uc3m | Universidad Carlos III de Madrid

e-Archivo

This is a postprint version of the following published document:

García-Olaya, A., de la Rosa, T., Borrajo, D. (2021). Selecting goals in oversubscription planning using relaxed plans. *Artificial Intelligence*, 291, 103414

DOI: 10.1016/j.artint.2020.103414

© Elsevier, 2021

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

Selecting goals in Oversubscription Planning using Relaxed Plans

Angel García-Olaya[a,*], Tomás de la Rosa[a], Daniel Borrajo[a]

[a] *Computer Science Department, Universidad Carlos III de Madrid. Leganés (Madrid) Spain*

Abstract

Planning deals with the task of finding an ordered set of actions that achieves some goals from an initial state. In many real-world applications it is unfeasible to find a plan achieving all goals due to limitations in the available resources. A common case consists of having a bound on a given cost measure that is less than the optimal cost needed to achieve all goals. Oversubscription planning (OSP) is the field of Automated Planning dealing with such kinds of problems. Usually, OSP generates plans that achieve only a subset of the goals set. In this paper we present a new technique to *a priori* select goals in no-hard-goals satisficing OSP by searching in the space of subsets of goals. A key property of the proposed approach is that it is planner-independent once the goals have been selected; it creates a new non-OSP problem that can be solved using off-the-shelf planners. Extensive experimental results show that the proposed approach outperforms state-of-the-art OSP techniques in several domains of the International Planning Competition.

Keywords: Automated planning, Oversubscription planning, Satisficing planning, Partial satisfaction planning

1. Introduction

The objective of *classical* Automated Planning is to find a sequence of actions transforming an initial state into a state where a set of goals is true. A valid plan is the sequence of actions reaching a state where all goals have been achieved. If any of the goals remains unachieved, the plan will not be valid. But in many real-world applications finding such a plan is unfeasible as there are more goals than those which can be achieved with the available resources. Thus, finding a plan that achieves all goals is not the objective of the problem solver. Instead, it should find a plan that achieves the maximum number of them, or more commonly, that maximizes some kind of utility, computed as a function of the reached goals.

The problem of planning with insufficient resources is tackled by Oversubscription planning (OSP) [1], which was initially motivated by some real tasks at NASA. In its original formulation a limited consumable resource, the battery of a rover, makes reaching all goals impossible; there are more goals than those that can be achieved given its stored energy. Most papers in the literature generalize the problem by representing the insufficient resource as a cost-bound of the plan [1, 2, 3, 4, 5]. Only plans with cost equal or lower than the bound will be valid, with the

*Corresponding author

Email addresses: agolaya@inf.uc3m.es (Angel García-Olaya), trosa@inf.uc3m.es (Tomás de la Rosa), dborrajo@ia.uc3m.es (Daniel Borrajo)

assumption that no plan will exist reaching all goals with such cost bound. Examples of such limited and insufficient resources appear in space exploration [1, 6], transportation [7] and in many other applications.

OSP appears in many real applications but surprisingly it has not drawn too much attention from the planning community after the seminal work of Smith [1]. Only recently some works have addressed this task, dealing with its optimal version [2, 3, 4, 5]. But the suboptimal OSP task remains mostly unexplored even if solving OSP tasks optimally is not feasible in many real cases. Currently, non-optimally solving OSP problems requires either creating entirely new planning systems or heavily modifying the code of the existing optimal ones.

This paper introduces a sub-optimal algorithm to solve OSP problems, with the key advantage of being planner independent. In a first step, relaxed plans [8] are used to compute *distances* among goals. Distances are an estimation of the cost needed to achieve a goal from a state where another goal has been achieved. A subsequent search in the space of goals' subsets uses those distances to find subsets of goals of increasing utility with estimated costs under the cost bound. Then, a suboptimal planner takes as input each new selected goals' set and tries to find a plan. The approach is planner independent once goals are selected: any planner supporting Planning Domain Definition Language (PDDL), version 2.1 [9] numeric preconditions or any cost-bounded planner can be used to find the plan. This feature makes our approach particularly attractive to solve oversubscription planning tasks off-the-shelf. It allows users to solve suboptimal OSP tasks without creating a new planning system or digging into planning code.

This paper extends our previous work [10] with the addition of an iterative improvement behavior; if a plan is found for the selected goals, it keeps searching for candidate goal sets that may improve the already achieved utility until the allotted time is exhausted. Also, a formal analysis of the algorithm has been included, as well as a more comprehensive evaluation, comparing it in more domains to alternative approaches found in the literature.

OSP is formally defined in the following section. Related work is described in Section 3. Section 4 presents the algorithm we use to solve the problem. Section 6 shows the experimental results and the comparison of our technique with other approaches. These approaches are previously presented in Section 5, which contains also a description of the domains and problems used for evaluation. Finally, Section 7 draws some conclusions and hints at future work.

2. Background

In this section, we formally define the OSP problem and show a way to encode it using the standard language for the description of planning tasks, PDDL [11, 12].

Definition 1 (STRIPS planning task with actions costs)**.** *A* STRIPS *planning task with actions costs is a tuple* $P = \{F, A, I, G, c\}$*, where*

- F *is a finite set of fluents,*
- A *is a finite set of instantiated actions, mapping one state into another one,*
- $I \subseteq F$ *is the initial state,*
- $G \subseteq F$ *is the set of goals, and*
- $c : A \mapsto \mathbb{R}_0^+$ *is a cost function.*

An action $a \in A$ is a tuple $\{pre(a), add(a), del(a)\}$, such that $pre(a)$, $add(a)$, $del(a) \subseteq F$. An action sequence $\pi = (a_1, a_2, ...a_n)$ is applicable in P if there exists a sequence of states $(s_0, s_1, ...s_n)$, such as $s_0 = I$, and $\forall i = 1...n$, $pre(a_i) \subseteq s_{i-1}$ and $s_i = (s_{i-1} \setminus del(a_i)) \cup add(a_i)$. An action sequence π is a plan for P if it is applicable in P and $G \subseteq s_n$.

Definition 2 (Cost of a plan)**.** *The cost of a plan π, solution to the planning task P, can be defined as: $C(\pi) = \sum_{a_i \in \pi} c(a_i)$.*

Definition 3 (STRIPS planning task with actions costs and utilities)**.** *A* STRIPS *planning task with actions costs and utilities is a tuple $P_u = \{F, A, I, G, c, u\}$, where $\{F, A, I, G, c\}$ is a* STRIPS *planning task with actions costs as defined above and $u : 2^F \mapsto \mathbb{R}_0^+$ is a utility function.*

The utility function is usually defined over individual facts $f \in F$ ($u : F \mapsto \mathbb{R}_0^+$). Facts $f \notin G$ for which $u(f) > 0$ are known as soft-goals, in contrast with facts in G, which are called hard-goals and most commonly are assigned zero utility. Facts that are not soft-goals or hard-goals are also assigned a zero utility. The definitions of plan and cost of a plan remain the same as in Definitions 1 and 2.

Definition 4 (Utility of a plan)**.** *The utility of a plan π that solves a* STRIPS *planning task with actions costs and utilities is $U(\pi) = \sum_{f_i \in s_n} u(f_i)$, where s_n is the state resulting of applying π to I.*

We consider additive utilities, as most work in the field does (see [13] for other types of utilities). An alternative representation of utilities consists of defining penalties to be minimized. A penalty is a function $p : F \mapsto \mathbb{R}_0^+$ and the penalty of a plan is $penalty(\pi) = \sum_{f_i \notin s_n} p(f_i)$. Again, facts that are not soft-goals are assigned a zero penalty, while facts in the hard-goals set have an implicit penalty of ∞.

Definition 5 (Oversubscription planning task)**.** *An oversubscription planning (OSP) task is a tuple $P_{OSP} = \{F, A, I, G, c, u, c_b\}$, where $\{F, A, I, G, c, u\}$ is a* STRIPS *planning task with actions costs and utilities as defined above and $c_b \in \mathbb{R}^+$ is a cost-bound.*

A sequence of actions $\pi = (a_1, a_2, ...a_n)$, is a plan for P_{OSP} if it is a plan for $\{F, A, I, G, c, u\}$ and $C(\pi) \leq c_b$. Most approaches in the literature consider $G = \emptyset$ for OSP tasks. This means there are no hard-goals; only soft-goals. Therefore, any plan under the cost-bound, even the empty one, is a valid plan. In such setting, the objective in OSP is to find a plan that maximizes the utility. In this paper we will assume there are no hard-goals, and we will use indistinctly the terms *goal* and *soft-goal*. Hence, we will define our task as $P_{OSP} = \{F, A, I, c, u, c_b\}$.

PDDL3 [12] allows us to define OSP tasks using *preferences* over the goal states. Figure 1 shows an example of the PDDL3 parametrized definition of the `navigate` action in the rovers OSP domain. The action has three parameters, the `rover`, the `origin` and the `destination`. The `total-cost` function accounts for the current cost of the plan, while `traverse-cost` represents the cost of the action. We have also defined `cost-bound` to constraint the maximum cost of any solution. The preconditions check that the rover is available, it is at the origin, it can move from the origin to the destination, and the cost after applying the action will not exceed the cost-bound[1]. Once the action is applied, the rover is no longer at the origin, it is at the destination and the total cost is increased conveniently.

[1]Notice that to formulate an OSP task using PDDL3, numeric preconditions must be added to all actions increasing the `total-cost`. An alternative encoding that avoids such numeric preconditions, but which is not standard PDDL yet, can be found at [4].

```
(:action navigate
 :parameters (?r - rover ?o - waypoint ?d - waypoint)
 :precondition (and (available ?r)
                    (at ?r ?o)
                    (can_traverse ?r ?o ?d)
                    (<= (+ (total-cost) (traverse-cost ?r ?o ?d))
                        (cost-bound)))
   :effect (and (not (at ?r ?o))
                (at ?r ?d)
                (increase (total-cost) (traverse-cost ?r ?o ?d))))
```

Figure 1: Example of a cost-bounded action in the Rovers domain.

Figure 2 shows an example of a problem defined for the rovers domain. It has three locations, one rover, equipped with a camera and placed at the second location, and a cost-bound of 20. The goal is to take an image of a couple of objectives which are only visible from the first and third locations, respectively. Instead of defining utilities, PDDL3 uses penalties. The `(preference ⟨preference tag⟩ ⟨literal⟩)` construct is used to define which literals are assigned a positive penalty. The specific penalties are defined in the metric by using `(is-violated ⟨preference tag⟩)`. In this example, if the robot cannot take the image of the first objective, it would pay a penalty of three, while for the second objective the penalty is two. The aim is to find a plan minimizing the total penalty. The example problem does not have hard-goals as it is usual in OSP.

```
(define (problem roverExample)
        (:domain rover)
        (:objects rover1 - rover
                  waypoint1 waypoint2 waypoint3 - waypoint
                  camera1 - camera
                  objective1 objective2 - objective)
        (:init (= (cost-bound) 20)
               (= (total-cost) 0)
               (at rover1 waypoint2)
               (available rover1)
               (can_traverse rover1 waypoint1 waypoint2)
               (can_traverse rover1 waypoint2 waypoint1)
               (can_traverse rover1 waypoint2 waypoint3)
               (can_traverse rover1 waypoint3 waypoint2)
               (on_board camera1 rover1)
               (visible_from objective1 waypoint1)
               (visible_from objective2 waypoint3)
               (= (traverse-cost rover1 waypoint1 waypoint2) 7)
               (= (traverse-cost rover1 waypoint2 waypoint1) 8)
               (= (traverse-cost rover1 waypoint2 waypoint3) 9)
               (= (traverse-cost rover1 waypoint3 waypoint2) 7))
        (:goal (and (preference p1 (taken_image objective1))
                    (preference p2 (taken_image objective2))))
        (:metric minimize (+ (* (is-violated p1) 3) (* (is-violated p2) 2))))
```

Figure 2: Example of an OSP problem in the Rovers domain.

3. Related work

A recent work established the different nature of OSP with respect to its two closest related planning tasks [2]: COST-BOUNDED and NET-BENEFIT planning. Planning with limited consumable resources, also known as Resource Constrained Planning or COST-BOUNDED planning, has recently

drawn some attention of the planning community [14, 15, 16]. It also establishes a cost-bound on the plan, but unlike OSP it assumes the bound is big enough to achieve all the goals. Also, it does not include any utility function; it only considers hard-goals and no soft-goals.

NET-BENEFIT planning is also quite close to OSP. In fact, both tasks are subtasks of Partial Satisfaction Planning (PSP) [17, 18], and have been often confused in the literature [2]. OSP and NET-BENEFIT define utilities over some facts, but there is no cost-bound in NET-BENEFIT. Also, the objective does not consist of maximizing $U(\pi)$, but some combination of utility and cost, usually $U(\pi) - C(\pi)$. Thus, it establishes a trade-off between the utility of achieving a goal and the cost of doing it [19]. Finding a plan with a NET-BENEFIT bigger than or equal to a value k is PSPACE-complete [18]. Optimal OSP is also PSPACE-complete [3]. Nevertheless, a study shows that, in the most common case of additive utilities, optimal NET-BENEFIT lies between polynomial and NP-complete, depending on the fragment considered. Fragments of both problems are characterized by their causal graph structure and variable domain sizes. The same fragments for OSP are always NP-complete [3]. NET-BENEFIT was included in the International Planning Competition (IPC)[2] in 2006 (IPC-2006) under the PREFERENCES track and in IPC-2008 under the NET-BENEFIT track. In the PREFERENCES track, problems can include not only soft-goals but also preferences on preconditions of actions. Benton [19] shows a way to transform PREFERENCES problems into NET-BENEFIT ones. Probably due to the IPC, most work on PSP has been on NET-BENEFIT [13, 19, 18, 20, 21, 22, 23]. In contrast, OSP has been less explored [1, 2, 3, 4, 5], even though the existence of a limited resource (time, fuel, battery, storage space, money, ...) is present in a large number of real domains. Given that many techniques used for the NET-BENEFIT task can also be applied to OSP, this section will summarize related work addressing any of the two techniques.

The optimal solution for a PSP problem can be computed by finding optimal plans for each of the 2^n combinations of the n goals of the problem, and then selecting the plan with maximum utility or net-benefit. Of course, this is intractable except for very simple problems. In practice, three different approaches can be found in the literature: *a priori* selection of goals, anytime search taking into account all goals, and compilation into a different problem.

Goals can be selected *a priori* to find the potentially best subset to plan for. In a first step, goals are selected using some criteria. Then, a planning step is performed taking into account only the selected goals. The seminal OSP work by Smith selects goals by creating an orienteering problem (OP), a variation of the traveling salesperson problem [1]. The OP is constructed in a process that involves creating an abstract state space by selecting a subset of the problem propositions and using relaxed plans to calculate the costs of achieving the goals in that abstract space. The resulting OP is solved using beam search. The solution is an ordered set of promising goals which are supplied one at a time to a partial-order planner.

Relaxed plans have been also used to select goals in NET-BENEFIT [23]. For each goal, the approach computes the highest net-benefit set containing it. The net-benefit of each individual goal is estimated using the cost of the relaxed plan solving a simplified problem where all the remaining goals have been removed. The same procedure is used to compute the net-benefit of all two-goals sets including that goal and every other one, keeping the set with the highest net-benefit, which is used to construct three-goals sets. This process is repeated until it finds a set that does not improve its net-benefit when adding a new goal. After performing this process for every goal, the set with highest net-benefit is selected for planning. We also use relaxed plans to select goals. In Section 4.3 we compare our approach against these two approaches, including an analysis of their complexity.

[2] http://www.icaps-conference.org/index.php/Main/Competitions

An alternative to goal selection is to take into account all goals when planning. For example, the planner can be instructed to incrementally find higher utility plans, or use previously found plans as a bound to prune nodes during search. In general, those approaches scale worse than the goal-selection ones as they need to reason taking into account the whole set of soft-goals instead of a subset of them, which results in bigger search spaces. In OSP, admissible heuristics based on landmarks for goal reachability and abstractions [24] have been used to guide the search using a best-first branch-and-bound algorithm [2]. The best plan found so far is stored and branches that do not increase the utility are pruned. More informative landmarks based on properties of OSP optimal plans have been proposed [5], although its performance is not far from a blind search approach. OPTIC [25] is a heuristic planner that combines a classical relaxed plan-graph heuristic for standard propositions, h_{FF} [8], and mixed integer programming for the soft-goals. It uses automata to reason about accomplishment of each soft-goal (unsatisfied, satisfied, or impossible to be satisfied from the current state). HPLAN-P [26] allows planning for temporally extended preferences, which are internally compiled to soft goals using automata. It also includes a variety of heuristic functions in combination with an incremental best-first planning algorithm which uses iteratively each of the heuristics to find a better plan. In domains where all actions have the same cost, one of the heuristics they propose, the *Preference distance function*, has some similarities with our approach. But they use the heuristic to guide the search while we use it to select goals. Also, they compute the sum of distances for all goals, while we compute a value for each goal.

A hybrid approach is presented in $\mathcal{S}apa^{\mathcal{PS}}$ [19]. It combines an anytime A^* algorithm with a heuristic that for each node estimates the set of goals with highest net-benefit and the plan that achieves them with the lowest cost. The search is done in the plans space. Do *et al.* [13] extend the NET-BENEFIT problem to the case of non-additive heuristics and introduce a framework to classify goal utilities and cost dependencies to account for different situations; e.g. when the utility of a goal depends on the achievement of another goal.

A common way to solve PSP problems is to transform them either into (simpler) planning problems or into another formulation. A PSP problem can be modeled as a Markov Decision Process [1, 18], obtaining a policy from which a plan finding the optimal solution can be extracted. However, this conversion does not scale well and it has not been reported to be used in practice. Using integer programming to find optimal plans for a given parallel plan length is another possible transformation [18]. Both GAMER [21] and MIPS-XXL [20] transform the NET-BENEFIT task into a STRIPS task with action costs, by adding some numeric literals whose values change if a soft-goal is not achieved.

The most successful transformation for the NET-BENEFIT task creates also a STRIPS with action costs task, with more actions, more fluents and more goals, but no soft-goals [27]. Given that the compiled version only has hard goals, any classical planner can be used to solve this new problem. For every soft-goal a new goal and two new actions are added to the domain. Both actions achieve the new goal. One action has the soft-goal as precondition and the action cost is zero. The second action does not have such precondition, and it has a cost equal to the soft-goal's penalty. Thus, the planner can decide whether to reach the new goal using the first or the second action, paying the penalty in the second case. This compilation, with some improvements and optimizations, in combination with LAMA [28], the winner of the satisficing track of the IPC-2008, outperforms any participant of the NET-BENEFIT or PREFERENCES tracks of the IPCs 2006 and 2008. This compilation can be easily adapted to reduce OSP to classical planning with the real-valued state variables supported by standard PDDL2.1 [2]. Figure 3 shows a simplified example of

the two dummy actions that need to be added to the rovers domain to handle the `taken-image` soft-goals defined in the problem of Figure 2.

A recent work presents two compilations transforming the OSP problem into a multiple cost function (MCF) planning task [4], where utilities form the primary cost function and the cost-bound the secondary one. Their first reformulation, *soft-goals*, is a straightforward adaptation of Keyder and Geffner's one [27]. The second transformation, *state-delta*, is based on the net utility of each operator in the domain. Using these compilations, they present two merge and shrink [29] admissible heuristics to decide which nodes to prune in optimal planning. Another work shows how these compilations allow the problem to be solved using A* search with bound-sensitive heuristics [30].

```
(:action soft-goal-collect
 :parameters (?o - objective ?s - softgoal)
 :precondition (and (at-softgoal ?o ?s)
                    (taken-image ?o))
 :effect (achieved-softgoal ?s))

(:action soft-goal-forgo
 :parameters (?o - objective ?s - softgoal)
 :precondition (and (at-softgoal ?o ?s)
                    (not (taken-image ?o)))
 :effect (and (achieved-softgoal ?s)
              (increase (total-penalty) (penalty ?s))))
```

Figure 3: Example of two inserted actions in a compiled OSP domain following Keyder and Geffner's transformation.

4. Algorithm

In this section, we describe *Relaxed Plan Linear Distances* (RPLD), the proposed algorithm, which performs an *a priori* selection of goals (Algorithm 1). First, *distances* between goals are computed based on a relaxed plan. These *distances* are estimates of costs of achieving each goal from the initial state, and costs of reaching a goal once any other goal has been achieved. Those *distances* are used to search for sets of goals that maximize the expected utility while keeping the estimated cost of reaching them under the cost-bound. Once an ideally non-oversubscribed set of goals is found, an external non-OSP planner is invoked to find a plan for it. Search continues returning sets of increasingly better utility and stops when the timeout is reached or no set improving the utility is found.

The inputs of the algorithm are an OSP task and the time the external planner will be given to find a plan for the selected goals' subset. Lines (1-3) initialize the maximum utility found so far (u_{max}) to the utility of the empty plan, and the selected goals set (G') and the set of non-achievable goals sets (D) to empty. D stores discarded sets of goals for which no plan has been found and is used to prune branches when selecting goals. Next, we compute the distances between goals (line 4) and invoke an anytime procedure SELECT-GOALS-AND-PLAN. This procedure performs a Depth First Branch and Bound (DFBnB) search over the sets of goals G' that seem to be achievable. Then, it tries to find plans for selected nodes. If this procedure exhausts the search space and there is still time, a second DFBnB process, ENFORCED-SELECT-GOALS-AND-PLAN, extends the search over the sets that are considered non-achievable according to the distances. If this second procedure finds a plan, it is returned. Otherwise, the plan found by the first procedure is returned. DFBnB is used as a greedy approach for goals' selection. It tends to find sets with

Algorithm 1 Relaxed Plan Linear Distances (RPLD)

Input: $P_{OSP} = \{F, A, I, c, u, c_b\}$: OSP task, t: time bound for the external planner
Output: Plan: π

$u_{max} \leftarrow U(\emptyset)$ ▷ Initialized to the utility of the empty plan
$G' \leftarrow \emptyset$ ▷ Selected goals set
$D \leftarrow \emptyset$ ▷ Used to prune the search while selecting goals
$\Delta \leftarrow$ CALCULATE-DISTANCES(P_{OSP})
$\pi \leftarrow$ SELECT-GOALS-AND-PLAN$(P_{OSP}, G', \Delta, D, u_{max}, t)$
$G' \leftarrow \emptyset$ ▷ Starting selection again
$\pi' \leftarrow$ ENFORCED-SELECT-GOALS-AND-PLAN$(P_{OSP}, G', \Delta, D, u_{max}, t)$
if $\pi' \neq \emptyset$ **then**
    return π'
else
    return π

a smaller number of goals than a more exhaustive search would do. Given that relaxed plans tend to underestimate the cost of reaching a goal in general, a smaller set reduces the chances of oversubscription. We confirmed this hypothesis with previous experiments, where less greedy algorithms produced often oversubscribed sets.

In the next subsections we will detail the distances calculation (CALCULATE-DISTANCES, Section 4.1), and the goal selection and planning procedures (SELECT-GOALS-AND-PLAN and ENFORCED-SELECT-GOALS-AND-PLAN, Section 4.2). We also analyze the formal properties of our algorithm and compare them to related work (Section 4.3).

4.1. Distances between goals

In the first step, a matrix Δ of *distances* among goals is computed. It is an $n \times (n+1)$ matrix, where $n = |\mathbb{G}|$ and $\mathbb{G} = \{f \in F \mid u(f) > 0\}$ are the soft-goals. $\Delta(x, 0)$, $1 \leq x \leq n$, also denoted Δ_{Ix}, contains the cost estimate for achieving goal g_x from the initial state. $\Delta(x, y)$, $1 \leq x, y \leq n$, also denoted Δ_{xy}, contains the cost estimate to achieve goal g_y once goal g_x has been reached.

Estimates are calculated using relaxed planning tasks. Any STRIPS-based planning task can be relaxed by changing the set of actions A by A'. Each $a'_i = \{pre(a_i), add(a_i), \emptyset\}, a'_i \in A', a_i \in A$. In other words, the delete effects of actions are removed. These new tasks are simpler to solve than the original ones and are commonly used to create heuristics for the original task. One such heuristic is h_{FF}, whose value corresponds to the cost of a plan for the relaxed task [8]. We use the value of h_{FF} to calculate distances in the following way:

Definition 6 (Distance to a goal)**.** *Let $P_{OSP} = \{F, A, I, c, u, c_b\}$ be an OSP planning task, $g_x \in F, u(g_x) > 0$, a goal and $P_r(g_x) = \{F, A', I, \{g_x\}, c\}$ be a relaxed* STRIPS *planning task with actions costs with a single goal. Let $\pi_r(g_x)$ be a relaxed plan that solves $P_r(g_x)$ using h_{FF} [8]. We define the distance from I to goal g_x, Δ_{Ix}, as $C(\pi_r(g_x))$.*

If a relaxed plan is not found while calculating a given distance, its value is set to infinity. If $C(\pi_r(g_x)) = 0$, which means the goal is already achieved in the initial state, Δ_{Ix} is set to infinity. The reason is that we assume the goal has to be undone to be reached again in the future and the zero distance is a misleading value.

$\pi_r(g_x)$ is a solution for the relaxed one-goal task, but it will usually not be a solution for the original one-goal task. Actually, very often it is not even applicable. However, Δ_{Ix} helps deciding

whether or not to include this goal into the set of goals to plan for. Its value, combined with the utility of each goal, could be enough to decide which goals to select if the cost of achieving a goal were independent of whether another goal had been reached before. But, in most cases, achieving a goal will change the cost of reaching other goals [1]. To estimate such dependency among goals we define the distance between two goals as follows:[3]

Definition 7 (Distance between two goals)**.** *Let $P_{OSP} = \{F, A, I, c, u, c_b\}$ be an OSP task, $\pi_r(g_x) = (a'_1, a'_2, ...a'_n)$ be the relaxed plan used to calculate Δ_{Ix} and $\pi(g_x)$ be a plan, where each a' in $\pi_r(g_x)$ has been substituted by its corresponding a. Let s_x be the state resulting of applying $\pi(g_x)$ to I, ignoring preconditions. A new task $P_r(g_x, g_y) = \left\{F, A', s_x, \{g_y\}, c\right\}$ can be created. The distance between goals g_x and g_y, named Δ_{xy}, is equal to the cost of a relaxed plan $\pi_r(g_x, g_y)$ that solves $P_r(g_x, g_y)$; $\Delta_{xy} = C(\pi_r(g_x, g_y))$.*

In general, $\Delta_{xy} \neq \Delta_{yx}$ as the cost of achieving g_y from g_x is usually different than the cost of achieving g_x from g_y. Depending on the domain the estimations can be far away from the real values. However, other alternatives to obtain better estimations of the cost, like solving equivalent one-goal non-relaxed tasks, are not useful in practice; computing them takes too long in general as our previous work showed [10].

This method of computing the distance between two goals does not fully consider all the interactions between them. There could be other alternatives to compute the distances as using other heuristic or complete methods. We performed proof-of-concept experiments using other alternative heuristic definitions for distances, like defining Δ_{xy} as the cost of a relaxed plan including both g_x and g_y or applying A' instead of A to generate s_x. Those approaches select sets of goals that are still oversubscribed more often than when using our distances approach. Hence, we decided to use the defined distance metric.

Definition 8 (Distance to an ordered list of goals)**.** *Let $P_{OSP} = \{F, A, I, c, u, c_b\}$ be an OSP planning task, and $G' = (g_1, g_2, ...g_k)$, $g_i \in F, 1 \leq i \leq k$, $u(g_i) > 0$, be an ordered list of goals. We define the distance from I to G' as the estimated cost of achieving those goals from I in order. It is computed as $\Delta(G') = \Delta_{I1} + \Delta_{1\,2} + ... + \Delta_{(k-1)\,k}$.*

The computation of Δ is summarized in Algorithm 2. Lines (1-3) create new sets of relaxed actions, no-precondition actions, and soft-goals. For each goal in this set we create a single-goal non-cost-bounded relaxed problem and solve it, storing the cost of the found plan as Δ_{Ix} (lines 5-7). Then, the actions in the relaxed plan are replaced by their no-precondition equivalent ones and the plan is applied to reach a new state (lines 8-9). Starting from this new initial state, single-goal non-cost-bounded relaxed problems are created and solved for each one of the remaining goals. The costs of those plans are stored as Δ_{xy}. Finally, the algorithm returns the distances matrix Δ.

4.2. Goals selection

The distances matrix is used to select goals as shown in Algorithms 3, 4 and 5. SELECT-GOALS-AND-PLAN (Algorithm 3) implements a DFBnB search that adds goals based on their utility and estimated cost. In line 2, the procedure GENERATE-SUCCESSORS creates a set of candidate goals S

[3]For simplicity, we are going to assume that a relaxed plan is a linear sequence of actions.

Algorithm 2 CALCULATE-DISTANCES

Input: OSP task: $P_{OSP} = \{F, A, I, c, u, c_b\}$
Output: Distances matrix: Δ

1: $A' \leftarrow \{\{pre(a), add(a), \emptyset\} \mid a \in A\}$ ▷ Create new sets of actions
2: $A'' \leftarrow \{\{\emptyset, add(a), del(a)\} \mid a \in A\}$
3: $\mathbb{G} \leftarrow \{f \in F \mid u(f) > 0\}$
4: **for all** $g_x \in \mathbb{G}$ **do**
5: $P_r(g_x) \leftarrow \{F, A', I, \{g_x\}, c\}$ ▷ Create one-goal relaxed tasks
6: $\pi_r(g_x) \leftarrow$ CREATE-RELAXED-PLAN$(P_r(g_x))$ ▷ Solve relaxed tasks
7: $\Delta(x, 0) \leftarrow C(\pi_r(g_x))$ ▷ Assign the cost of the relaxed plan to Δ_{Ix}
8: $\pi(g_x) \leftarrow$ REPLACE$(\pi_r(g_x), A', A'')$ ▷ Replace each $a \in A'$ by its equivalent $a \in A''$
9: $s_x \leftarrow$ APPLY$(\pi(g_x), I)$ ▷ Apply the relaxed plan
10: **for all** $g_y \in \mathbb{G}$ **do**
11: $P_r(g_x, g_y) \leftarrow \left\{F, A', s_x, \{g_y\}, c\right\}$
12: $\pi_r(g_x, g_y) \leftarrow$ CREATE-RELAXED-PLAN$(P_r(g_x, g_y))$
13: $\Delta(x, y) \leftarrow C(\pi_r(g_x, g_y))$ ▷ Assign the cost of the relaxed plan to Δ_{xy}
14: **return** Δ

to extend the input set $G' = \{g_1, g_2, \ldots g_k\}$. We first define S' to be the set of goals that can be added to G' maintaining the total cost under the cost bound.

$$S' = \{g_i \in (\mathbb{G} \setminus G') \mid (\Delta(G') + \Delta_{ki}) \leq c_b\} \tag{1}$$

Then S can be computed as:

$$S = \{g_i \in S' \mid \nexists d \in D, d \subseteq (G' \cup \{g_i\})\} \tag{2}$$

In other words, S contains all the goals that can be added to G', maintaining the total cost under the cost-bound and that do no result in supersets of already non-achieved sets (sets for which the procedure INVOKE-PLANNER did not find a plan in previous iterations). If there are no successors, G' is a terminal node and cannot be further expanded. In that case, if its utility is greater than the maximum utility found so far, we try to find a plan for it by calling INVOKE-PLANNER (lines 3-6), and return its result (the plan found or the empty plan). In any other case, this branch of the search is a dead-end and the algorithm returns the empty plan. In non-terminal nodes, it selects the successor with lowest cost from those with highest utility (lines 8-9), and recursively invokes SELECT-GOALS-AND-PLAN (line 10). If the recursive call finds a plan improving the utility value, it is stored (lines 11-12) and the algorithm expands the next sibling. If there are no more successors, the procedure returns π (lines 17-18). If no plan were found for any of its successors and $u(G') > u_{max}$, G' would become a terminal node and would be planned for. However, we do not wait to expand all the successors to plan for G'. If no plan is found for the first successor, the algorithm tries to find a plan for G'. If no plan is found for it, the algorithm returns the empty plan (lines 13-16). If a plan is found, it keeps expanding successors. This procedure is equivalent to removing the last goal from the selected set until a plan is found. It has two practical advantages. First, it allows for an early pruning; a regular DFBnB algorithm would try first all the terminal nodes. Given that the distances tend to underestimate the real cost, the OSP tasks in those terminal nodes are often still oversubscribed. Checking a set G' as soon as any of its supersets seems to be oversubscribed allows the algorithm to test if this branch deserves to be explored. Second, in general, the smaller the set is, the easier it is that it is not

oversubscribed. Thus, also the easier it is to find a plan for it. If no plan is found for a terminal node n, instead of keep expanding other terminal nodes, the algorithm tries to find a plan for nodes with $u > u_{max}$ in the path to n, thus having a first plan even if of small utility.

Algorithm 3 SELECT-GOALS-AND-PLAN

Input: $P_{OSP} = \{F, A, I, c, u, c_b\}, G', \Delta, D, u_{max}, t$
Output: π: plan found or $\emptyset$ if no plan rooted at G' improves u_{max}

```
1: π ← ∅
2: S ← GENERATE-SUCCESSORS(G', Δ, D, c_b, u)        ▷ Successors of the current set
3: if S = ∅ then                                    ▷ Terminal node, we check its utility
4:     if u(G') > u_max then
5:         π ← INVOKE-PLANNER({F, A, I, G', c, c_b}, t, u_max, D)
6:     return π
7: repeat                                           ▷ If not terminal, we expand it
8:     g_x ← SELECT(G', S, Δ, u)                    ▷ The child with highest utility
9:     S ← S \ {g_x}                                ▷ Remove it from the set of children
10:    p ← SELECT-GOALS-AND-PLAN(P_OSP, G' ∪ {g_x}, Δ, D, u_max, t)   ▷ Expand it
11:    if p ≠ ∅ then
12:        π ← p                                    ▷ Updating the best plan found so far
13:    else if u(G') > u_max then
14:        π ← INVOKE-PLANNER({F, A, I, G', c, c_b}, t, u_max, D)
15:        if π = ∅ then
16:            return π                             ▷ If no plan for G', it is not expanded anymore
17: until S = ∅
18: return π
```

INVOKE-PLANNER calls an external planner to find a plan for a new COST-BOUNDED planning task $P_{cb} = \{F, A, I, G', c, c_b\}$. G' can still be oversubscribed, so we run the planner with a time bound. There is a trade-off between devoting more time to a still potentially oversubscribed set of goals or backtracking. Backtracking will increase the chances of finding a solution, but this solution will have less utility. Different strategies to set this time are discussed in Section 5.4. If a plan is found, the procedure updates the maximum utility, and returns the solution. Otherwise, it adds this set to the non-achievable ones to prune its supersets in the search, and returns an empty plan. In most cases we cannot prove there is no plan for the set as very often t does not allow to exhaust the search space.

Algorithm 4 INVOKE-PLANNER

Input: Cost-bounded problem $P = \{F, A, I, G', c, c_b\}, t, u_{max}, D$
Output: π: plan found or $\emptyset$ if no plan is found

```
1: π ← PLAN({F, A, I, G', c, c_b}, t)
2: if π ≠ ∅ then
3:     u_max ← U(π)                                 ▷ Updating the best utility
4: else
5:     D ← D ∪ {G'}                                 ▷ Supersets of G' will be pruned
6: return π
```

When SELECT-GOALS-AND-PLAN has tried all the combinations of goals under the cost-bound, the bound is relaxed and ENFORCED-SELECT-GOALS-AND-PLAN is invoked. It is also a DFBnB search,

but instead of trying to find a plan for the terminal nodes only, it will try to find a plan for every expanded node with utility higher than the maximum one. The set of successors will be:

$$S = \{g_i \in (\mathbb{G} \setminus G') \mid \nexists d \in D, d \subseteq (G' \cup \{g_i\})\} \tag{3}$$

In other words, it is composed of all the non-pruned sets of goals resulting from adding a goal to the current set, despite their estimated cost. It first tries to find a plan for G' (lines 2-5). If a plan is found, it is expanded (lines 6-12). Finally, the best plan found either for G' or one of its successors is returned.

Algorithm 5 ENFORCED-SELECT-GOALS-AND-PLAN

Input: $P_{OSP} = \{F, A, I, c, u, c_b\}, G', \Delta, D, u_{max}, t$
Output: π: plan found or $\emptyset$ if no plan rooted at G' improves u_{max}

1: $\pi \leftarrow \emptyset$
2: **if** $u(G') > u_{max}$ **then**
3: $\quad \pi \leftarrow$ INVOKE-PLANNER$(\{F, A, I, G', c, c_b\}, t, u_{max}, D)$
4: $\quad$ **if** $\pi = \emptyset$ **then**
5: $\quad\quad$ **return** π ▷ This branch will not be expanded
6: $S \leftarrow$ GENERATE-SUCCESSORS(G', D, u) ▷ Successors of the current set
7: **while** $S \neq \emptyset$ **do** ▷ If not terminal, we expand it
8: $\quad g_x \leftarrow$ SELECT(G', S, Δ, u) ▷ The child with highest utility
9: $\quad S \leftarrow S \setminus \{g_x\}$ ▷ Remove it from the set of children
10: $\quad p \leftarrow$ ENFORCED-SELECT-GOALS-AND-PLAN$(P_{OSP}, G' \cup \{g_x\}, \Delta, D, u_{max}, t)$
11: $\quad$ **if** $p \neq \emptyset$ **then**
12: $\quad\quad \pi \leftarrow p$ ▷ Updating best plan found
13: **return** π

Figures 4a to 4e show the algorithm execution for a four-goals problem, none of them holding in the initial state (initially $u_{max} = 0$). All have unitary utility $u(g_i) = 1, \forall g_i \in \mathbb{G}$, so in this case selection will be performed only attending to distance. The cost-bound is 20, and the distances matrix is shown on the right of Figure 4a. SELECT-GOALS-AND-PLAN will add goals to G' until it reaches the terminal node $G' = \{g_1, g_2, g_3, g_4\}$ (Figure 4a). Let us assume that no plan is found for them in the allotted time. Then, $\{g_1, g_2, g_3, g_4\}$ is added to D to prune its supersets (none in this example). We prune all the supersets of these goals as the order in G' is relevant during the selection process, but not for planning, i.e. we assume that if INVOKE-PLANNER did not find a plan for $\{g_1, g_2, g_3, g_4\}$, no plan will be found for its supersets, if there were any.[4] As no plan is returned and $u(\{g_1, g_2, g_3\}) > u_{max}$, the planner is invoked with $\{g_1, g_2, g_3\}$. If no plan is found, the algorithm backtracks again: $\{g_1, g_2, g_3\}$ is marked as unachievable, and it tries to find a plan for the remaining two goals. If a plan is found for $G' = \{g_1, g_2\}$, backtracking finishes and $\{g_1, g_2\}$ is expanded. $\{g_1, g_2, g_4\}$ is a terminal node and improves utility, so the planner is invoked again (Figure 4b). If no plan is found for it, it will be added to D and the algorithm will backtrack. In this case, it will not try to find plans for the parent node as its utility is not higher than the maximum found ($u_{max} = 2$). The next terminal node with a higher utility is $\{g_1, g_3, g_4\}$ (Figure 4c). If no plan is found again, the branch rooted on g_1 will be completely explored and

[4] As said, this is only true if the external planner is given enough time to find a plan or prove that no plan exists. In any other case it may happen that no plan is found for the set, but adding a new goal guides the planner in such a way that a plan is found for the superset. In our experiments the external planner is usually not able to prove no plan exists.

SELECT-GOALS-AND-PLAN will continue exploring the remaining branches where no set improving the utility exists (Figure 4d). Once SELECT-GOALS-AND-PLAN finishes, ENFORCED-SELECT-GOALS-AND-PLAN will be invoked. The first node improving utility is $\{g_2, g_3, g_4\}$ (Figure 4e). If no plan is found, the search continues, but all the nodes with higher utility ($u > 2$) are already pruned, so the algorithm finishes.

Distances are used to prune successors (Equation 1) but play a minor role in search; they will only be used to break ties if more than one node have the highest utility. Despite how close a goal seems to be to the previously selected one, it will only be picked up if it is the highest-utility one among the successors. But, when all goals have equal utility, the next selected node for expansion will be the closest one to the current node. In such case, distances are the only criteria to select the next goal. In the experimental section we will see how this slightly different behavior impacts the quality of the found plans.

4.3. Analysis of the Goal Selection Algorithm

In this section we will analyze our algorithm, determine the computational complexity of selecting goals using our approach, and compare it with similar works that also select goals. Given that the empty plan is always a solution for the satisficing no-hard-goals OSP task, any algorithm generating the empty plan is trivially complete. If we ignore this trivial case and let $\pi(G')$ be a plan achieving a given set of goals G' and $\mathbb{G}_{sol} = \{G' \mid \exists \pi(G'), C(\pi(G')) \leq c_b\}$, the set containing all the sets of goals for which a plan exists with cost less than or equal the cost bound, the algorithm will be asymptotically complete if it is able to find a solution for at least one $G' \in \mathbb{G}_{sol}$ if given enough time and memory. It will be optimal if it guarantees that it will find a plan for any of the goal sets in $\mathbb{G}^* = \{G^x \in \mathbb{G}_{sol} \mid U(G^x) \geq U(G^y), \forall G^y \in \mathbb{G}_{sol}\}$, the set of optimal goals sets.

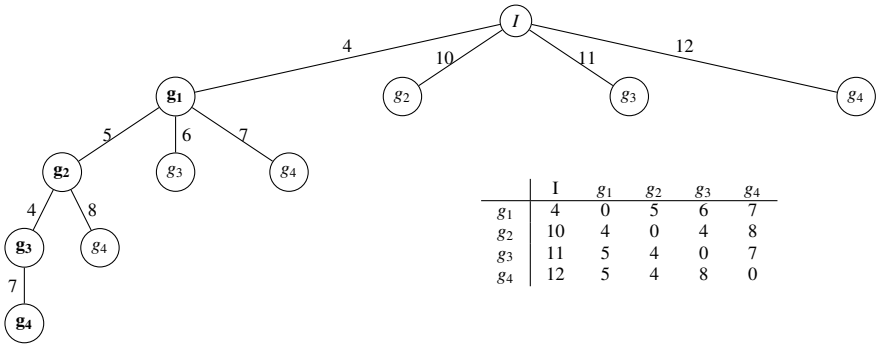
Theorem 1. *The algorithm is sound if the external planner is sound and returns a plan whose cost is less than or equal to the cost-bound.*

Proof. According to Definition 5, any plan whose cost is less than or equal to the cost-bound is a valid plan for the only soft-goals OSP problem. Let G' be a selected set of goals and $\pi(G')$ the plan found for it by the external planner. Given that the external planner solves a cost-bounded problem if it is sound and returns solutions not exceeding the cost-bound all the plans it finds will be solutions for the OSP problem and then $\pi(G')$ will be a valid solution. □

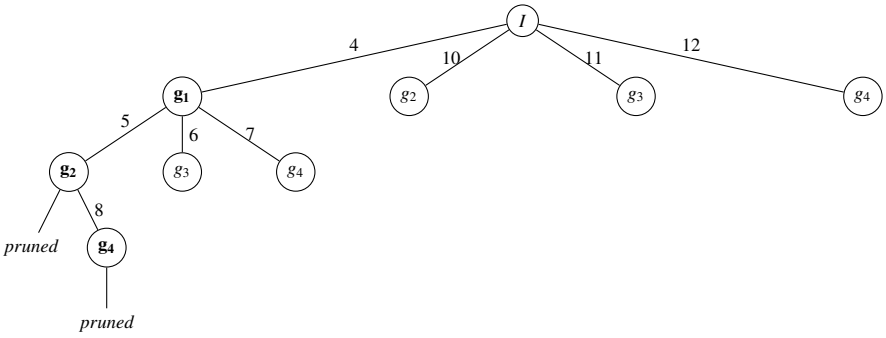
The algorithm is not complete due to the limited time given to the external planner. But, it is asymptotically complete and optimal if given enough time and memory.

Proposition 1. *If the planning space of the generated $P_{cb} = \{F, A, I, G', c, c_b\}$ is finite, the external planner is sound and complete and it is given enough time and memory resources, the algorithm will not prune any solution.*

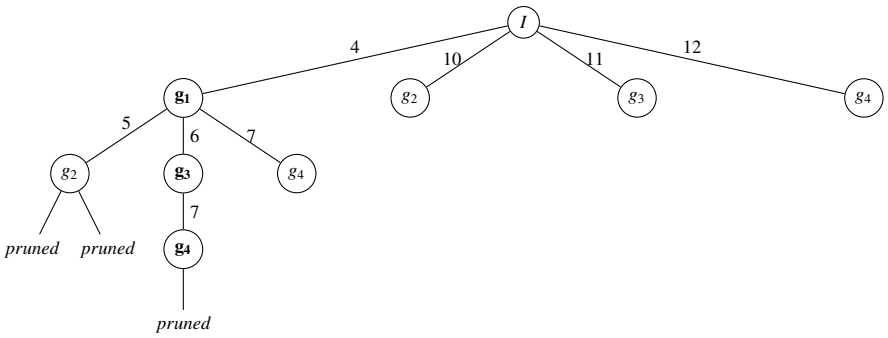
Proof. The only pruning we perform over a standard planner is due to the use of D, the set of pruned sets of goals. The set G' will be pruned (added to D) if and only if the external planner is not able to find a plan for it. Assuming finite planning space and enough time and memory, a sound and complete external planner will be able to either find a plan or exhaust the search space and prove no plan exists. Successors of a node in the DFBnB search only add one goal to the selected goals list. If there is no plan for the goals in node G', there will be no plan for any superset of them, so no descendant of it will be a solution. Therefore, the pruning defined by



(a) Initial selection of goals: $\{g_1, g_2, g_3, g_4\}$ is a terminal node improving $U(\emptyset)$.

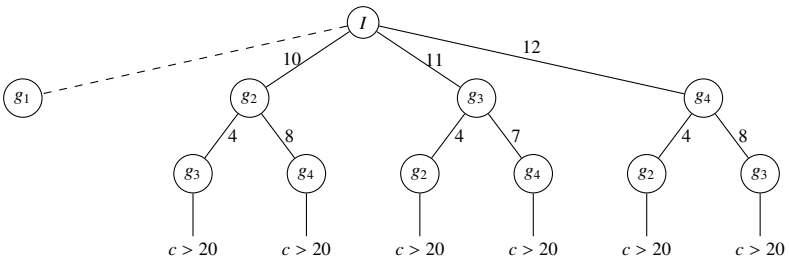


(b) Second terminal node: $\{g_1, g_2, g_4\}$. The algorithm backtracked until a plan was found for $\{g_1, g_2\}$ and its only non-pruned child was expanded.

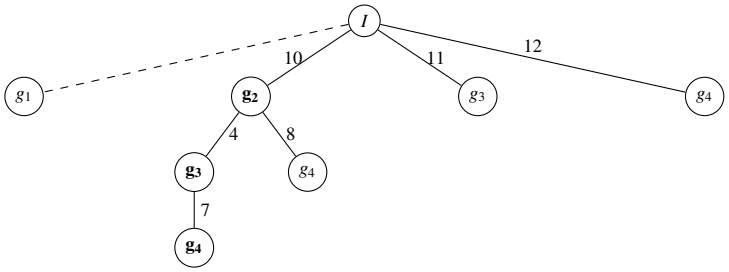


(c) Third terminal node: $\{g_1, g_3, g_4\}$. No plan was found for $\{g_1, g_2, g_4\}$.

Figure 4: Example of different iterations of goal selection and planning when all the goals have equal utility. The cost bound is 20. Nodes represent goals, numbers on arcs are the distances among them. The utility of a node is equal to its depth, and has been omitted for clarity. Goals selected at each iteration are marked in bold.



(d) No plan is found for $\{g_1, g_3, g_4\}$ and no goals' set under the cost-bound improves the best utility found so far. SELECT-GOALS-AND-PLAN has exhausted all its search space. All branches containing g_1 were explored in previous iterations and have been omitted for simplicity.



(e) ENFORCED-SELECT-GOALS-AND-PLAN ignores distances and finds $\{g_2, g_3, g_4\}$.

Figure 4: Example of different iterations of goal selection and planning when all the goals have equal utility. The cost bound is 20. Nodes represent goals, numbers on arcs are the distances among them. The utility of a node is equal to its depth, and has been omitted for clarity. Goals selected at each iteration are marked in bold.

using D does not prevent the planner finding a solution to the task and the complete algorithm (RPLD) will not prune any solution to the OSP task. □

Proposition 2. *If there is at least one element $G^x \in \mathbb{G}^*$ with one permutation $\sigma = (g_1, g_2 \dots g_k)$, $k = |G^x|$ such that $\Delta(\sigma) \leq c_b$ (at least a permutation of an optimal set of achievable goals has an estimated cost less than or equal to the cost-bound),* SELECT-GOALS-AND-PLAN *will eventually select it if given enough time and resources.*

Proof. SELECT-GOALS-AND-PLAN implements DFBnB, a complete and optimal search algorithm [31]. It will try to find a plan for any terminal node, as well as for any node such that the algorithm did not find a solution for one of its successors and the node's utility is higher than the maximum one. Then, SELECT-GOALS-AND-PLAN will iteratively find increasingly higher utility solutions until it eventually expands a branch containing a set of goals with optimal utility.

□

Proposition 3. *If $\Delta(\sigma) > c_b$ for all the permutations $\sigma = (g_1, g_2 \dots g_k), k = |G^x|$ of all $G^x \in \mathbb{G}^*$,* ENFORCED-SELECT-GOALS-AND-PLAN *will eventually select one of them if given enough time and resources.*

Proof. This Proposition is proved in a similar way as Proposition 2. ENFORCED-SELECT-GOALS-AND-PLAN is a regular DFBnB, where a plan is searched for all nodes increasing utility. Eventually an optimal node will be selected. □

Theorem 2. *If the external planner is sound and complete, the planning space finite, and the planner is given enough resources,* RPLD *is asymptotically complete and optimal.*

Proof. The proof is trivial considering Propositions 1, 2 and 3 and considering that the external planner will always find a plan for an achievable set if such a plan exists and it is given enough time and resources. □

Regarding computational complexity, the most computationally expensive component of our approach within a single *goal selection and planning* iteration is the generation of the relaxed plans needed for the distances' computation. For an *n-goals* problem, n suboptimal relaxed plans have to be extracted to compute Δ_{Ix}, whose time complexity is polynomial in the size of the task [32]. These relaxed plans are applied to obtain the new initial states for each goal. The complexity of the application operation is linear in the number of steps of the relaxed plans. Next, for each goal g_x, *n-1* relaxed plans need to be created to compute Δ_{xy}. Therefore, a total of $n + n(n-1) = n^2$ relaxed plans have to be computed.

The single other work directly using relaxed plans to select goals [23], has a worst-case complexity, when low oversubscription, of $n \times \sum_{i=1}^{n-1} i \approx n^3$ relaxed plans. In case of high oversubscription, i.e. only a few of the goals can be achieved, the practical complexity of their algorithm decreases dramatically, while ours remains constant, as we need to compute all the distances despite of the degree of oversubscription. There are two possible improvements to decrease the complexity of our algorithm. First, to compute Δ during goal selection, which would reduce the number of needed relaxed plans. In the worst case, when all goals have the same utility and all are selected, we would need n relaxed plans for the initial node, $n-1$ for the second level and so on, which reduces the complexity to $(n^2 + n)/2$ relaxed plans. Second, we can reuse part of the effort performed to create a relaxed plan, which consists of two time-polynomial phases; a forward expansion of the initial state until reaching a state where all goals are achieved and a backward selection of actions to construct the relaxed plan. Currently, we are performing both steps for all goals, but the first step can be reused. In order to compute Δ_{Ix}, we can expand the graph until all goals are reached. The obtained graph, which is exactly equal to the one generated for the farthest goal, can be reused to extract relaxed plans for all goals, thus saving $n-1$ expansions. The same can be done to compute Δ_{xy}, saving $n-1$ extra expansions. We plan to explore these improvements in the future, to evaluate if they really make practical differences in the time spent computing distances.

Once distances are computed, and before starting the first planning process we still need to select the goals. In comparison, Sanchez-Nigenda approach just needs to pick the set with highest estimated NET-BENEFIT and start planning. But the use of Depth-First search for the first selection of goals, with a maximum depth of n in case all the goals are selected, makes the maximum size of the search graph to be $(n^2 + n)/2$. This results in goal selection time to be negligible compared to the time needed to compute the relaxed plans, as there is no backtracking when searching for the first set of goals.

The construction of the Orienteering Problem (OP) by Smith also uses relaxed plans [1]. One relaxed plan for each goal is created at each node in the OP from which the goal can be achieved. This allows his algorithm to estimate the cost of reaching the goal from that node. The size of this set of nodes depends on the number of nodes in the OP from which the goal can be achieved, considering that the OP is a projection of the original state space into a subset O of the propositions of that space. The size of this OP is exponential on the number of independent propositions in O, which depends on a threshold that has to be defined for each domain, and

that, in the worst case, can be the complete state space for the problem. Our goals selection can be also recasted as an OP of linear size n, where goals are cities, utilities rewards and distances travel costs.

5. Experimental Setup

In this section we will present the experimental setup: planners, domains and problems, quality metrics and selection of parameters. All the experiments have been conducted on an Intel Xeon X3470 4 cores CPU at 2.93 GHz running Ubuntu 14.04.4. A maximum of 14 GB of memory out of the 16 GB available has been allowed. Only one out of the 4 cores has been used to avoid interactions among them to distort the results. Following the IPC rules, a total of 1800 seconds have been allotted to each problem. To compute the distances between goals we have used a LISP implementation based on the parser of the Sayphi planner [33], which implements the computation of relaxed plans as in the FF planner [8]. All solutions have been checked using the VAL plan validation software [34].

5.1. Planners

In addition to our full RPLD approach, we report the results of a non-incremental one, named RPLD_{first}, which stops after finding the first solution. Despite some recent work on optimal OSP [4, 5], there are no freely available satisficing planners tailored for it, as most of the planners focus on the NET-BENEFIT problem. But, as said, OSP problems can be expressed using PDDL3.0 *preferences* and there are also some compilations to classical planning [4] that can be used for comparison. Taking this into account we have compared our technique against:

- An optimal OSP approach, based on the combined results of the four configurations of the optimal OSP planner recently presented by Katz *et al.* [4]. The four configurations will be run and their results added as if they were a single planner. We will call it OPTIMAL in the tables. Even if we do not expect it to behave well in hard tasks, it has been included as a baseline for simple tasks.

- Two PDDL3.0 compliant planners: MIPS-XXL [20] and OPTIC [25]. MIPS-XXL is an optimal planner, but it is able to return intermediate non-optimal plans too. It ranked second in the last edition of the NET-BENEFIT track (IPC-2008). GAMER [21], the winner, exhausts the memory even with the simplest OSP problems and no other editions of that track have been carried out since then. Meanwhile, OPTIC is one of the rare modern planners supporting PDDL3 preferences, and although it is specially designed for temporal domains it also handles non-temporal ones.

- A compilation of OSP into classical planning: as said in Section 3, it is easy to adapt Keyder and Geffner compilation [27] to deal with OSP. This compilation has shown the best performance for NET-BENEFIT problems. To our understanding this compilation is similar to the *soft-goals* one of Katz *et al.* [4], but we will use a satisficing planner to find a plan instead of looking for the optimal one. We will call it COMPILED.

- A greedy BASELINE approach: it sorts all the goals by their utility and selects the highest utility one. A new problem is created that only contains this goal. If no plan is found, the goal with the second highest utility is selected and so on. Once a plan with one goal has been found, all the combinations of two goals with higher utility than the one found in the

previous step are selected and sorted by utility. Again, a search for a plan is performed. If a plan is found, the algorithm proceeds adding one more goal at a time. In order to improve the search, if G_i^n is a set of n goals for which no plan has been found at the current step, all the supersets of G_i^n are pruned. In problems with a large amount of goals, just finding the n combinations of k goals and sorting them by utility requires a significant amount of memory, exhausting it after a few levels.

- A satisficing version of the Katz *et al.* planner, named KATZ-SAT in the tables. It has been modified to output plans each time a better utility is found instead of just returning the optimal plan. Experiments have been also performed with the four configurations. In this case, they are not combined, the same way that the results of the different configurations of RPLD, COMPILED or BASELINE are not combined.

We have also performed experiments on some of the optimal domains proposed by Katz *et al.* [4] to evaluate how far from the optimal our solutions are. In those experiments OPTIMAL has been run in an Intel Xeon 3.4 GHz with 30 GB of memory, out of the 32 GB available, in an attempt to increase the number of problems for which an optimal solution is known.

RPLD, BASELINE and COMPILED rely on an external planner. In order to make fair comparisons, we use the same planner for all of them. The natural choice is to select one of the planners of the novel COST-BOUNDED track of the IPC-2018 [35]. They are designed exactly for our needs: finding a plan under a cost bound. COMPILED is a multiple cost function (MCF) task [4]. One cost function accounts for the penalties and the other one for the actions' cost. The former is increased when a goal is not achieved and the later when an action is applied. The cost-bound is set for the second one and the goal is to find plans minimizing the first one. All but one of the COST-BOUNDED planners rely on the Fast-Downward planner parser, which only allows the actions to increase a numerical function, the total-cost. Thus, they cannot be used for MCF problems as MCF requires some actions to increase the cost function and other actions to increase the penalties function. The only exception is the family of BFWS planners [36]. But these planners only allow to set the cost-bound over the formula defined on the metric, so it is not possible to bound the total-cost and minimize the penalties at the same time. If the metric minimizes the total-cost, then it forgoes all the goals to find an *optimal* zero-cost plan. If the metric minimizes the penalties, then it finds plans exceeding the cost-bound as there is no way to encode it in the problem. Finally, if the metric is a combination of both, it can still find incorrect plans as part of the budget for penalties can be used to include actions that exceed the cost-bound.

Another way to encode the cost-bound consists of using sub-optimal planners that support numeric preconditions: a numeric precondition checking that the cost-bound is not exceeded can be added to every action increasing it. This is valid for the RPLD, BASELINE and COMPILED versions. For the later, the metric can be set to minimize the penalties. According to IPC results, BFWS is the best sub-optimal planner supporting numeric preconditions. It is also a good example of a non-portfolio state-of-the-art planner: it was the first non-portfolio planner both in the cost-bounded track (4^{th} overall) and in the satisficing track (3^{rd} overall).

As these three techniques are planner-independent and to ensure results are not affected by the external planner used, we have also reviewed the competitors of the SEQUENTIAL-SATISFICING track of the IPC-2014 [37] and IPC-2011 [38]. In IPC-2014, the candidates, in descending order of overall score, are BFS(f), DAE_YAHSP, YAHSP3-MT and YAHSP3. The YAHSP family ignores numeric preconditions, creating incorrect plans that do not respect the cost-bound, leaving BFS(f), which can be considered a predecessor of BFWS, as the only reasonable election. More candidates

can be found in IPC-2011: PROBE, CBP2, DAE_YAHSP, YAHSP2, YAHSP-MT, CBP, LPRPG-P, ACOPLAN and ACOPLAN2. PROBE has the same problem as the YAHSP family, so we chose the second best, CBP2. We will use BFWS, BFS(f) [39] and CBP2 [40] for the BASELINE, COMPILED and RPLD approaches. BFS(f) sometimes returns invalid plans exceeding the cost-bound. This behavior is rarely observed in the COMPILED problems, but it can be oserved quite often in the RPLD and BASELINE approaches. The same applies to BFWS. It finds several invalid plans for COMPILED, where, as explained, the cost-bound cannot be used. But it does not find invalid plans for the other two approaches, which can use it (see Section 6.3 for details).

5.2. Domains and Problems Description

The former planners have been tested in the 14 domains of the SEQUENTIAL-SATISFICING track of the IPC-2011[5]. In order to create OSP tasks, a new numeric literal has been added to each domain to account for the cost-bound. All the actions increasing the cost of the plan have a new precondition that does not allow to apply them if their cost plus the current accumulated cost, exceeds the cost-bound. We generated new problems after transforming the domain. For each original problem of the IPC, the minimum cost found by any planner in the IPC-2011 has been used to create three equivalent problems, with a cost-bound of 25%, 50% and 75% of that minimum cost, respectively. These three values allow us to study the variations in performance for high (25%), medium (50%) or low (75%) oversubscription. In cases where a problem was not solved by any planner, an estimated cost of the problem has been established. This is the case of several *Floortile* problems, and in the last problem of *Nomystery* and *Sokoban*. As usual in OSP, all the original goals have been marked as soft-goals; no hard-goals are considered. All domains use action costs, except for *Tidybot* and *VisitAll* where the cost of the plan is the plan length. Both have been modified adding a cost of one to every action.

To test the influence of different distributions of utilities among goals, two versions of each problem have been defined. In the first one, that we will call `util1`, $u(g_i) = 1, \forall g_i \in \mathbb{G}$. In the second one, `util10`, the utility of each goal is a random natural value, $1 \leq u(g_i) \leq 10, \forall g_i \in \mathbb{G}$. This gives a total of six versions for each original problem (three degrees of oversubscription and two utility distributions), which results in 120 problems per domain.

As we mentioned before, OSP problems can be seen as Multiple Cost Functions (MCF) ones, with the utilities being the primary cost and the cost-bound the secondary one. If we only maximize utilities, or more commonly minimize penalties, the planners will search blindly with respect to the cost-bound. A way to tackle this would be to modify their heuristics, so they also take into account the cost-bound as a secondary criterion [30, 41]. However, this would imply modifying the code of the planners, and in the case of COMPILED, it would make it no longer planner-independent. An alternative way, which also keeps the compilation planner-independent, consists of modifying the problem metric. The problem can be converted into a kind of NET-BENEFIT-OSP one, i.e. both the total cost and the penalties have to be minimized. However, the focus needs to be on the penalties; the planner should not drop any achievable goal because its utility is lower than its cost. In order to test whether guiding planners by cost improves results or not, we have created problems with different metrics depending on a factor that relates the penalties and the plan cost: `minimize(+ (penalties) (* factor (/ (plan-cost) (cost-bound))))`. We have tried four different values for `factor`:

[5] Available at: http://www.plg.inf.uc3m.es/ipc2011-deterministic/

- `factor=0`: the planner only focuses on the penalties, using the cost-bound just as a way to prune branches during search.
- `factor=1`: the relative cost of the plan with respect to the cost-bound has the same weight in the metric than a goal with utility one.
- `factor=2`: the relative cost of the plan with respect to the cost-bound has the same weight in the metric than a goal with utility two. In our first set of experiments, where all goals have $u = 1$, this means the relative cost of the plan is equivalent to two goals.
- `factor=` $\sum_{g_i \in G} u(g_i)/2$: The relative cost of the plan with respect to the cost-bound is equal to half the addition of the utilities of all the goals. From the three options, this is the approach focusing more on the plan cost and less on the penalties.

This yields a total of four different configurations for COMPILED, MIPS-XXL and OPTIC, three of them taking into account the cost-bound to guide the planner and one searching blindly with respect to it. The higher the factor in the former metric is, the more informed the planner is with respect to the cost-bound. However, it will also be more likely that the planner will try to select states minimizing the plan cost instead of the penalties. In the extreme, if this factor is very high, the empty plan could be the best solution.

Tables 1, 2 and 3 describe various characteristics of the selected domains that could influence the results. Table 1 shows whether some goals are already achieved in the initial state and if they can be deleted. In that case, the utility of the empty plan will be positive and plans can have lower utility if they delete some of those goals. The table also shows which type of predicates appear in the goals. In *ParcPrinter* some predicate goals can be undone. For instance, `(Sideup)` which is never present in the initial state, or `(Notprintedwith)`, which is present in the initial state. But the latter cannot be re-achieved once deleted, so a valid plan will never delete it. There are also goals, like `(Hasimage)`, which are never present in the initial state and that cannot be deleted once achieved. In the case of *PegSolitaire* all goals can be undone and plans usually require to delete some of them. In *Scanalyzer*, `(analyzed car)` cannot be deleted, and as a consequence it never appears in the initial state, while `(on car segment)` can be undone and sometimes it appears in the initial state. In *Sokoban*, there are problems where stones have to be moved from their original locations to be placed again at the same place at the end. In *VisitAll* goals are permanent; once a location has been visited, it cannot be made *unvisited*. But the initial location is also a goal of the plan, so the empty plan always has some utility. In the case of *WoodWorking*, there is also a mix of both types of goals: `(colour)`, `(treatment)` and `(surface-condition)` can be undone, while `(available)` and `(wood)` are not reachable again once undone.

Table 2 shows the minimum, maximum and median number of goals of the problems of the selected domains. As it can be seen, there is a huge variation among domains.

Table 3 shows the empty-plan utility as a percentage of the total utility for all the domains where there is at least one problem with $U(\emptyset) > 0$. In general, the larger this value is, the more difficult will be to find a plan improving it. The first column shows the domain name and the number of problems where the utility of the empty plan is not null. The second and third column show the mean empty plan utility vs. total utility ratio for the `util1` and `util10` configurations, respectively.

Domain	Deletable goals	Goals at initial state	Predicates on goals
BARMAN	all	no	(contains container beverage)
ELEVATORS	all	no	(passenger-at passenger count)
FLOORTILE	no	no	(painted tile color)
NOMYSTERY	all	no	(at locatable location)
OPENSTACKS	no	no	(shipped order)
PARCPRINTER	some	yes	(hasimage sheet side image) (sideup sheet side) (stackedin sheet location) (notprintedwith sheet side color)
PARKING	all	yes	(behind-car car car) (at-curb-num car curb)
PEGSOLITAIRE	all	yes	(occupied location) (free location)
SCANALYZER	some	yes	(on car segment) (analyzed car)
SOKOBAN	all	yes	(at-goal stone)
TIDYBOT	no	no	(object-done object)
TRANSPORT	all	no	(at-package package location)
VISITALL	no	yes	(visited place)
WOODWORKING	some	yes	(available woodobj) (surface-condition woodobj surface) (treatment part treatmentstatus) (colour part acolour) (wood woodobj awood)

Table 1: Characterization of domain goals. The second column states whether a goal can be removed once it has been achieved. The third column shows if there are problems where some goals are already present in the initial state. The last column shows the predicates that can appear as goals.

Domain	Minimum	Maximum	Median
BARMAN	9	14	12
ELEVATORS	14	60	40
FLOORTILE	12	49	25
NOMYSTERY	6	15	11
OPENSTACKS	50	250	115
PARCPRINTER	36	90	54
PARKING	22	32	27
PEGSOLITAIRE	33	33	33
SCANALYZER	16	36	24
SOKOBAN	1	12	5
TIDYBOT	4	12	4
TRANSPORT	12	22	20
VISITALL	144	2500	962
WOODWORKING	9	140	98

Table 2: Minimum, maximum and median of the number of goals for each domain.

Domain	Mean utility of the empty plan	
	`util1`	`util10`
PARCPRINTER (20/20)	48%±3%	47%±4%
PARKING (13/20)	4%±3%	4%±4%
PEGSOLITAIRE (19/20)	44%±15%	45%±15%
SCANALYZER (14/20)	35%±24%	37%±22%
SOKOBAN (3/20)	5%±14%	6%±17%
VISITALL (20/20)	0%	0%
WOODWORKING (20/20)	12%±12%	13%±15%

Table 3: Mean utility of the empty plan, in percentage of the total utility, for the domains where the empty plan has utility bigger than zero. The numbers in parenthesis are the number of problems where $U(\emptyset) > 0$.

5.3. Plan quality metrics

We measure the performance of the different planners similarly to the last IPCs; it is relative to the performance of the other planners. For each problem, a planner gets a quality score between 0 and 1 in the following way. Let $(u_1, u_2 ... u_n)$ be the utility found by each planner for a given problem p, or 0 if the planner does not solve the problem. n is the number of planners, and $u_{max} = \max(u_1, u_2 ... u_n)$, the maximum utility found by any planner for this problem. For that problem, planner i receives a quality score of u_i/u_{max}. The total score of a planner for a domain is the sum of the scores for each individual problem in that domain: $\sum_p u_i/u_{max}$. As there are 20 problems per domain, the score will be in the range 0 to 20.

This score schema is the standard one used by the planning community, but it has some drawbacks [42]. For instance, it does not take into account the difficulty of the problems. Both easy and difficult problems contribute equally to the score. An alternative score that partially solves this problem is to consider the accumulated utility for a domain. Difficult problems tend in general to have more goals, and consequently their maximum achievable utility tends to be also higher. Adding the utilities of the goals achieved for all the problems in a domain gives a good idea of the performance of a planner in that domain, emphasizing the performance on difficult

problems. For each domain, planner i will obtain a utility score of $\sum_p u_i / \sum_p u_{max}$, a number in the range 0 to 1. To ease comparisons between quality and utility, the utility score will be multiplied by 20, so both metrics will be in the range 0 to 20.

5.4. Selection of parameters

Our algorithm receives one parameter, the time to search for a plan before a set is discarded and marked as probably still oversubscribed. The number of selected goals can be considered to set this parameter value. Usually, the more goals selected, the longer the time needed to find a plan. But the time bound for the planner also depends on the domain; there are domains where finding a plan is more difficult than in others, even in problems with a similar number of goals. As an estimation of both aspects, in our previous work [10] we used the time spent in distances computation, which depends on both the number of goals and the problem. The planner was given the same time used to compute distances to find a plan. As this time was sometimes very small (in the order of hundreds of milliseconds), a minimum of 10 seconds was allotted. This approach has good performance in small and medium size problems, but when the number of goals is high, the time spent computing distances is high, too. This usually leads to devoting too much time to unsolvable sets. In turn, this exhausts the available time for planning, resulting in no solution found.

Alternatively, we can set the planning time by analyzing the general performance, in terms of time, of the planners we are using, and to set a fixed value for the planning time. When faced with the original non-oversubscribed problems, one of the used planners, CBP2, found the first solution in less than 45 seconds in 69% of the solved problems, in less than 60 seconds in 74%, and in less than 90 seconds in 77% of them. In contrast, only 11% of the problems needed more than 300 seconds to be solved. This is consistent with the literature; if a planner does not find the first plan quickly it is very likely it is not going to find it at all [43]. In our case we are only interested in this first solution and we are planning for easier problems than the original ones, given that they will almost always have a lower number of goals. So 90 seconds seems to be a good choice for the time to search for a plan before giving up. This alternative has the disadvantage of not increasing the time for complex problems, but this is somehow compensated by the incremental behavior. Usually, it is better not to spend too much time with a set of goals, finding at least one solution that might be improved later. This is commonly also true in real applications: finding a bad solution is better than not finding a solution at all.

BASELINE will also be given 90 seconds to solve each combination of goals before marking it as *non-achievable* and switching to the next one. If it marks all the sets for a given level as *non-achievable*, the time is doubled and search starts again with the highest utility set.

6. Experimental Results

In this section we summarize the results of the paper. First, we will analyze the behavior of our RPLD approach (Sections 6.1 and 6.2). The objective of this first set of experiments is to evaluate how long it takes to compute the distances and select the goals, and to assess how often the procedure returns sets that are still oversubscribed. Next, we compare the performance of RPLD against the remaining planners in terms of quality and utility (section 6.3).

6.1. Distances Computation

Figure 5 shows graphically the time RPLD needed to compute the distances for each domain and problem. For the sake of simplicity, this figure does not show data for *VisitAll*. In this domain, the algorithm exhausts the 1800 seconds bound while computing the distances in all problems, except for the four first ones. This is due to its high number of goals. In this domain, times range from 40 seconds for the simplest problem (144 goals), to about 1300 seconds for a 324-goals problem. For problems with 400 goals or more, distances cannot be computed in the alloted time. *Scanalyzer* is not shown either. Our parser exhausts the available memory while instantiating the problem in most cases. In the four problems it solved, all of them with 16 goals, the time to compute distances lies between 17 and 20 seconds.

In the remaining domains, distances computation times vary from less than a tenth of second to hundreds of seconds. In *Barman*, *Floortile*, *Parcprinter*, *PegSolitaire*, *Sokoban* and *Tidybot* distances are computed in less than 10 seconds, and in many cases in less than one second. Meanwhile, *Parking* and *Transport* always need less than 30 seconds. In *NoMystery*, time grows dramatically with the number of goals, ranging from less than one second for a 6-goal problem to more than 300 seconds for a 14-goals one. But time also depends on the problem structure as the two 15-goals problems take 192 and 77 seconds, respectively. *Openstacks* smallest problems, with 50 goals, only take 2 seconds, but one of the two biggest ones, with 250 goals, exhausts the 1800 seconds, taking the other one 1500 seconds. In *Elevators*, time varies between one second (14 goals) and more than 400 seconds (60 goals). In *Woodworking*, most problems, having 69 or more goals, need more than 100 seconds. The exception is a problem with only 9 goals which takes less than 1 second.

So, as expected, the distances computation time depends on the domain and problem. Problems with similar number of goals can require very different computation times, even in the same domain. For example, problems in *PegSolitaire* and *Parking* have similar number of goals, in the order of 30. In *PegSolitaire* the computation takes about one second, while in *Parking* it takes more than 10. Even if *ParcPrinter* is one of the domains with more goals, up to 90 with a median of 54, distances are computed always in less than three seconds. Also in some domains, as previously said for *NoMystery*, problems with the same number of goals can take values as different as 77 and 192 seconds.

Figure 6 shows the computation time of distances per goal. It varies between some milliseconds for the simplest problems of *Floortile* to about 20 seconds for some *NoMystery* problems. When considering problems with a similar number of goals, *ParcPrinter* has the lowest distances computation times per goal, while *NoMystery* and *Elevators* have the highest. A total of n^2 relaxed plans have to be computed, so in most cases the time tends to grow polynomially with the number of goals. The two extreme cases are *Openstacks*, where small increases on the number of goals result in dramatic increases on the time per goal and *Woodworking*, where the time per goal grows slowly with the number of goals.

6.2. Goals selection

Given that a Depth-First algorithm is used to select goals, time devoted to perform this step is quite short. The first goal set is selected in a millisecond or less in 66% of the problems, and in less than a tenth of a second in 96% of them. Only 12 problems out of the 1680, 7 of them in *Visitall*, needed more than one second. The maximum time observed was 12 seconds for one of the *Visitall* problems.

We analyze now if distances are a good estimator of the real costs of achieving goals. Or, at least, if they allow us to select non-oversubscribed sets. We have allotted the external planner

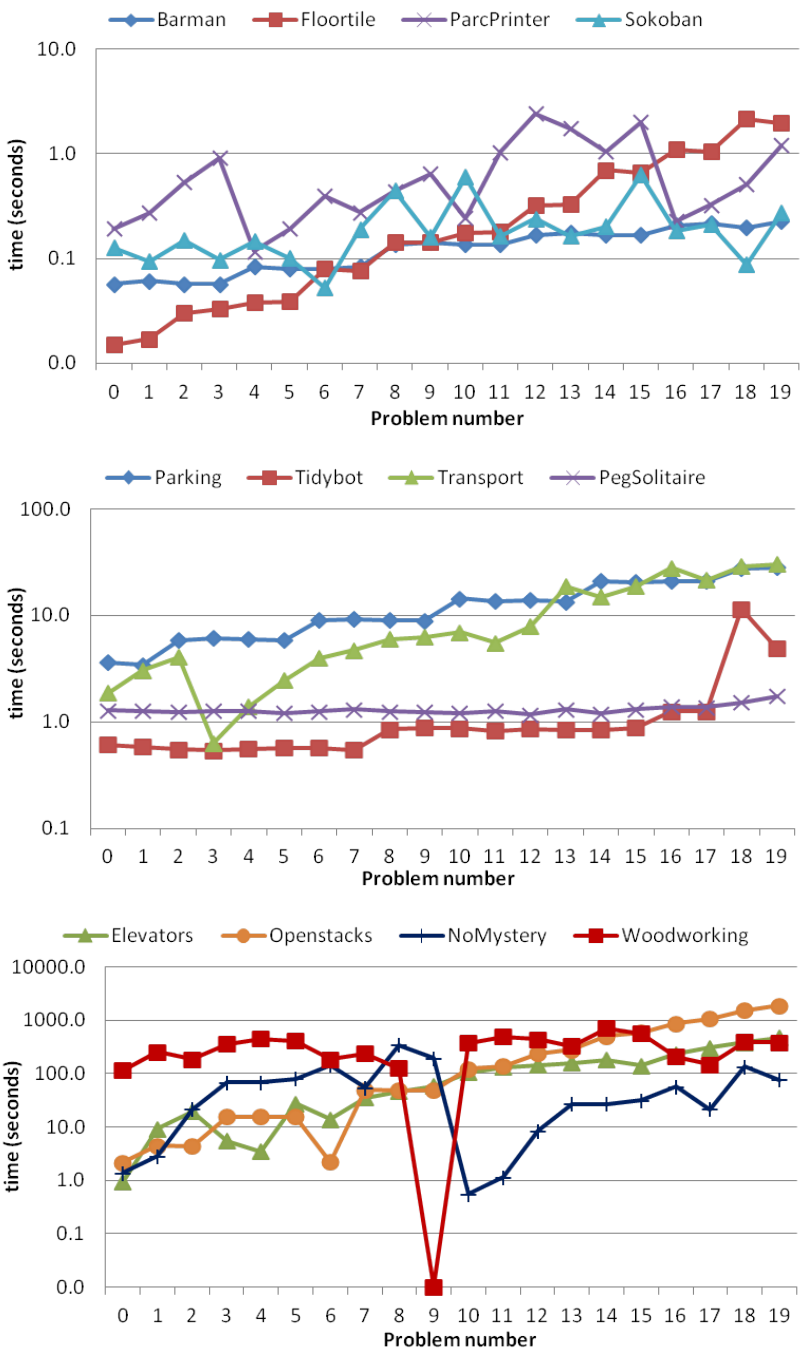


Figure 5: Time to compute distances. The *x* axis represents the problem number, and the *y* axis represents the time in seconds spent to compute the distances for each problem (logarithmic scale). Data for *VisitAll* and *Scanalyzer* domains are not shown.

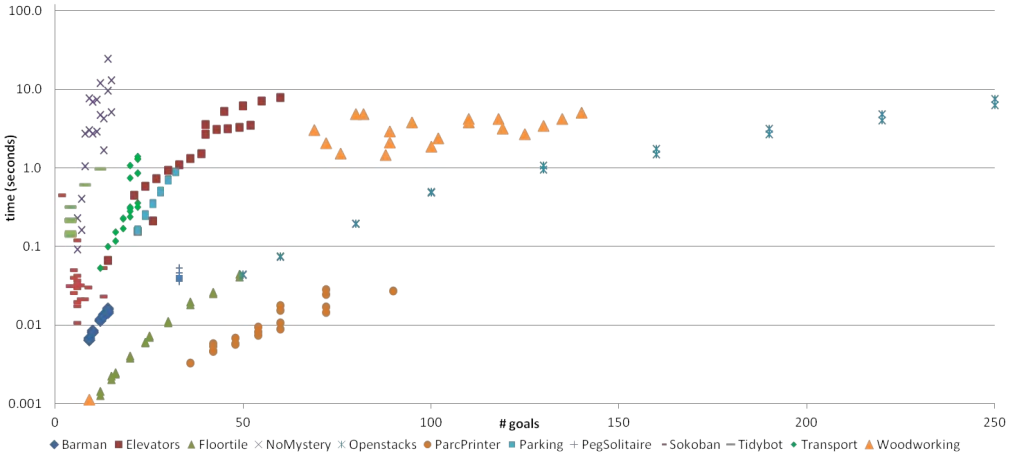


Figure 6: Time to compute distances per goal. The x axis represents the number of goals, and the y axis represents the average time in seconds spent to compute the distances per goal (logarithmic scale). Data for *VisitAll* and *Scanalyzer* domains are not shown.

the remaining time until the 1800 seconds limit to find a plan for the initial selected set. We have performed the experiments with BFWS, BFS(f) and CBP2. Even if not finding a plan is a good indicator of oversubscription, it does not necessarily mean the set is still oversubscribed. It may happen that another planner would be able to find a plan.

Table 4 shows the percentage of problems for which a plan for the first selected set was found by any of the planners. A percentage close to 100% means that there was little oversubscription in the selected sets for that domain and that our approach was good in discovering a set of achievable goals. Conversely, low percentages mean most sets were still oversubscribed: distances were highly underestimating the cost of reaching goals, which made the algorithm select more goals than those that can be achieved. Again, *Scanalyzer* and *Visitall* have been removed from the analysis as the low number of problems for which goals could be selected does not allow us to draw relevant conclusions.

As expected, data show that our approach tends to overestimate the number of achievable goals. When all goals have the same utility, more than half of the problems seem to be still oversubscribed in 8 out of the 12 domains: a plan for the selected goals was found in less than 50% of the problems. The more oversubscribed domains are *Elevators* (only 12% of the problems are solved), *Floortile* (12%), *PegSolitaire* (18%) and particularly *Transport* (2%). In these domains, distances are not a good estimator of real costs. On the other hand *NoMystery*, *Openstacks* and *Woodworking* obtain quite good results, as most problems were not oversubscribed.

The results for high oversubscription are generally better. This is not surprising: as the cost-bound is increased (low oversubscription) more goals can be added to the set and the chances that it is oversubscribed are higher. Exceptions are *PegSolitaire* and *NoMystery*. In *PegSolitaire*, costs of the plans are small, between 7 and 19. Several goals can be achieved at zero cost once the first one is achieved and almost half of them are true in the initial state. Having a higher cost bound may help to reach the goals that are not true in the initial state even if the set is bigger, given that many of the newly added goals will be already available in the initial state or can be

	util1				util10			
Domain	**H**	**M**	**L**	μ	**H**	**M**	**L**	μ
BARMAN	50%	40%	35%	42%	65%	65%	60%	63%
ELEVATORS	20%	10%	5%	12%	50%	60%	50%	53%
FLOORTILE	30%	5%	0%	12%	90%	90%	75%	85%
NOMYSTERY	70%	60%	80%	70%	55%	95%	95%	82%
OPENSTACKS	95%	95%	95%	95%	85%	95%	95%	92%
PARCPRINTER	50%	40%	40%	43%	65%	65%	60%	63%
PARKING	50%	30%	30%	37%	75%	85%	100%	87%
PEGSOL	5%	20%	30%	18%	30%	85%	95%	70%
SOKOBAN	70%	45%	50%	55%	50%	40%	40%	43%
TIDYBOT	55%	25%	15%	32%	60%	30%	25%	38%
TRANSPORT	5%	0%	0%	2%	10%	10%	10%	10%
WOODWORKING	100%	100%	100%	100%	100%	100%	100%	100%

Table 4: Percentage of problems for which a plan is found for the initially selected goals set, allowing the planner all the remaining time after selecting the goals, up to 1800 seconds. We show data for each utility profile and oversubscription degree. Data for *Scanalyzer* and *Visitall* domains are not shown.

achieved with zero cost once another goal is reached. In *NoMystery*, differences between the three degrees of oversubscription are smaller and do not allow us to draw any clear conclusion.

Results are better when goals have different utilities. At least half of the problems are not oversubscribed in nine domains. The main reason is that for the `util10` setting, goals are selected greedily based on their utility, using the distances just to break ties and check if the cost-bound has been reached. Thus, in general less goals are selected. The highest improvements occur in *Floortile* and *PegSolitaire*. The main exception is *Sokoban*, where the distances seem to capture well which goals are related and not considering them in the selection process results in worse performance. Results for high oversubscription are better, except for *PegSolitaire*, *NoMystery* and *Parking*. This had been observed before when all goals had the same utility. In *NoMystery*, the cost of reaching a goal depends on the connections among locations. Greedily selecting the goals tends to add a low number of goals to the list, and increasing the bound eases finding a path reaching all of them even if more goals are added. *Parking* gets a 100% of non-oversubscribed problems for low oversubscription. It seems that, again, increasing the cost bound eases finding plans even if the number of selected goals is also slightly raised.

Next, we wanted to measure to which extent we are underestimating real costs. Therefore, we have allowed the planner to search for 90 seconds, removing one goal if a plan is not found. This is repeated until a plan is found or no goal is left. We can get an estimation of how oversubscribed the initial set was, or at least how oversubscribed it was to be solved using our approach, by comparing the number of achieved goals with the number of initially selected ones.[6] Table 5 shows the achieved/selected goals ratio for each domain, utility profile and oversubscription degree.

The worst results are obtained in *Barman*, *Tidybot* and *Transport*. For *Barman* the main reason is a combination of the low number of goals and the poorly informed the relaxed plan heuristic is in this domain (it is explicitly designed to challenge this heuristic). No more than

[6]This is only an estimation, as 90 seconds are usually not enough to exhaust the search space. As we are interested in finding an achievable set and not on the time spent to do it, the overall process has not been limited by time, so finding this set may have taken more than 1800 seconds.

	util1				util10			
Domain	**H**	**M**	**L**	**Total**	**H**	**M**	**L**	**Total**
BARMAN	63%	42%	33%	39%	58%	39%	31%	34%
ELEVATORS	81%	73%	70%	69%	79%	81%	79%	76%
FLOORTILE	86%	71%	64%	68%	98%	96%	94%	93%
NOMYSTERY	87%	91%	94%	90%	72%	98%	99%	94%
OPENSTACKS	100%	100%	100%	100%	90%	100%	100%	98%
PARCPRINTER	95%	93%	92%	92%	80%	74%	74%	75%
PARKING	74%	69%	64%	66%	83%	89%	97%	91%
PEGSOL	63%	85%	88%	80%	83%	98%	100%	94%
SOKOBAN	86%	80%	78%	81%	61%	73%	77%	77%
TIDYBOT	59%	53%	51%	48%	61%	44%	58%	44%
TRANSPORT	45%	42%	45%	42%	42%	39%	45%	39%
WOODWORKING	100%	99%	98%	98%	99%	100%	100%	99%

Table 5: Percentage of selected goals achieved allowing the planner 90 seconds to find a plan. The last column for each utility profile shows the accumulated results for all problems for a given domain: the total number of goals achieved in that domain divided by the total number of goals selected. Data for *Scanalyzer* and *Visitall* domains are not shown.

five goals are selected in any problem in the high oversubscription setting, and four or less goals are selected in 13 out of the 20 problems. Not achieving just a single goal of the selected ones results in a high decrease in the percentage of achieved goals. For example, if only two goals are achieved when three goals are selected, the success ratio is 66.6%. This was observed in seven out of the 20 problems. For medium and low oversubscription, the results are worse, as expected. In this domain, the results of `util10` are worse than those of `util1`. The reason is that it has two types of goals, with one of them much easier to achieve than the other one. To some extent, this is captured by our approach when all goals have the same utility. But when goals have different utilities, the greedy approach used to select goals ignores this fact.

Tidybot is also penalized by the low number of goals as in many problems only one or two goals are selected. In the case of *Transport*, results are also quite bad for all the degrees of oversubscription and both utility profiles. The heuristic highly underestimates the cost of achieving the goals, resulting in highly oversubscribed sets. Actually this was reflected on the bad results of the previous table.

The good results for *Elevators*, *Floortile*, *Parking* and *PegSolitaire* contrast with their poor performance in Table 4. This indicates the original sets were not too oversubscribed and that our estimation was just adding a few extra goals. We obtain good results in *NoMystery* and *OpenStacks*. This was expected considering their performance in Table 4. *Parcprinter* also shows good results, but when goals have different utilities, the greedy selection does not take into account the tight interaction between goals: up to six goals are used to describe the way a given paper sheet must be printed.

In *Sokoban* no major differences can be seen regarding utility, except for high oversubscription. But, differences are probably due to a small number of problems, where one goal is not achieved, given that this is one of the domains with a lower number of goals. Finally, results for *Woodworking* are quite good, but slightly worse than those in Table 4. These results show that 90 seconds can be sometimes not enough in that domain to find a plan for solvable problems.

Results show that the distances overestimate the number of achievable goals in most problems. However, it may happen they are underestimating for the remaining ones, as they could return non-oversubscribed sets that contain far less goals than those that could be really achieved.

We will discuss that in the next section, where we will compare the results of the first selection of goals against the results obtained by the whole RPLD approach, which iteratively tries to add new goals once a plan has been found.

6.3. Scores: quality and utility of the plans

Tables 6 and 7 show the accumulated scores in quality and utility for the 14 domains. We just show the results of the best version for each approach, except in cases where various versions have a similar score, but behave differently depending on the domain.

	util1			util10		
Planner	**High**	**Medium**	**Low**	**High**	**Medium**	**Low**
BASELINE-BFWS	185.0	184.0	181.5	218.8	215.7	211.8
COMPILED-BFS(f)	131.2	131.9	127.2	127.9	120.0	121.9
COMPILED-BFWS	109.1	106.4	112.1	143.5	129.9	122.9
COMPILED-CBP2	161.5	129.3	108.2	161.2	129.0	109.4
RPLD-CBP2	225.7	208.1	188.4	226.1	214.9	202.2
RPLD-BFWS$_{first}$	203.5	196.1	198.9	191.5	200.2	204.5
RPLD-BFWS	**230.8**	**227.7**	**226.5**	**236.2**	**235.3**	**234.5**
EMPTY	48.2	39.5	34.9	46.2	38.1	33.9
KATZ-SAT	181.9	154.8	137.9	190.4	163.4	147.0
OPTIMAL	96.0	68.0	51.0	96.0	68.0	54.0
MIPS-XXL	94.2	73.8	63.9	94.7	77.5	68.4
OPTIC	176.8	158.2	151.8	177.0	156.1	149.4

Table 6: Quality: summary of the results for all the domains.

	util1			util10		
Planner	**High**	**Medium**	**Low**	**High**	**Medium**	**Low**
BASELINE-BFWS	185.1	178.2	174.0	221.1	212.3	205.6
COMPILED-BFS(f)	134.6	131.3	125.6	130.0	119.3	120.3
COMPILED-BFWS	112.4	105.8	106.0	146.6	126.8	116.2
COMPILED-CBP2	157.3	123.4	102.1	157.1	123.2	102.3
RPLD-CBP2	230.5	208.0	186.4	231.9	216.6	202.4
RPLD-BFWS$_{first}$	202.5	195.5	197.9	195.5	202.9	206.7
RPLD-BFWS	**235.0**	**228.5**	**226.6**	**241.8**	**238.7**	**237.1**
EMPTY	50.9	40.8	35.8	49.1	39.5	34.6
KATZ-SAT	183.4	151.5	133.2	194.5	160.1	140.9
OPTIMAL	88.2	60.9	45.8	89.0	61.6	48.2
MIPS-XXL	86.1	70.4	62.1	87.1	73.9	65.3
OPTIC	174.9	159.4	153.3	177.3	157.8	150.0

Table 7: Utility: summary of the results for all the domains.

The best results, both in quality and utility, are obtained by RPLD with the BFWS planner, with CBP2 planner close to it in high oversubscription but farther away in medium and low. The results with BFS(f) are worse, as exceeding the cost bound makes invalid the found plans in 532 out of the 1680 problems. RPLD-BFWS obtains a slightly worse score when the degree of oversubscription

decreases, as expected (see Section 6.2). Scores are smaller for util1 than for util10, but for util10 the difference with the second classified, BASELINE-BFWS is smaller too.

Our non-incremental approach, RPLD-BFWS$_{first}$ is the second best in quality and utility for the util1 setting. It gets 86% to 88% of the quality or utility of the incremental approach, showing that the first selection of goals is quite good, even if it can be improved by adding more goals once a plan has been found. Its performance slightly decreases for util10, obtaining a 81% to 87% of the score of the incremental version and falling below BASELINE-BFWS. But it still beats all other approaches. As a reminder, the goal selection is mainly driven by utilities for util10, with distances only used to break ties. The slight decrease in score for high oversubscription could indicate that distances are not only good to have an estimation of how many goals can be achieved, but also point out which goals should be tried together as they seem to be closely interrelated. However, differences are so small to be conclusive.

BASELINE-BFWS, obtains the second place for all the oversubscription degrees, with quality scores 18-45 points lower than RPLD-BFWS. BASELINE-CBP2, not shown in tables, behaves slightly worse for high oversubscription and quite worse for medium and low, in a somehow similar way as in RPLD. This seems to indicate that CBP2 does not properly handle harder problems. Its equivalent BFS(f) version is again penalized by the high number of invalid plans found. Both quality and utility are higher for util10 than for util1, getting closer to RPLD and surpassing RPLD$_{first}$. It seems greedily selecting goals works better when goals have different utilities.

OPTIC and KATZ-SAT are the third classified planners. Tables show the results of the factor=1 version for OPTIC, which is slightly better than the non-cost-guided (factor=0) and the factor=2 ones. factor=2 obtains 24 points less than the other approaches for the util1 high oversubscription setting. When the guidance on the cost is increased (factor=$\sum$u/2), results are quite bad, because the planner tries to reduce the cost focusing on the actions' costs instead of minimizing the penalties. The best KATZ-SAT version is the blind one. It is better than OPTIC in high oversubscription, equivalent in medium and worse in low, with better results in util1.

Unlike in the NET-BENEFIT problem, the COMPILED approach does not work well and it only beats MIPS-XXL. Running with BFS(f), the best results are obtained with factor=1, virtually tied with factor=2. factor=$\sum$u/2 obtains worse results except for low oversubscription, where it has the best performance. When no guidance about the cost is given on the metric (factor=0), COMPILED-BFS(f) is not able to find any solution other than the obvious ignore-all-soft-goals one. BFS(f) produces a lot of invalid plans exceeding the cost, but usually at least one or two correct plans are found for each problem in this case. Interestingly, the best performance is obtained by the non-cost-guided version when using CBP2 for COMPILED. The factor=1 and factor=2 versions obtain between 5 to 8 points less. Giving more weight to the cost, as factor=$\sum$u/2 does, results in a quite poor score, particularly in high oversubscription. This shows that guiding the search by the cost can be a good or bad idea depending on the planner used for the COMPILED problems. Comparing both planners, COMPILED-CBP2 clearly dominates in the high oversubscription tasks, while COMPILED-BFS(f) dominates in the low oversubscription ones. COMPILED-BFS(f) is slightly better for medium oversubscription when all goals have the same utility and behaves worse when using different utilities.

MIPS-XXL scores even below OPTIMAL for high oversubscription, far away from the other ones, mainly because it is not able to parse most problems. Its performance does not seem to depend on the different versions of the problems, except for factor=$\sum$u/2, where results are worse.

No major differences can be seen between quality and utility; planners are classified in the same order. A detailed analysis shows that the utility scores of RPLD are in general slightly better

than the quality ones, and the difference with the runner up is also slightly increased. However, differences are too small to draw conclusions.

Tables 8 and 9 show the number of domains where each approach has the highest quality and utility, respectively. Unlike the previous tables, which only showed the results obtained by the best version of each planner, these tables combine the results of all the configurations: BASELINE/RPLD shows the number of domains where BASELINE/RPLD is the best approach, either running with BFWS, CBP2 or with BFS(f). The values obtained with the different `factors` are also combined for MIPS-XXL and OPTIC. Similarly, results obtained by any planner with any `factor` are combined for COMPILED. The performance of the best versions of each planner presented in the previous Tables 6 and 7 is shown in parenthesis. As it can be seen, RPLD dominates both in quality and utility.

planner	**util = 1**			**1 ≤ util ≤ 10**			**total**
	25%	**50%**	**75%**	**25%**	**50%**	**75%**	
BASELINE	1(0)	1(0)	2(2)	1(0)	3(3)	4(4)	12(9)
COMPILED	3(2)	2(1)	1(1)	3(2)	1(1)	1(1)	11(8)
RPLD	**9(4)**	**9(7)**	**8(6)**	**7(4)**	**5(2)**	**5(2)**	**42(25)**
KATZ-SAT	3(3)	2(2)	2(2)	3(3)	3(3)	2(2)	15(15)
MIPS-XXL	1(1)	1(1)	0(0)	1(1)	1(1)	0(0)	4(4)
OPTIC	2(1)	1(0)	1(0)	3(1)	2(0)	2(0)	11(2)

Table 8: Score: number of domains where each approach obtains the highest score. Numbers in parentheses show results obtained by the configuration with the best overall score for each technique (BASELINE$_{bfws}$, COMPILED$_{bfws}$, RPLD$_{bfws}$, KATZ-SAT$_{blind}$, MIPS-XXL$_{non-guided}$ and OPTIC$_{factor=1}$) .

planner	**util = 1**			**1 ≤ util ≤ 10**			**total**
	25%	**50%**	**75%**	**25%**	**50%**	**75%**	
BASELINE	1(0)	1(0)	2(2)	1(0)	4(3)	3(3)	12(8)
COMPILED	3(2)	2(1)	1(1)	3(2)	1(1)	0(0)	10(7)
RPLD	**10(4)**	**9(7)**	**8(6)**	**7(5)**	**5(2)**	**6(3)**	**45(27)**
KATZ-SAT	3(3)	1(1)	2(2)	4(4)	3(3)	2(2)	15(15)
MIPS-XXL	1(1)	1(1)	0(0)	2(2)	1(1)	0(0)	5(5)
OPTIC	2(1)	1(0)	1(0)	4(2)	2(0)	3(0)	13(3)

Table 9: Utility: number of domains where each approach obtains the highest utility. Numbers in parentheses show results obtained by the configuration with the best overall utility for each technique.

RPLD obtains its best results in *Elevators*, *NoMystery*, *Openstacks* and *Transport*. Interestingly, in the first three domains the selected goals sets are usually not too oversubscribed, as shown by Table 5, while *Transport* is one of the domains where our approach generates worse estimations of the achievable sets. The good performance of our approach in this last domain is probably due to a bad performance of its competitors, more than to its own merits.

The worst performance of RPLD is in *Scanalyzer*, where it is unable to ground the problems. It also obtains a bad performance in *Visitall*, where distances cannot be computed due to the high number of goals. *Barman* is mostly dominated by the COMPILED versions. In *Floortile*, both RPLD and OPTIC obtain the best results, with a small advantage for the former. There is also a tie between those two approaches in *ParcPrinter*, with RPLD getting the best results when all goals have the same utility and OPTIC when utilities vary. In *PegSolitaire*, COMPILED, MIPS-XXL and OPTIC

obtain similar results, although RPLD is also competitive. *Scanalyzer* is dominated by OPTIC, while COMPILED's performance is close. In *Sokoban* and *Tidybot* the low number of goals makes BASELINE a good competitor, but RPLD also obtains good results, particularly in the latter domain. OPTIC is not able to solve any *Tidybot* problem as the domain includes negative preconditions. COMPILED is also the best performer in *Visitall*, but OPTIC obtains good results in utility. Finally, COMPILED and RPLD obtain similar results in *Woodworking*.

As expected OPTIMAL obtains poor results in general. The problems we use belong to the satisficing track of the IPC and are in general too difficult to be solved optimally. To better compare how far from optimal our approach is, we have conducted a second set of experiments with optimal domains, using the ones created by Katz *et al.* [4]. They give a utility of 10 to original goals of the problem and a random value between 1 and 5 to 5% of the facts of the problem. In addition of working with bounds of 25%-75%, they also have problems with 100% of the cost. We have used the four configurations of OPTIMAL to search for optimal plans in those domains. Table 10 shows in how many of the problems for which an optimal solution is known our approach finds the optimal solution too (*opt.* column). It also shows its percentage of the optimal quality score (*score* column) considering only those problems, computed as described in Section 5.3. For example, our approach optimally solves 12 out of the 16 problems for which an optimal solution could be computed in BARMAN 25%. Considering those 16 problems, the score of our approach is 96% of the optimal one. The quality score is very often close to the optimal one, and our approach solves many problems optimally or near optimally in most domains.

	25%		**50%**		**75%**		**100%**	
Domain	**opt.**	**score**	**opt.**	**score**	**opt.**	**score**	**opt.**	**score**
BARMAN	12/16	96%	6/8	98%	4/8	92%	0/8	88%
ELEVATORS	20/20	100%	18/19	100%	15/19	97%	9/17	94%
FLOORTILE	6/9	98%	3/6	95%	2/4	97%	2/2	100%
NOMYSTERY	15/18	91%	11/16	95%	9/10	99%	9/9	100%
OPENSTACKS	18/20	98%	15/20	96%	10/17	89%	11/17	87%
PARCPRINTER	10/14	99%	6/9	97%	6/9	98%	8/8	100%
PARKING	5/12	92%	0/3	83%	0/1	90%	0/0	0%
PEGSOL	4/20	92%	4/20	93%	5/19	96%	16/18	99%
SCANALYZER	9/12	88%	9/9	100%	7/9	98%	5/9	96%
SOKOBAN	19/20	99%	14/20	97%	14/20	97%	10/19	95%
TIDYBOT	20/20	100%	17/20	99%	10/19	91%	9/14	92%
TRANSPORT	9/14	96%	10/11	99%	8/11	96%	4/8	93%
VISITALL	9/17	98%	9/12	99%	8/9	99%	8/9	99%
WOODWORKING	7/19	85%	2/11	87%	3/6	98%	3/4	99%

Table 10: Results on the optimal versions of the domains as defined in [4]. For each oversubscription degree the first column shows the number of problems for which an optimal solution is known and how many of them we solved optimally. The second column shows the percentage of the optimal score we obtain.

7. Conclusions and Future Work

In this paper we have presented RPLD, a planner-independent technique to solve satisficing OSP problems. To our knowledge it is the only off-the-shelf approach currently available for suboptimal OSP. It is based on selecting a subset of all the goals by estimating the cost of achieving them by means of distances computed using relaxed plans. We have tested our approach with

three planners, CBP2, BFWS and BFS(f), in fourteen domains, with three oversubscription degrees, and two utility profiles. As there are no other freely available planners tailored for the suboptimal OSP problem, we have compared our approach against two planners supporting soft-goals (MIPS-XXL and OPTIC), a greedy selecting-goals baseline, four different configurations of an OSP optimal planner, which has also been modified to generate non-optimal plans, and an adaptation of Keyder&Geffner's compilation for NET-BENEFIT problems. Results show that our approach obtains the overall best scores for all levels of oversubscription and utility profiles. Considering the winners for each domain, our RPLD approach obtains the highest score in 42 out of the 84 configurations of domain, utility profile and oversubscription degree, which raises to 45 out of 84 in the case of utility. In contrast, the remaining approaches obtain better results than our approach in a maximum of 15 configurations. Testing on smaller problems, where it is easier to compute optimal plans, our approach finds plans that are very close to the optimal ones in most cases.

The time to compute the distances depends on the number of goals, as $|goals|^2$ relaxed plans have to be created, but also on the domain. Differences of up to one order of magnitude can be observed among domains having similar number of goals. Also, the structure of problems is relevant, with similar problems of the same domain having huge differences on computation time. A simple improvement to our approach would be to compute distances as needed, instead of pre-computing them at start. This would increase our performance in domains with a large number of goals, where most of the time is spent in computing distances which will not be used in the selection process. It will also allow us to tackle domains with hundreds of goals like *Visitall*. Reusing part of the effort done to create the relaxed plans instead of starting every time from scratch could also reduce the time spent computing distances.

Once distances are computed, selecting goals is done by using a DFBnB algorithm driven by utility. This makes selection very fast and also reduces the number of selected goals compared to a less greedy search. The experimental results show that the generated set seems to be still oversubscribed in many cases. From the results obtained from RPLD_{first}, our non-incremental approach, the selection process is slightly more accurate when goals have the same utility. In such setting, distances are used both to estimate how many goals can be selected under the cost-bound, and also which goals seem to be closer in terms of cost; a measure of how interrelated they are. Selecting those interrelated goals produces good results and increases the quality of the first plan found. In contrast, when goals have different utilities, distances are only used to select the highest-utility goals. This is done until we estimate the cost bound has been reached, ignoring lower utility goals that seem to be closer to the already selected ones. This seems to ignore dependencies among goals, producing slightly worse first plans, which are nevertheless improved thanks to the incremental behavior. In the future we want to explore ways to bias the search taking into account also the cost. When a high utility goal is selected, lower utility goals which seem to be close to it should have chances to be selected too.

In some problems the algorithms that select goals (SELECT-GOALS-AND-PLAN and ENFORCED-SELECT-GOALS-AND-PLAN) are able to explore all the search space and finish before the timeout. A possible improvement in those cases is to start again, revisiting discarded sets of goals and increasing the time alloted to INVOKE-PLANNER to find a plan for them.

We also plan to test our approach in problems having both hard and soft-goals. The extension is quite straightforward by forcing all hard goals to be included in the selected goals set and by making sure that none of them is removed during backtracking if a plan is not found. We also want to tackle NET-BENEFIT problems and in the long term temporal domains or domains with several limiting resources.

Using other heuristics, or even a combination of them, to compute the distances is an alternative we also want to explore in the future. An easy way to improve the results is to follow the current trends in planning and use a portfolio instead of a single planner once goals are selected. Actually, the combination of CBP2 and BFWS seems to be a good starting point.

A totally different approach we want to explore is to use techniques similar to those of the IBACOP [44] planner; trying to predict if the planner will find a solution for a given set of goals. This can be done once the goals have been selected, not even trying to search for a plan if the prediction says we will not find it, or we can take into account the prediction when selecting goals in order to add or not a new goal to the selected ones.

Finally, as we are performing several searches for a single problem, we want to exploit this information to refine the values of the computed distances. For instance, we would like to compare their estimated values with the ones found in intermediate plans or to adjust the plan search time for each domain using this information. Also, inspired by the idea of the goal agenda [45], we could bias the planner to follow the order in which goals have been selected.

Acknowledgements

This work has been partially funded by the European Union ECHORD++ project (FP7-ICT-601116), by FEDER/Ministerio de Ciencia, Innovación y Universidades - Agencia Estatal de Investigación TIN2017-88476-C2-2-R, RTC-2016-5407-4 and RTI2018-099522-B-C43 projects and the ESA GOTCHA project (4000117648/16/NL/GLC/fk). We would like to thank Michael Katz, Daniel Muller and Nir Lipovetzky for supporting us in using their planners and the reviewers for their helpful comments.

References

[1] D. Smith, Choosing objectives in over-subscription planning, in: Proceedings of the 14th International Conference on Automated Planning and Scheduling, 2004, pp. 393–401.

[2] C. Domshlak, V. Mirkis, Deterministic Oversubscription Planning as Heuristic Search : Abstractions and Reformulations, Journal of Artificial Intelligence Research 52 (2015) 97–169. doi:http://dx.doi.org/10.1613/jair.4443.

[3] M. Katz, V. Mirkis, In Search of tractability for partial satisfaction planning, in: Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16), 2016, pp. 3154–3160.

[4] M. Katz, E. Keyder, D. Winterer, F. Pommerening, Oversubscription Planning as Classical Planning with Multiple Cost Functions, in: International Conference on Automated Planning and Scheduling, 2019, pp. 237–245.
URL https://aaai.org/ojs/index.php/ICAPS/article/view/3482/3350

[5] D. Muller, E. Karpas, Value Driven Landmarks for Oversubscription Planning, in: International Conference on Automated Planning and Scheduling, Delft, The Netherlands, 2018, pp. 171–179.

[6] J. Ocón, J. M. Delfa, T. De La Rosa Turbides, A. Garcia-Olaya, Y. Escudero Martín, In-orbit autonomous assembly of large structures and habitats for planetary explorations using planning and scheduling techniques, in: Proceedings of the International Astronautical Congress, IAC, Vol. 4, 2017, pp. 2697–2703.

[7] J. García, Á. Torralba, J. E. Flórez, D. Borrajo, C. Linares López, A. Garcia-Olaya, TIMIPLAN: A Tool for Transportation Tasks, Springer International Publishing, Cham, 2016. doi:10.1007/978-3-319-25808-9.
URL http://link.springer.com/10.1007/978-3-319-25808-9

[8] J. Hoffmann, The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables, Journal of Artificial Intelligence Research 20 (2003) 291–341.

[9] M. Fox, D. Long, PDDL2.1: an Extension to PDDL for Expressing Temporal Planning Domains, Journal of Artificial Intelligence Research (JAIR) 20 (1) (2003) 61–124.

[10] A. García-Olaya, T. de la Rosa, D. Borrajo, Using the Relaxed Plan Heuristic to Select Goals in Oversubscription Planning Problems, in: J. Lozano, J. Gómez, J. Moreno (Eds.), Advances in Artificial Intelligence, Vol. 7023 of Lecture Notes in Computer Science, Springer Berlin-Heidelberg, 2011, pp. 183–192. doi:10.1007/978-3-642-25274-7_19.

[11] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, D. Wilkins, PDDL - The Planning Domain Definition Language, Tech. Rep. CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control (1998).
[12] A. E. Gerevini, P. Haslum, D. Long, A. Saetti, Y. Dimopoulos, Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners, Artificial Intelligence 173 (5-6) (2009) 619–68. doi:10.1016/j.artint.2008.10.012.
[13] M. B. Do, J. Benton, M. van den Briel, S. Kambhampati, Planning with goal utility dependencies, in: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07), Hyderabad, India, 2007, pp. 1872–1878.
[14] A. Gerevini, A. Saetti, I. Serina, An approach to efficient planning with numerical fluents and multi-criteria plan quality, Artificial Intelligence 172(8-9) (2008) 899–944.
[15] J. Hoffmann, H. Kautz, C. Gomes, B. Selman, SAT encodings of state-space reachability problems in numeric domains, in: M. Veloso (Ed.), Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07), Morgan Kaufmann, Hyderabad, India, 2007, pp. 1918–1923.
[16] H. Nakhost, J. Hoffmann, M. Mueller, Resource-constrained planning: A Monte Carlo random walk approach, in: International Conference on Automated Planning and Scheduling (ICAPS-12), Sao Paulo, Brazil, 2012, pp. 181–189.
[17] J. A. Baier, S. A. McIlraith, Planning with preferences, AI Magazine 29 (4) (2008) 25–36.
[18] M. van den Briel, R. Sanchez, M. B. Do, S. Kambhampati, Effective approaches for partial satisfaction (over-subscription) planning, in: Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-04), 2004, pp. 562–569.
[19] J. Benton, M. Do, S. Kambhampati, Anytime heuristic search for partial satisfaction planning, Artificial Intelligence 173 (2009) 562–592.
[20] S. Edelkamp, S. Jabbar, Mips-xxl: Featuring external shortest path search for sequential optimal plans and external branch-and-bound for optimal net benefit, in: Proc. 2008 International Planning Competition, Sydney, Australia, 2008.
[21] S. Edelkamp, P. Kissmann, Gamer: Bridging planning and general game playing with symbolic search, in: Proc. 2008 International Planning Competition, Sydney, Australia, 2008.
[22] P. Haslum, Additive and reversed relaxed reachability heuristics revisited, in: Proc. 2008 International Planning Competition, Sydney, Australia, 2008.
[23] R. Sanchez-Nigenda, S. Kambhampati, Planning graph heuristics for selecting objectives in over-subscription planning problems, in: Proceedings of the 15th Intl. Conf. on Automated Planning and Scheduling (ICAPS-05), 2005, pp. 192–201.
[24] M. Helmert, C. Domshlak, Landmarks, Critical Paths and Abstractions: What's the Difference Anyway?, in: Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS), 2009, pp. 162–169.
[25] J. Benton, A. Coles, A. Coles, Temporal Planning with Preferences and Time-Dependent Continuous Costs, in: Proceedings of the 22nd International Conference on Automated Planning and Scheduling, 2012, pp. 2–10.
[26] J. A. Baier, F. Bacchus, S. A. McIlraith, A heuristic search approach to planning with temporally extended preferences, Artificial Intelligence 173 (5-6) (2009) 593–618.
[27] E. Keyder, H. Geffner, Soft goals can be compiled away, Journal of Artificial Intelligence Research 36 (2009) 547–556.
[28] S. Ritcher, M. Westphal, The lama planner: Guiding cost-based anytime planning with landmarks, Journal of Artificial Intelligence Research 39 (2010) 127–177.
[29] M. Helmert, P. Haslum, J. Hoffmann, R. Nissim, Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces, J. ACM 61 (3). doi:10.1145/2559951.
URL https://doi.org/10.1145/2559951
[30] M. Katz, E. Keyder, A* Search and Bound-Sensitive Heuristics for Oversubscription Planning, in: Proceedings of the 11th Workshop on Heuristics and Search for Domain-independent Planning (HSDIP), 2019, pp. 81–88.
[31] W. Zhang, Depth-first branch-and-bound versus local search: A case study, in: AAAI/IAAI, 2000, pp. 930–935.
[32] J. Hoffman, B. Nebel, The FF planning system: Fast plan generation through heuristic search, Journal of Artificial Intelligence Research 14 (2001) 253–302.
[33] T. de la Rosa, A. García-Olaya, D. Borrajo, A case-based approach to heuristic planning, Applied Intelligence 39 (1) (2013) 184–201. doi:10.1007/s10489-012-0404-6.
[34] R. Howey, D. Long, M. Fox, VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL, in: The Sixteenth IEEE International Conference on Tools with Artificial Intelligence (ICTAI-04), Boca Raton (Florida), United States, 2004, pp. 294–301.
[35] A. Torralba, F. Pommerening, Planner abstracts for the classical tracks in the international planning competition 2018, Tech. rep. (2018).

[36] N. Lipovetzky, H. Geffner, Best-first width search: Exploration and exploitation in classical planning, in: AAAI Conference on Artificial Intelligence, 2017, pp. 3590–3596.
URL https://www.aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14862/14161

[37] M. Vallati, L. Chrpa, T. L. McCluskey, The 2014 International Planning Competition Description of Participating Planners. Deterministic Track, Tech. rep. (2014).

[38] A. García-Olaya, S. Jiménez, C. Linares López, The 2011 international planning competition, Tech. rep. (June 2011).
URL http://hdl.handle.net/10016/11710

[39] N. Lipovetzky, H. Geffner, Width and serialization of classical planning problems, in: Frontiers in Artificial Intelligence and Applications, Vol. 242, IOS Press, 2012, pp. 540–545. doi:10.3233/978-1-61499-098-7-540.

[40] R. Fuentetaja, D. Borrajo, C. Linares, A look-ahead B & B search for cost-based planning, in: Proceedings of the Thirteenth Conference of the Spanish Association for Artificial Intelligence (CAEPIA-09), Sevilla, Spain, 2009, pp. 105–114.

[41] I. Refanidis, I. Vlahavas, Multiobjective heuristic state-space planning, Artificial Intelligence 145 (1-2) (2003) 1–32. doi:10.1016/S0004-3702(02)00371-5.

[42] C. Linares López, S. Jiménez Celorrio, Á. García Olaya, The deterministic part of the seventh International Planning Competition, Artificial Intelligence 223 (2015) 82–119. doi:10.1016/j.artint.2015.01.004.

[43] M. Roberts, A. Howe, Learning from planner performance, Artificial Intelligence 173 (5-6) (2009) 536–561. doi:10.1016/j.artint.2008.11.009.

[44] I. Cenamor, T. de la Rosa, F. Fernández, The ibacop planning system: Instance-based configured portfolios, Journal of Artificial Intelligence Research (JAIR) 56 (2016) 657–691. doi:10.1613/jair.5080.

[45] J. Koehler, J. Hoffmann, On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm, Journal of Artificial Intelligence Research 12 (2000) 338–386.