other variables).

**Lemma 2.** Given an OSP task \( \Pi = \langle V, s_0, u; O, c, b \rangle \), and a state \( s \), for any plan \( \pi \) that improves over \( s \), that is, \( u(s[\pi]) > u(s) \), it holds that: 1. \( s_0[\pi] \cap \text{improved}(s) \neq \emptyset \), and 2. \( \pi \cap O_{\text{gPos}}(s) \neq \emptyset \), that is, at least one action along \( \pi \) is gross positive.

**Proof.** Assuming to the contrary that \( s_0[\pi] \cap \text{improved}(s) = \emptyset \), by definition of \( \text{improved}(s) \) we have \( u_v(s[\pi]) \leq u_v(s[v]) \), and thus \( \sum_{(v/d) \in \pi} u_v(d) \leq \sum_{(v/d) \in s_0} u_v(d) \) and, in turn, \( u(s[\pi]) \leq u(s) \), contradicting the assumption of the lemma that \( \pi \) improves over \( s \).

Since \( s_0[\pi] \cap \text{improved}(s) \neq \emptyset \), consider any proposition \( \langle v/d \rangle \in s_0[\pi] \cap \text{improved}(s) \). By definition of \( \text{improved}(s) \), \( \langle v/d \rangle \neq \emptyset \), \( s \pi \) must contain an operator \( o \) achieving \( \langle v/d \rangle \), and by definition of \( O_{\text{gPos}}(s) \), \( o \in O_{\text{gPos}}(s) \).

Applying any of those actions indicates a valuable achievement in \( \Pi \). As illustrated in Figure 1(c) in our running example, the number of the landmarks derived with focus on gross positive utility actions can be substantially increased.

**The Window of Opportunity to Improve**

In tasks with an initial state carrying the lowest possible utility, always, all the constraints are satisfied by the last net positive action application, which makes Lemma 1 a sufficient condition. When it comes to tasks with an initial state utility \( u(s_0) \) greater than the lowest possible, this set of separate events is likely to happen. When the gross positive action and the net positive action are different events, achievements may be lost between the occurrences of both of them.

Furthermore, Lemma 1 is stated concerning the net utility of an action, which is a relative term. The decision if the net utility of an action is enough to improve the initial state, depends on which state the action is applied in. When the initial state is the lowest possible, obviously, every net positive utility action is a good indicator for improvement. On the other hand, gross positive actions are defined in absolute terms, and allow for evaluation of achievements in relation to a reference state, in the context of a sequence of actions rather than a single action. The integration of those two properties to criteria for optimal plans with any initial utility state is through timing. The occurrence of a gross positive action establishes a dynamic reference point. Once a reference point is established, a net positive utility improvement must occur in relation to that point. The length of the ‘window of opportunity’ to achieve an improvement over the initial state is defined by the time that the achievement of the gross positive action keeps holding, that is, an improved proposition (or a set of propositions) over the initial state.
from improved \((s_0)\) holds. When there is no achievement yet, or the achievement is not maintained, the net positive utility actions (which are not gross positive) do not bring a sustainable (global) benefit. To acquire more accurate landmarks, we have to capture the information about the occurrence of an effective gross-positive achievement, which allows for recognition of the ‘zone’ in which an effective search can terminate. A net positive action that is applied out of that ‘zone’ can be safely treated as just an intermediate action along with a plan.

**Maintained Achievements.** In Lemma 3 we define the maintained proposition property of sequential application of actions in a deterministic scenario. This notation is similar to the *maintenance goals* notion, which is used in the context of temporal planing (Haddawy and Hanks 1993; 1998). The proof of Lemma 3 is rather simple, it finalizes the claim in Lemma 2 by defining the ‘goal area’ where the plan may terminate with achievement in hand. In plain words, Lemma 3 states that for each improving plan \(\pi\) over the initial state there is a suffix plan \(\pi’\) such that (i) starts with a gross positive action \(o_{grs}\), and (ii) a valuable proposition achieved by \(o_{grs}\) is maintained along \(\pi’\).

**Lemma 3.** Given an OSP task \(\Pi = (V, s_0; u, O, c, b)\), and a state \(s\), for any plan \(\pi\) that improves over \(s\), that is, \(u(s_0[\pi]) > u(s)\), there is a suffix \(\pi’\) of \(\pi = \tau_0 \cdots \tau_i\) such that: 1. \(\pi’\) starts with a gross positive action \(o_{grs}\), and 2. for some \(\langle v/d, o_o \rangle \in \text{improved}(s)\), \(\langle v/d, o_o \rangle\) is maintained along \(\pi’\), that is, for each non-empty prefix \(\pi''\) of \(\pi’\), \(\langle v/d, o_o \rangle \in s_0[\tau_0 \cdots \tau_i]\).

**Proof.** Let \(\pi = \langle o_1, \ldots, o_n \rangle\) be a plan for \(\Pi\) such that \(u(s_0[\pi]) > u(s)\). By Lemma 2, we have \(s_0[\pi] \cap \text{improved}(s) \neq \emptyset\), and for each \(\langle v/d, o_o \rangle \in s_0[\pi] \cap \text{improved}(s)\), the plan \(\pi\) contains an operator achieving \(\langle v/d, o_o \rangle\). Consider any such proposition \(\langle v/d, o_o \rangle \in s_0[\pi] \cap \text{improved}(s)\), and let \(o_i, i \in [n]\), be the last instance of an operator along \(\pi\) that achieves \(\langle v/d, o_o \rangle\). Since \(\langle v/d, o_o \rangle \in s_0[\pi]\), then no operator instance along the suffix \(\pi’ = \langle o_1, \ldots, o_k \rangle\) of \(\pi\) changes the value of \(v\), and thus \(\pi’\) satisfies the claim of the lemma. \(\square\)

**Synergistic Criteria for a Valuable Plan.** Given an OSP task \(\Pi\) with a general additive utility function and an initial state \(s_0\), considering a plan \(\pi\) for \(\Pi\) with \(u(s_0[\pi]) > u(s_0)\), from Lemma 1, Lemma 2 and Lemma 3 we know the following properties for a valuable plan \(\pi\): (1); the last action \(o_o\) along \(\pi\), we have \(u(o_o) > 0\), (2): \(\pi\) contains at least one gross positive action, and (3): an achievement of at least one gross positive action is maintained, that is, holds along a suffix of \(\pi\). Putting it all together, we are now able to devise additional constraints on valuable plans for OSP tasks. Any valuable plan \(\pi\) has a valuable partial plan.

Consider a plan \(\pi = \langle o_1, \ldots, o_i, \ldots, o_n \rangle\), with \(i, j \in [n]\), such that \(i < j \leq n\) and a valuable proposition \(\langle v/d, o_o \rangle \in \text{eff}(o_o)\), such that \(\langle v/d, o_o \rangle \in \text{improved}(s_0)\), let the end-state be \(s = s_0[\pi]\) such that \(u(s) > u(s_0)\). Suppose that, an action \(o_j\) such that \(\langle v/d, o_o \rangle \in \text{pre}(o_j)\) and \(\langle v/d', o_o \rangle \in \text{eff}(o_j)\), where \(d \neq d', v \neq v'\) and \(v/d, v/d' \in \text{improved}(s_0)\). In the example, the proposition \(\langle v/d, o_o \rangle \in \text{improved}(s_0)\) is achieved by an intermediate action \(o_j\) to satisfy a precondition of action \(o_o\) and lost immediately after, i.e. not maintained. Obviously, the proposition \(\langle v/d, o_o \rangle \in \text{improved}(s_0)\) must be re-acquired again, and the plan may include another occurrence of this action, that is, \(\pi = \langle o_1, \ldots, o_i, \ldots, o_j, \ldots, o_n \rangle\) so that \(\langle v/d, o_o \rangle\) will
hold at the end-state, that is, \((v/d) \in s \cap \text{improved}(s_0) \subseteq s\). Hence, multiple occurrence of gross positive action \(o_j\) in the action sequence \(\pi = \langle o_1, \ldots, o_j, \ldots, o_i \rangle\), may miss landmarks for this sequence.

To acquire more accurate landmarks we have to capture this information in the compiled classical planning task \(\Pi\). We allow for multiple action repetition by duplicating gross positive actions and extending each with an auxiliary control indicator \(g_{grs}\) as follows: The action \(o_{nm}\) is associated with a proposition \((v/d)\) that is not-maintained, and constructed as \(\text{pre}(o_{nm}) = \text{pre}(o) \cup \langle g_{grs}/0,0 \rangle, \text{eff}(o_{nm}) = \langle g_{grs}/1,0 \rangle, c(o_{nm}) = c(o)\). The action \(o_m\) is associated with a maintained proposition \((v/d)\), \(o_{nm}\) can be applied only once along plan \(\pi\) and constructed as \(\text{pre}(o_m) = \langle g_{grs}/0,0 \rangle, \text{eff}(o_m) = \langle g_{grs}/1,0 \rangle, c(o_m) = c(o)\). Since \(o\) is split to \(o_{nm}\) and \(o_m\), it may occur in different landmarks.

### The Valuable Plan Compilation

Relying on Theorem 4, in what follows, we focus on the value driven landmarks for an OSP task \(\Pi\).

**Definition 4.** Let \(\Pi = (V, s_0, w; O, c, b)\) be an OSP task and \(\text{improved}(s)\)(with \(s_0\) at the start) be the set of all propositions \(\langle v/d \rangle \in D\) such that \(u_c(v) > u_c(s'[v])\). The valuable plan compilation of \(\Pi\) is a classical planning task \(\Pi' = (V', s_0', G'; O', c', b')\) constructed as \(V' = V \cup E \cup Y\) where \(E = \{\text{srch}/1,\text{get/1}, \text{end/1}, g_{grs}g_{net}/1\}\) with \(dom = \{1,0\}\) and, \(Y = \{y_{grs}^v \mid v \in V, d \in dom(v)\}\). \(s_0' = s_0 \cup \{\langle y_{grs}^v/0 \rangle \mid y_{grs}^v \in Y, d \in dom(v)\}\). \(G' = \{\langle g/1 \rangle\}\) and \(O'\) constructed of sets of actions as follows:

- \(\{o_k \mid o \in O, u(o) > 0, \text{eff}(o) \land \text{improved}(s) \neq \emptyset\}\) where \(o_k = \langle\text{pre}(o) \cup \langle g_{grs}/0,0 \rangle \cup \langle g_{net}/0,0 \rangle, \text{eff}(o) \cup \langle g_{grs}/1,0 \rangle, c(o)\rangle\)
- \(\{o_{nm} \mid o \in O, u(o) > 0\}\) where \(o_{nm} = \langle\text{pre}(o) \cup \langle g_{grs}/1,0 \rangle, \text{eff}(o) \cup \langle g_{net}/1,0 \rangle, c(o)\rangle\)
- \(\{o_{nm} \mid o \in O, u(o) \leq 0, \text{eff}(o) \land \text{improved}(s) = \emptyset\}\) where \(o_{nm} = \langle\text{pre}(o) \cup \langle \text{end}/1,0 \rangle \cup \langle g_{net}/0,0 \rangle, \text{eff}(o) \cup \langle g_{net}/1,0 \rangle, c(o)\rangle\)
- \(\{o_{nc} \mid o \in O, u(o) \leq 0, \text{eff}(o) \land \text{improved}(s) = \emptyset\}\) where \(o_{nc} = \langle\text{pre}(o) \cup \langle \text{srch}/1,0 \rangle, \text{eff}(o) \cup \langle g_{net}/1,0 \rangle, c(o)\rangle\)
- \(\text{finish}_k = \langle\text{get/1} \cup \langle \text{srch}/0,0 \rangle, \text{eff}(\text{finish}) = 0\rangle\)
- \(\text{finish}_{G_k} = \langle\text{get/1} \cup \langle \text{srch}/0,0 \rangle, \text{eff}(\text{finish}_{G_k}) = 0\rangle\)
- \(\{o_{grsc}, o_{net}^{grsc} \mid (v/d) \in \text{improved}(s)\}\) where \(o_{grsc} = \langle\text{pre}(o) \cup \langle g_{grs}/0,0 \cup \langle \text{end}/0,0 \rangle \cup \langle y_{grs}^v/0 \rangle, \text{eff}(o_{grsc}) = 0\rangle\)
- \(o_{net} = \langle\text{pre}(o) \cup \langle g_{net}/1,0 \rangle, \text{eff}(o_{net}) = 0\rangle\)
- \(o_{net}^{grsc} = \langle\text{pre}(o) \cup \langle g_{net}/0,0 \cup \langle y_{grs}^v/0 \rangle, \text{eff}(o_{net}^{grsc}) = 0\rangle\)

In plain words, \(\Pi\) extends the structure of \(\Pi\) with a zero cost auxiliary actions and a zero utility auxiliary propositions. This auxiliary control structure allows for transformation of an OSP task to a certain classical planning task, that reflects the original OSP task properties in a way that they will be considered by the classical planning landmarks generation procedure. A valuable state for an OSP task \(\Pi\) with an initial state \(s_0\) can be acquired with a set of events. Striving to extract refined landmarks of good quality, it will be beneficial to provide the landmark extraction procedure with as much as possible information about the events and the constraints that must hold in any valuable plan for an OSP task, that is, as accurate as possible auxiliary control structure. The auxiliary control structure constructed as follows:

- the propositions \(\text{srch}, \text{get/1}, \text{end/1}, g_{grs}, g_{net}\) and \(g\) together with the actions \(o_{grsc}, o_{net}^{grsc}\) and \(\text{finish}\) enforce the achievements with respect to constraints from Theorem 4.
- the goal proposition \(g\) can be achieved directly by a net positive utility action, which is also gross positive action.
- the artificial goal proposition \(g\) can be achieved by a net positive utility action, which is not a gross positive action, only after the verification phase is terminated, that is, the actions \(O_{grsc}, O_{net}^{grsc}\) and \(\text{finish}\) applied. The verification phase is as follows: 1. regular actions can be applied only before \(\text{finish}\) action. 2. \(O_{grsc}\) actions can be applied only after \(\text{finish}\) action. 3. last occurrence of net utility actions (with \(g_{net}\) verification) can be applied only after \(O_{grsc}\) action. 4. goal achieving actions can be applied only with \(g_{net}^{grsc} \in O_{net}^{grsc}\) action.

With this auxiliary control structure, the first part of any plan for \(\Pi\) determines a plan for \(\Pi\), the second part “verifies” that the end-state of that plan achieves a subset of valuable propositions from \(\text{improved}(s_0)\), the third part terminates the search with a net positive utility action. Finally, the last part grantees that improved \(\text{set}(s) \cap s_0[s] \neq \emptyset\) by verifying that the valuable proposition that initiated the verification phase, have not been lost during verification phase.

Constructing \(\Pi\) from \(\Pi\) is trivially polynomial time, it allows for effective discovery of value driven landmark for \(\Pi\) using the standard machinery for landmark discovery.

**Theorem 5.** For any OSP task \(\Pi\), any landmark \(L\) for \(\Pi\) such that \(L \subseteq O\) is a value-landmark \(L\) for \(\Pi\).

With Theorem 5\(^4\) in hand, we can derive \(\epsilon\)-landmarks for \(\Pi\) using classical planning landmark extraction methods. **Incorporating the Compilation into the Planner.** Based on the compilation described above, we modified the inc-compile-and-BFBB (Domshlak and Mirikis 2015) planner in two ways. First, we used the landmarks extracted from our compilation rather than the original compilation (Domshlak and Mirikis 2015). Second, instead of maintaining the set of good facts, we modify our landmark extraction technique to refer only to the best solution found so far, \(n^*\).

Specifically, gross positive actions in the second iteration and onward refer to an improvement relative to \(n^*\), rather than to the initial state \(s_0\).

\(^4\)Due to space limitations, relegated to a full technical report.

**Selective Action Split**

In a SAS\(^+\) action, a variable could have an effect defined on it without having a precondition. This makes computing the
net utility of the action impossible, as the net utility depends on the state where the action is applied. In what follows we release our simplifying assumption of actions being represented in SAS.

Let us denote by \( S_a \) the set of states in which action \( a \) is applicable, and by \( S_a^{\pi} \) the set of states in which \( a \) is applicable and has positive net utility. Since \( a \) is represented in SAS, \( S_a^{\pi} \) is a Cartesian product of all possible propositions of the variables for which \( a \) has no preconditions, with \( a \)’s preconditions. Let us also denote the set of variables which appear in \( a \)’s effects but not in its preconditions by \( V_a \). We could create multiple copies of \( a \), one for each possible assignment to \( V_a \), and compute for each of these whether it has positive net utility or not.

However, since we are only interested in checking whether the net utility is positive or not, we can potentially reduce the number of copies of \( a \) needed. Conceptually, we can think of a propositional formula expressed in DNF over the propositions associated with \( V_a \), whose satisfying assignments are exactly those in which \( a \) has positive net utility, and another DNF formula whose satisfying assignments are those where \( a \) has non-positive net utility. Then it is enough to create a copy of \( a \) for each clause in these two DNF formulas, which guarantees that each copy always has either positive or non-positive net utility. Our implementation exploits mutual exclusion relations and cases where the utility of a variable is constant, to reduce the set of possible assignments even further, but we omit the exact details for the sake of brevity.

**Empirical Evaluation**

As OSP still lacks a standard suite of benchmarks for comparative evaluation, we have cast in this role the STRIPS classical planning tasks from the International Planning Competitions (IPC) 1998-2006. This “translation” to OSP was done by associating a separate value with each proposition in the corresponding classical IPC task.

In order to transform a classical planning task \( \Pi \) into a set of OSP tasks, we first solve \( \Pi \) optimally using Fast Downward (Helmert 2006) using the admissible landmarks heuristics (Karpas and Domshlak 2009) with a timeout of 10 minutes. Let us denote the cost of an optimal solution to \( \Pi \) by \( c^* \). Similarly to (Domshlak and Mirkis 2015) we then generate several different OSP tasks, with budgets corresponding to 25%, 50%, 75%, and 100% of \( c^* \). To complete the process, we must then assign a utility to each proposition in \( \Pi \). We used 2 different methods of assigning utilities:

**non-negative** Each proposition \( \langle v/d \rangle \) of multi-valued variable \( v \in V \) was assigned a utility as follows:

\[
    u(\langle v/d \rangle) = \begin{cases} 
    2 & \text{if } \langle v/d \rangle \text{ is a goal proposition in } \Pi \\
    1 & \text{if } v \text{ is a goal variable in } \Pi \\
    & \text{but } \langle v/d \rangle \text{ is not a goal proposition} \\
    0 & \text{otherwise}
    \end{cases}
\]

**negative** Let \( s \) be the state reached by an optimal solution of \( \Pi \). Then

\[
    u(\langle v/d \rangle) = \begin{cases} 
    1 & \text{if } \langle v/d \rangle \text{ is a goal proposition in } \Pi \\
    -1 & \text{if } \langle v/d \rangle \in s \text{ but } \langle v/d \rangle \text{ is not a goal proposition} \\
    0 & \text{otherwise}
    \end{cases}
\]

**Non-negative Utility** For the non-negative utility assignment method, we compared our approach, referred to as the *improving* approach here, to the *collecting* approach (Domshlak and Mirkis 2015). In both cases we ran inc-compile-and-BFBB with the blind heuristic, using our modified version of it for the improving approach.

We begin by comparing the informativeness of the landmarks of our *improving* approach, relative to that of the landmarks of the *collecting* approach. Figures 2a-2d compare the number of expanded nodes for each problem using collecting and improving approaches. As these figures show, the number of nodes expanded by the *improving* approach was always lower than the number of nodes expanded by the *collecting* approach. A closer look shows that as the budget decreases, our approach becomes more effective.

Table 1 shows the total number of problems solved by each approach for each budget. These results show the *improving* approach is able to solve more problems. Furthermore, in some cases, it is able to prove that there is no way to improve upon the initial state with the given budget — this happens when the total cost of the value-landmarks discovered exceeds the budget. In this case, there is no need to perform search; the total number of problems for each domain where this occurs is in parentheses. Additionally, the table shows the total planning time for the set of commonly solved problems in each domain, where again the *improving* approach is faster on average.

**Negative Utility** For this utility assignment method, we could not compare to the *collecting* approach, since it is restricted to non-negative utilities. Thus, we only evaluated the *improving* approach using our modified version of inc-compile-and-BFBB.

Table 2 shows the results of the *improving* approach on these problems. While we have nothing to compare the results with, our planner does manage to optimally solve all but 1 of the tasks, and manages to solve quite a few of them with no search.

**Conclusion and Future Work**

We have addressed OSP with general additive utility functions. By defining the notions of net utility of an action, and of a gross positive action, we were able to exploit observations about the structure of optimal plans to derive value-landmarks. These landmarks can then be used in an OSP planner to obtain state-of-the-art performance.

These notions are not restricted to OSP planning. For example, the notion of the net utility of an action can be adapted to classical planning, where an action has net positive utility if it achieves some goal facts without deleting any other goal facts. We believe exploiting this observation in the context of classical planning can be beneficial.
Figure 2: Comparative view of the number of expanded nodes for the fact oriented collecting approach (x-axis) vs. the action oriented improving approach (y-axis) on tasks with different budgets relative to $c^*$. 

Table 1: Number of problems solved and (in brackets) solved with no search at all, across the different budgets using the improving approach and the collecting approach. 

Table 2: Performance of the utility setting independent OSP solver across the different budgets, on domains with negative utility setting, in terms of expanded nodes. The number of problems that were solved with no search is in brackets.
References


