Using Random Sampling Trees for Automated Planning

Vidal Alcázar
Computer Science Department
Universidad Carlos III de Madrid (Spain)
E-mail: valcazar@inf.uc3m.es

Susana Fernández
Computer Science Department
Universidad Carlos III de Madrid (Spain)
E-mail: sfarregu@inf.uc3m.es

Daniel Borrajo
Computer Science Department
Universidad Carlos III de Madrid (Spain)
E-mail: dborrajo@ia.uc3m.es

Manuela Veloso
Computer Science Department
Carnegie Mellon University
E-mail: mmv@cs.cmu.edu

Rapidly-exploring random trees (RRTs) are data structures and search algorithms designed to be used in continuous path planning problems. They are one of the most successful state-of-the-art techniques in motion planning, as they offer a great degree of flexibility and reliability. However, their use in other fields in which search is a commonly used approach has not been thoroughly analyzed. In this work we propose the use of RRTs as a search algorithm for automated planning. We analyze the advantages and disadvantages that this approach has over previously used search algorithms and the challenges of adapting RRTs for implicit and discrete search spaces.

Keywords: automated planning, heuristic search, random sampling

1. Introduction

Currently most of the state-of-the-art planners are based on the heuristic forward search paradigm first employed by HSP [5]. While this represented a huge leap in performance compared to previous approaches, this kind of planners also suffers from some shortcomings. In particular, certain characteristics of the search space of planning problems hinder their performance. Large plateaus for h values, local minima in the search space and areas in which the heuristic function is misleading represent the main challenges for these planners. Furthermore, most successful planners use techniques that increase the greediness of the search process, which often exacerbates this problem. A couple of examples of these approaches are pruning techniques like helpful actions introduced by FF [23] and greedy search algorithms like Enforced Hill Climbing (EHC) used by FF and greedy best-first search used by HSP and Fast Downward [21].

Motion planning is an area closely related to automated planning. Problems in motion planning consist on finding a collision-free path that connects an initial configuration of geometric bodies to a final goal configuration. Some examples of motion planning problems are robotics [28,11], animated simulations [12], drug design [13] and manufacturing [15] to name a few. A broad range of techniques have been used in this area, although the current trend is to use algorithms that randomly sample the search space due to their reliability, simplicity and consistent behavior. Probabilistic roadmaps (PRMs) [25] and Rapidly-exploring Random Trees (RRTs) [29] are the most representative techniques based on this approach.

Algorithms based on random sampling have two main uses: multi-query path planning, in which several problems with different initial and goal configurations must be solved in the same search space, and single-query path planning, in which there is only a single problem to be solved for a given search space. In the case of single-query path planning, RRT-Connect [27], a variation of the traditional RRTs used in multi-query path planning, is one of the most widely used algorithms. RRT-Connect builds a tree from the initial and the goal configurations by iteratively extending towards sampled points while trying to join the newly created branches with the goal or with a node belong-

AI Communications
ISSN 0921-7126, IOS Press.

ing to the opposite tree. This keeps a nice balance between exploitation and exploration and it is often more reliable than previous methods like potential fields, which tend to get stuck in local *minima*.

Single-query motion planning and satisficing planning have many points in common. However, bringing techniques from one area to the other is not straightforward. The main difference between the two areas is the defining characteristics of the search space. In motion planning, the original search space of these problems is an euclidean explicit continuous space, whereas in automated planning the search space is a multi-dimensional implicit discrete space composed of the states reachable from the initial state and the goal states. This has lead to both areas being developed without much interaction despite the potential benefits of an exchange of knowledge between the two communities.

In this work, we try to bridge the gap between the two areas by proposing the use of an RRT in automated planning. The motivation is that RRTs may be able to overcome some of the shortcomings that forward search planners have while keeping most of their good properties. This paper builds on previous work by the same authors [2], mainly extending the sampling methods and presenting a much more extensive experimentation; the major contributions of this paper are the following:

- We describe how to implement a search algorithm for domain-independent planning based on RRTs.
- We propose a general and efficient way of employing domain-independent reachability heuristics for their use as the distance estimator in the RRT.
- We present a method based on constraint satisfaction to sample implicit search spaces at random.
- We perform experimentation over a broad set of domains analyzing the impact of the different parameters that characterize the algorithm.

This document is organized as follows: first, some background and an analysis of previous works will be given; next, the advantages of using RRTs in automated planning will be presented as well as a description of how to overcome some problems regarding their implementation; later, some experimentation will be done to back up our claims; and, finally, some conclusions and future lines of research will be added.

2. Background

In this section we will present both automated planning and RRTs. Regarding RRTs, this includes both the original definition as a data structure and its subsequent evolution as a single-query search algorithm in motion planning.

2.1. Automated Planning

A propositional formalization of a planning task is defined as a tuple $P = (S, A, I, G)$, where S is a set of atomic propositions (also known as *facts*), A is the set of grounded actions derived from the operators of the domain, $I \subseteq S$ is the initial state, $G \subseteq S$ the set of goal propositions. We also define $c(a)$ as the cost of applying action $a \in A$ in any state $s \subseteq S$. Each action $a \in A$ is defined as a triple *(pre(a), add(a), del(a))* (preconditions, add effects and delete effects) where *pre(a), add(a), del(a)* $\subseteq S$.

Finding a solution to a planning problem P consists of generating a sequence of actions $(a_1, a_2, \ldots, a_n)$ where $a_i \in A$. The solution plan is related to a sequence of states $(s_0, s_1, s_2, \ldots, s_n)$ such that $s_i \subseteq S$, $s_0 = I$, $G \subseteq s_n$ and s_i results from executing the action $a_i \in A$ in the state s_{i-1}, $\forall i = 1..n$. The cost of a plan is defined by $\sum_{i=1}^{n} c(a_i)$.

2.2. Rapidly-exploring Random Trees

RRTs [29] were proposed as both a sampling algorithm and a data structure designed to allow fast searches in high-dimensional spaces in motion planning. RRTs are progressively built towards unexplored regions of the space from an initial configuration. Configurations describe the position, orientation and velocity of the movable objects in motion planning and are equivalent to states in other search applications.

At start, the algorithm creates a tree containing the initial configuration. At every step, a random q_{rand} configuration is chosen from all the configuration space and for that configuration the nearest configuration already in the tree q_{near} is computed. For this a definition of distance is required (in motion planning the euclidean distance is usually chosen as the distance measure). When the nearest configuration is found, a local planner tries to join q_{near} with q_{rand} with a limit distance ε. If q_{rand} was reached, it is added to the tree and connected with an edge to q_{near}. If q_{rand} was not reached, then the configuration q_{new} obtained at the end

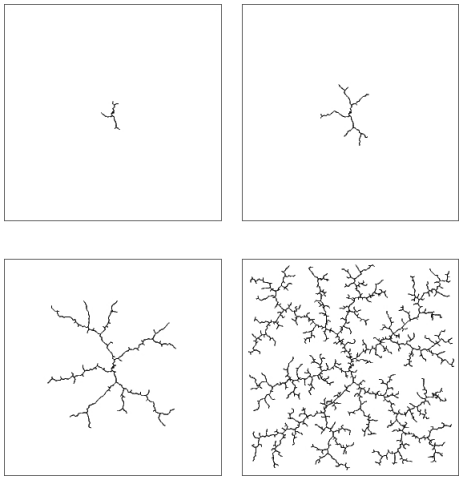

Fig. 1. Progressive construction of an RRT.

Algorithm 1: Description of the building process of an RRT.

Data: Search space S, initial configuration q_{init}, limit ε, ending criteria *end*
Result: RRT *tree*
begin
  $tree \longleftarrow q_{init}$
  while $\neg$ *end* **do**
    $q_{rand} \longleftarrow sampleSpace(S)$
    $q_{near} \longleftarrow nearest(tree, q_{rand}, S)$
    $q_{new} \longleftarrow join(q_{near}, q_{rand}, \varepsilon, S)$
    if reachable(q_{new}) **then**
      $addConfiguration(tree, q_{near}, q_{new})$
  return *tree*
end

of the local search is added to the tree in the same way as long as there was no collision with an obstacle during search. In the search literature, the term *local search* refers to search algorithms that do not keep track of all the states that they have visited. The most representative algorithm of this kind is Hill Climbing, although many others exist. Here, though, whenever we use the term *local search* we mean the process of solving the subproblem needed to create a new branch of the tree. This operation is called the *Extend* step, illustrated in Figure 2. This process is repeated until some criterion is met, like a limit on the size of the tree. Algorithm 1 gives an outline of the process.

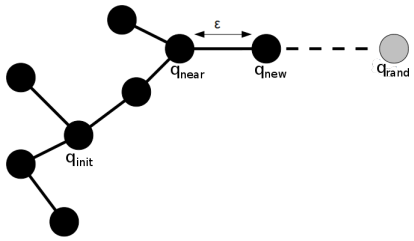

Fig. 2. Extend phase of an RRT.

Once the RRT has been built, multiple queries can be issued. For each query, the nearest configurations (node) of the tree to both the initial and the goal configurations of the query are found. Then, the initial and final configurations are joined to the tree to those nearest configurations using the local planner and a path is retrieved by tracing back edges through the tree structure.

The key advantage of RRTs is that they are intrinsically biased towards regions with a low density of configurations in their building process. This can be explained by looking at the Voronoi diagram at every step of the building process. The Voronoi diagram is composed by Voronoi regions; Voronoi regions associated to a given node q of the tree are areas such that every point in the area is closer to q than to any other node q' of the RRT. The Voronoi region of a given node is larger when the area around that node has not been explored. This way, the probability of a configuration being sampled in an unexplored region is higher as larger Voronoi regions will be more likely to contain the sampled configuration [3]. This has the advantage of naturally guiding the tree by extending nodes at the edge of unexplored regions with a higher probability while just performing uniform sampling. Besides, the characteristics of the Voronoi diagram are indicative of the adequateness of the tree. For example, a tree whose Voronoi diagram is formed by regions of similar size covers uniformly the search space, whereas large disparities in the size of the regions mean that the tree may have left big areas of the search space unexplored. Apart from this, another notable characteristic is that RRTs are probabilistically complete, as they will cover the whole search space if the number of sampled configurations tends to infinity. Figure 3 shows the Voronoi diagrams of the RRTs previously shown in Figure 1.

2.3. RRT-Connect

After corroborating how successful RRTs were for multi-query motion planning problems, researchers in motion planning realized that using multi-query RRTs was often more efficient and robust than using spe-

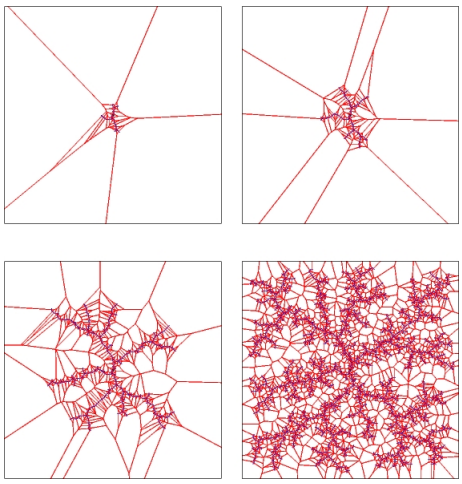

Fig. 3. Voronoi Diagram of an RRT.

cific single-query motion planning algorithms even for a single query. Motivated by this fact and aiming to develop a more suitable RRT-like algorithm for the single-query case, a variation for single-query problems called RRT-Connect was proposed [27]. The modifications introduced were the following:

- Two trees are grown at the same time by alternately expanding them. The initial configuration of the trees are the initial and the goal configuration respectively.
- The trees are expanded towards randomly sampled configurations with a probability p, and towards the nearest node of the opposite tree with a probability $1-p$. Hence, with a probability $1-p$ the closest distance among the $m \times n$ distances between nodes from both trees is found and the node of the expanding tree is expanded towards the node of the non-expanding tree. With a probability p a random configuration is sampled and both trees are expanded as usual.
- The *Expand* phase is repeated several times until an obstacle is found. The resulting nodes from the local searches limited by ε are added to the tree. This is called the *Connect* phase.

Growing the trees from the initial and the goal configurations and towards the opposite tree gives the algorithm its characteristic single-query behavior. The *Connect* phase was added after empirically testing that the additional greediness that it introduced improved the performance in many cases. A common modification is also extending the tree towards the opposite tree after every q_{new} is added when sampling randomly, extending from that q_{new} configuration towards the opposite tree. This helps in cases in which both trees are stuck in regions of the search space that are close as per the distance measure, but in which local searches consistently fail due to obstacles.

3. Previous Work

Although RRTs have not been frequently used in areas other than motion planning, there is previous work in which they have been employed for problems relevant to automated planning. In particular, an adaptation of RRTs for discrete search spaces and a planning search algorithm similar to RRT-Connect have been proposed.

3.1. RRTs in Discrete Search Spaces

Despite RRTs being designed for continuous search spaces, researchers from other areas proposed their implementation for search problems in discrete search spaces [33]. The main motivation of this work was adapting the RRTs to grid worlds and similar classical search problems. Its authors analyzed the main challenges of adapting RRTs, in particular the need of defining an alternative measure of distance to find the nearest node of the tree to a sampled state and the issues related to adapting the local planner that must substitute the *Expand* phase of the traditional RRTs. The proposed alternative to the widely used euclidean distance was an "ad-hoc" heuristic estimation of the cost of reaching the sampled state from a given node of the tree. As for the local planner, the limit ε that was used to limit the reach of the *Expand* phase was substituted by a limit on the number of nodes expanded by the local planner. In this case, once the limit ε was reached the node in the local search with the best heuristic estimate towards the sampled space was chosen, and either only that node or all the nodes on the path leading to it were added to the tree.

While the approach was successful for the proposed problems, there are two main problems that make it impossible to adapt it to automated planning in a straightforward way. First, the search spaces in the experimentation they performed are explicit, whereas in automated planning the search space is implicit. This adds an additional complexity to the sampling process that must be dealt with in some way. Second, the heuristics for both the distance estimation and the local planners were designed individually for every particular problem and thus are not useful in the more general case of automated planning.

3.2. RRT-Plan

Directly related to automated planning, the planner RRT-Plan [9] was proposed as a stochastic planning algorithm inspired by RRTs. In this case, the EHC search phase of FF [23], a deterministic propositional planner, was used as the local planner. The building process of the RRT was similar to the one proposed for discrete search spaces; that is, to impose a limit on the number of nodes as ε and add the expanded node closest to the sampled state to the tree. In this case, though, the tree was built only from the initial state due to the difficulty of performing regression in automated planning.

The key aspects in this work are two: the computation of the distance necessary to find the nearest node to the sampled or the goal state, and sampling in an implicit search space. In RRTs one of the most critical points is the computation of the nearest node in every *Expand* step, which may become prohibitively expensive as the size of the tree grows with the search. The most frequently used distance estimations in automated planning are the heuristics based on the reachability analysis in the relaxed problem employed by forward search planners, like the h^{add} heuristic used by HSP [5] or the relaxed plan heuristic introduced by FF [23]. The problem with these heuristics is that, although computable in polynomial time, they are usually still relatively expensive to compute, to the point that they usually constitute the bottleneck in satisficing planning. To avoid recomputing the reachability analysis from every node in the tree, every time a new local search towards a state is done, the authors propose caching the cost of achieving every goal proposition from a node whenever that node is added to the tree. This way, by adding the costs of the propositions that form the sampled state, h^{add} can be obtained without needing to perform a reachability analysis.

Regarding sampling, RRT-Plan does not sample the search state uniformly. Instead, it chooses a subset of propositions $s \subseteq S$ from the goal set such that $s \subseteq G$ and uses s as q_{rand}. This is due to the fact that, although sampling a state by choosing random propositions in automated planning is trivial, determining whether a given sampled state belongs to the search space is PSPACE-complete, as it is as hard, in terms of computational complexity, as solving the original problem itself. This problem is avoided by the sampling technique of RRT-Plan in the sense that, if the problem is solvable, G must be reachable. Hence, any of its possible subsets is also reachable. In addition, RRT-Plan performs goal locking; i.e., when a goal proposition p that was part of a given sampled state $s \subseteq G \mid p \in s$ is achieved, any subsequent searches from the added q_{new} node and its children nodes are not allowed to delete p.

Whereas RRT-Plan effectively addresses the problem of sampling states in implicit search spaces, this kind of sampling limits most of the advantages RRTs have to offer. By choosing subsets of the goal set instead of sampling more uniformly the search space, the RRT does not tend to expand towards unexplored regions. Thus, it loses the implicit balance between exploration and exploitation during search that characterizes them. In fact, by choosing this method, RRT-Plan actually benefits from random guesses over the order of the goals instead of exploiting the characteristics of RRTs. As a side note, this could actually be seen as a method similar to the goal agenda [26], albeit with random selection of subsets and the possibility in this case to recover from wrong orderings.

4. Advantages of RRTs in Automated Planning

As mentioned in the introduction, during the last years there has been a big improvement in performance in the area of propositional planning. The most representative approach among those that contributed to this improvement is the use of forward heuristic search algorithms along with reachability heuristics and other associated techniques. However, heuristic search planners suffer from several problems that detract from their performance. These problems are related mainly to the characteristics of the search space that most common planning domains have. Search spaces in automated planning tend to have big plateaus in terms of the h value. The high number of transpositions and the high branching factor that are characteristic of many domains aggravate this fact. Heuristic search planners that use best-first search algorithms are particularly affected by this, as they consider total orders when generating new nodes and are mostly unable to detect symmetries and transpositions during search. It has been shown that techniques that increase the greediness of the search algorithm, like helpful actions from FF and look-ahead states from YAHSP [40], tend to partially alleviate these problems. However, even though reachability heuristics have proved to be relatively reliable for the most part, in some cases they can also be quite misguided. This increased greediness can be disadvantageous at times.

Figure 4 shows a typical example of a best-first search algorithm getting stuck in an *h plateau* due

to inaccuracies in the heuristic. In this example, the euclidean distance used as heuristic ignores the obstacles. Because of this, the search advances forward until the obstacle is found. Hence, the search algorithm must explore the whole striped area before it can continue advancing towards the goal. This highlights the imbalance between exploitation and exploration these approaches have. This problem has been previously studied, and several methods that tried to minimize its negative impact on search have been proposed [30,38]. However, this imbalance still remains as one of the main shortcomings of best-first search algorithms. To partially address this issue, we consider expanding nodes towards randomly sampled states so a more diverse exploration of the search space is done. In this example, a bias that would make the search advance towards q_{rand} could avoid the basin flooding phenomenon that greedier approaches suffer from.

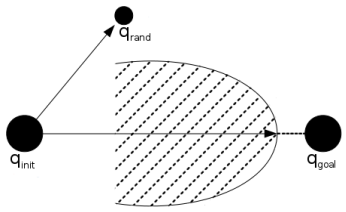

Fig. 4. Simple example of a best-first search algorithm greedily exploring an *h plateau* due to the heuristic ignoring the obstacles. Advancing towards some randomly sampled state like q_{rand} can alleviate this problem.

RRTs incrementally grow a tree towards both randomly sampled states and the goal. Therefore, they are less likely to suffer from the same problem as best-first search algorithms. The main advantages that they have over other algorithms in automated planning are the following:

- They keep a better balance between exploration and exploitation during search.
- Local searches minimize exploring plateaus, as the maximum size of the search is bounded.
- They use considerably less memory, as only a relatively sparse tree must be kept in memory.
- They can employ a broad range of techniques during local searches.

All in all, the alternation of random sampling and search towards the goal that single-query RRTs have is their most characterizing aspect. Thanks to this, they do not commit to a specific area of the search space and hence they tend to recover better than best-first search from falling into local *minima*. In terms of memory, the worst case is the same for best-first search algorithms and RRTs. However, RRTs must keep in memory only the tree and the nodes from the current local search. Trees are typically much sparser than the area explored by best-first algorithms, which makes them much more memory efficient on average. Memory is usually not a problem in satisficing planning because of the time needed for the heuristic computation, although in some instances it can be an important limiting factor.

5. Implementing the RRT

Due to the differences in the search space, adapting RRTs from motion planning to automated planning is not trivial. In this work we propose an implementation partially based on RRTs for discrete search spaces and RRT-Plan with some changes critical to their performance.

5.1. Sampling

The main reason why RRTs have not been considered for automated planning is the difficulty of properly sampling the search space. The difficulty arises from the fact that choosing propositions from S at random may lead to generating spurious states. Spurious states were initially defined as states that are not reachable from I [5], although other definitions exist [44]. In fact, there may be the case that a state reachable from I is unreachable from G in regression (such a state would be a dead end in progression), so a more general definition of spurious state is a state that cannot belong to any solution path of the problem. RRT-Plan circumvented this by substituting uniform sampling with subsets of goal propositions. However, this negates some of the advantages that RRTs have, as explained before.

Checking whether a state is spurious or not is as hard as solving the problem itself [10], so an approximative approach must be used instead. Here we propose the use of state invariants as constraints to reduce the chances of obtaining a spurious state when uniformly sampling the search space. In particular, we propose the use of mutually exclusivity between propositions [5] (already employed by evolutionary planners like $D_A E_X$ [4], which decomposes the problem using sampling techniques) and *"exactly-1"* invariant groups.

5.1.1. Mutually Exclusivity Between Propositions

First, we will define mutually exclusivity between propositions.

Definition 1. *(Mutex) A set of propositions* $M = \{p_1, \ldots, p_m\} \mid p_i \in S$ *is mutually exclusive of size m (mutex of size m) if there is no state* $s \subseteq S$ *that may belong to a solution path such that* $M \subseteq s$.

A common example of mutex is the location of an object in a transportation problem. For instance, in the *Logistics* domain, the propositions *(at truck1 loc1)* and *(at truck1 loc2)* are mutex, as a truck cannot be in two places at the same time. The previous example is a mutex of size 2 (also known as a *binary mutex*), but mutexes of greater size are also possible. An example of mutex of size 3 in the *Blocksworld* domain is a tower of three blocks that forms a cycle, *i.e.* M=$\{$*(on A B), (on B C), (on C A)*$\}$. An important remark is that no subset of two elements of M is a mutex, which means that mutexes of greater size cannot be built parting from sets of mutexes of smaller size.

The most common method to find mutexes is the h^m heuristic [5]. h^m performs a reachability analysis in P^m [20], a semi-relaxed version of the original problem in which the atoms are actually sets of m propositions. This way, if the value of an m atom is infinite (which means that it is unreachable in P^m), then we can conclude that the unreachable atom is a mutex of size m. However, the time needed to compute h^m grows exponentially with m, so in most cases it is impractical to compute mutexes of size greater than two. In fact, increasing the size of m is a clear case of diminishing returns, as the bigger the size of the mutex the fewer spurious states will tend to contain it, and thus the less useful it will probably be to detect spurious states.

Another method for finding binary mutexes is the monotonicity analysis generally employed to generate a multi-valued formalization of the problem [22]. This monotonicity analysis ensures that the number of propositions true at the same time that belong to a set $I_g = \{p_0, p_1, \ldots, p_n\}$ can never increase. Hence, if the number of propositions of I_g true in the initial state is one (formally, $|I_g \cap I| = 1$), then all possible pairs of propositions of I_g are binary mutexes ($\forall P_m = \{p_i, p_j\} | p_i, p_j \in I_g \wedge p_i \neq p_j$, P_m is a binary mutex). For example, in a *Logistics* problem in which a truck can move between three different locations *loc1*, *loc2* and *loc3*, the number of propositions true in the set M_a=$\{$*(at truck1 loc1), (at truck1 loc2), (at truck1 loc3)*$\}$ can never increase, as whenever a truck moves to a location it must leave the location of origin. Therefore, we can infer that $\{$*(at truck1 loc1), (at truck1 loc2)*$\}$, $\{$*(at truck1 loc1), (at truck1 loc3)*$\}$ and $\{$*(at truck1 loc2), (at truck1 loc3)*$\}$ are binary mutexes.

In terms of efficiency, computing the monotonicity analysis is in most cases much more efficient that computing h^2, as it works exclusively with the domain definition and the initial state. However, the set of mutexes found by the monotonicity analysis is a strict subset of the set of mutexes found by h^2. For this reason, in this work we will use h^2 to compute mutexes.

Another advantage of h^2 is that it can be computed backwards too in order to find extra mutexes [19]. This is done by reversing the domain as proposed by Massey [32] and performing a backwards reachability analysis in P^2 after deducing which propositions are false in $s_\star$ with mutexes computed forward. Mutexes computed backward, as opposed to regular mutexes, are violated in spurious states reached forward during search.

5.1.2. "exactly-1" *Invariant Groups*

Mutexes alone may not suffice to prune spurious states [1]. For instance, in the *Floortile* domain, cells can be either clear, painted or occupied by some robot. One can sample a state $s \subseteq S$ by choosing propositions from S at random and discarding propositions mutex with the chosen ones, like $D_A E_X$ does. By doing this it is possible to obtain a state s in which there are fewer occupied cells than robots. In such a state at least one robot would not be at any location at all because all cells are either clear or occupied by another robot, and hence s would be spurious. Figure 5 shows a state of the *Floortile* domain in which a robot cannot be placed anywhere.

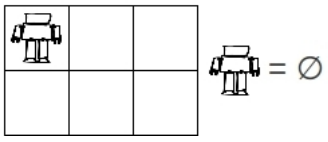

Fig. 5. Example of spurious sampled state in the *Floortile* domain. The second robot has no valid location, because all cells are either clear or occupied by another robot.

In order to avoid this, *"exactly-1"* invariant groups can be used.

Definition 2. *(*"exactly-1" *invariant group) An* "exactly-1" *invariant group is a set of propositions* $I_1 = \{p_1, \ldots, p_m\} | p_i \in S$ *such that in every non-spurious state* $s \subseteq S$ *there is exactly one proposition* $p \in I_1$ *such that* $p \in s$. *Formally,* $\forall s \subseteq S$ *such that s is non-spurious,* $|I_1 \cap s| = 1$.

By using *"exactly-1"* invariant groups one can infer an additional state invariant: if $I_g = \{p_0, p_1, \ldots, p_n\}$ is an *"exactly-1"* invariant group, then $(p_0 \vee p_1 \vee \ldots \vee p_n)$ is a state invariant of the problem. *"exactly-1"* invariant groups can be derived from any binary mutex computation method. Candidate sets of propositions are sets $I_g = \{p_0, p_1, \ldots, p_n\}$ such that all possible pairs of propositions of I_g are binary mutexes, like the invariant groups obtained from the aforementioned monotonicity analysis. For a candidate set of propositions I_g to be an *"exactly-1"* invariant group, every action $a \in A$ that adds a proposition $p \in I_g$ must also delete exactly one proposition $p' \in I_g$ and *vice versa*. Then, I_g is an *"exactly-1"* invariant group. Formally, I_g is an *"exactly-1"* invariant group if $\forall p_i, p_j \in I_g | i \neq j$ then p_i, p_j are mutex and $\forall a \in A : |add(a) \cap I_g| = |del(a) \cap I_g| = 1$.

5.1.3. Selecting Propositions

Sampling a state using state invariants as constraints is analogous to solving a Constraint Satisfaction Problem (CSP). A CSP is formally defined as a triple *CSP=(V,D,C)*, where V are the variables of the problem, D are the domains of the variables in D and C are the constraints of the problem. In this CSP the *"exactly-1"* invariant groups are the variables in V, the propositions of the *"exactly-1"* invariant groups are the domain D of the variables in V and the binary mutexes of the problem are the constraints of C. The objective is to choose a proposition $p \in S$ from every *"exactly-1"* invariant group I_1 such that it is not mutex with any other chosen proposition $p' \in S$. This ensures that the complete sampled state $s \in S$ satisfies all *"exactly-1"* invariant groups and does not violate any binary mutex.

Solving a CSP is NP-complete. Actually, for some planning instances solving the CSP that represents the sampling process may be on average very time consuming if it is done naively. In our implementation we use forward checking [18] to improve the performance of the backtracking procedure needed for solving the CSP. The order of the variables (the order in which the *"exactly-1"* invariant groups are selected to be satisfied) is static, although it may vary between different sampling processes. *"exactly-1"* invariant groups with the highest cardinality are chosen first, with ties broken randomly every time a new state is sampled. Ordering of values of variables is chosen at random. This aims to reproduce the behavior of the degree (most constraining variable) heuristic [6] while trying to obtain sampled states as diverse as possible.

5.1.4. Ensuring the Reachability of Goals

Even after using state invariants, it may happen that the goal is not reachable from the sampled state. For example, a sampled state in the *Sokoban* domain may contain a configuration of blocks such that some block cannot be moved anymore. While this sampled state may not violate any state invariant, the sampled state is a dead end unless the unmovable blocks are at a goal location, since the original goal is not reachable. To address this problem, a regular reachability analysis can be done from the sampled state. If some proposition $p \in G$ is unreachable, then the sampled state can be safely discarded. This is again an incomplete method, but in cases such as the aforementioned one it is useful to detect spurious states.

5.2. Distance Estimation

One of the most expensive steps in an RRT is finding the closest node to a sampled state. Besides, the usual distance estimation in automated planning, the heuristics derived from a reachability analysis, are also computationally costly. RRT-Plan solved this by caching the cost of achieving a goal proposition from every node of the tree and using that information to compute h^{add}, just like HSPr does [5] when searching backwards. Despite being an efficient solution, this shares the same problem as HSPr: only h^{add} can be computed using that information. h^{add} tends to greatly overestimate the cost of achieving the goal set and other heuristics of the same kind, like the FF heuristic, are on average more accurate [16]. Therefore, in our implementation, *best supporters*, that is, actions that first achieve a given proposition in the reachability analysis, are cached as proposed by Alcázar *et al.* [1]. This allows to compute not only h^{add} but also other heuristics like the FF heuristic (by tracing back the relaxed plan using the cached best supporters). The time of computing the heuristic once the best supporters are known is usually very small compared to the time needed to perform the reachability analysis - linear in the size of the relaxed plan -, so this approach allows to get more accurate (or diverse) heuristic estimates without incurring a significant overhead.

5.3. Tree Construction

RRTs can be built in several ways. The combination of the *Extend* and *Connect* phases, the possibility of greedily advancing towards the goal with a probability $1 - p$ instead of sampling with a probability p,

the way new nodes are added (only the closest node to the sampled state or all the nodes on the path to that state),... allow for a broad range of different options. In this work, we have chosen to build the tree in the following way:

- the tree is built from the initial state I only, as regression is overall not as efficient as progression in automated planning [1];
- every node in the tree contains a state, a link to its parent, a plan that leads from its parent to the state, and the cached best supporters for every proposition $q \in S$ so h^{FF} can be computed efficiently;
- ε limits the number of expanded nodes in every local search (limiting the time is another possible stopping criterion, but we preferred to use the same implementation as RTT-Plan);
- there is a probability p of advancing towards a sampled state and a probability $1-p$ of advancing towards the goal from the closest node to the original goal G. It may happen that the closest node to G was already expanded towards G in an earlier iteration and the new generated node q_{new} from that expansion is farther from the goal than G; that is, $h^{FF}(q_{new}) > h^{FF}(q_{near})$. Since planners are for the most part deterministic, it does not make sense to repeat the search - it would lead to the same q_{new} -, so in fact the node selected with a probability $1-p$ is the closest node *among those that have never served as the origin of a local search towards the goal before*;
- a single node is added to the tree after every local search (instead of all the nodes along the solution path) to try to obtain a sparse yet representative enough tree;
- when performing a local search, if a solution for the subproblem was not found, the last expanded node is added to the tree (be it when expanding towards a sampled state or the original goal G itself) to add to the tree the most promising node among those generated during the local search;
- after adding a new node q_{new} from the local search towards a sampled state, a new local search from q_{new} to q_{goal} is performed.

No *Connect* phase is performed. This is because the *Connect* phase is probably counter-productive if it is done towards sampled states - sampled states may be completely irrelevant to the solution and the main benefit obtained from them is the additional bias towards exploration anyway - and partially overlaps with the expansions towards the goal with a probability $1-p$.

This configuration is the basis of the planner presented in this paper; we called this planner Randomly-exploring Planning Tree (RPT). Algorithm 2 describes the whole process.

Algorithm 2: Search process of RPT.

Data: Search space S, limit ε, initial state q_{init}, goal q_{goal}
Result: Plan $solution$
begin
  $tree \longleftarrow q_{init}$
  while $\neg$ *goalReached()* **do**
    if *random()*$< p$ **then**
      $q_{rand} \longleftarrow$ *sampleSpace(S)*
      $q_{near} \longleftarrow$ *nearest(tree,*q_{rand}*,S)*
      $q_{new} \longleftarrow$ *join(*q_{near}, q_{rand}*,*ε*,S)*
      addNode(tree,q_{near}, q_{new}*)*
      $q_{near_{goal}} \longleftarrow q_{new}$
    else
      $q_{near_{goal}} \longleftarrow$ *nearest(tree,*q_{goal}*,S)*
    $q_{new_{goal}} \longleftarrow$ *join(*$q_{near_{goal}}, q_{goal}$*,*$\varepsilon$*,S)*
    addNode(tree,$q_{near_{goal}}, q_{new_{goal}}$*)*
  $solution \longleftarrow$ *traceBack(tree,*q_{goal}*)*
  return $solution$
end

5.4. Choice of the Local Planner

The choice of the planner used in the local search is subject to some restrictions. First, after every *Extend* phase a new node to the tree is added even if a solution for the subproblem could not be found. This means that the local planner must be able to return an executable plan also when no solution was found, which rules out some planning paradigms like partial-order planners [43] and SAT-based planners [37]. Second, the tree is built forward, so the local planner must return a forward-executable plan. Again, backward search planners like HSPr [5] and FDr [1] cannot be used for this reason. Another important point is the preprocessing time. Since multiple local searches may be done, it is desirable that the time spent by the local planner prior to search is as short as possible. For example, the use of heuristics that require a relatively long preprocessing time and depend on either the initial state or the goals, like Pattern Databases [14], are discouraged.

In this work, the Fast Downward planning system [21] was used as the local planner. It was config-

ured to use greedy best-first search with lazy evaluation as its search algorithm. The heuristic is the FF heuristic [23]. Preferred operators obtained from the FF heuristic were enabled.

6. Experimentation

In this section, we test the proposed approach against other state-of-the-art planners. The maximum available memory was set to 6GB and the time limit was 1800 seconds, as in the last International Planning Competition (IPC-2011). The selected domains were all the domains from the deterministic track of the past IPCs. In the cases in which a domain appeared in more than one competition, we selected the instances from the hardest set. The criteria are the same as the ones used by Rintanen [37]. Additionally, we used the test problems from the learning track of the 2008 and 2011 IPCs for those domains that were not used in the deterministic track. The exception is the *Sokoban* domain; the definition of the domain and the structure of the problems are significantly different from the ones of the deterministic track, so we employed both versions. We call the version from the learning track *Sokoban-learning*. In this section, we will focus on the number of problems solved (also known as coverage), so all actions will be treated as if they had unit cost.

RPT was implemented on top of Fast Downward [21]. Since RRTs are stochastic algorithms, the experiments are repeated five times and the arithmetic mean and standard deviation are reported. The computation of h^2 was implemented in Fast Downward and the mutexes obtained from the computation of h^2 forward and backward. *"exactly-1"* invariant groups were obtained from the monotonicity analysis done by the translator of Fast Downward. To further exploit the state invariants, spurious actions were pruned by disambiguating their preconditions [1]. We set a limit of 300 seconds for the h^2 computation and the disambiguation of actions.

The planners we compare against are the local planner itself described in Section 5.4, that we call FD, and the first phase of the LAMA planner [36], the winner of the IPCs held in 2008 and 2011. We used the last revision from Fast Downward's public repository for both planners.[1] In fact, LAMA is the same as FD, but also uses an additional landmark counting heuristic combined with the regular FF heuristic. Preferred operators are obtained from the landmark heuristic too and the final set of preferred operators is the union of the sets of preferred operators obtained from both heuristics. Both heuristics are combined in an alternation queue. This queue expands the best node as per the correspondent heuristic alternatively at every node expansion.

There are two critical parameters that affect the behavior of RPT in our implementation: the limit on the number of expanded nodes in the local search ε and the probability p of expanding towards a sampled state instead of towards the original goal G. In the experimentation we tried six different combinations resulting from combinations of ε and p. The values used for ε were $\varepsilon = 10000, 100000$. The values used for p were $p = 0.2, 0.5, 0.8$. By experimenting with different versions we aim to understand which are the factors that can have an impact on the performance of the algorithm.

Additionally and for the sake of completeness, we compare the aforementioned configurations of RPT against a configuration with an artificially low ε of $\varepsilon = 1000$ and $p = 0.5$ and with a version named *goals* which, instead of sampling as described in Section 5.1.3, samples by choosing random subsets of goals. This is so to compare our sampling approach with the sampling approach originally proposed by RRT-Plan. The *goals* version has $\varepsilon = 10000$ and $p = 0.5$ as parameters.

6.1. Coverage

Table 1 shows the coverage (mean and standard deviation over 5 runs) of all the evaluated planners. Results show that RPT is overall better than the base planner FD and competitive with the state-of-the-art planner LAMA. Total coverage favors all the tested configurations of RPT but the one with $\varepsilon = 1000$ and *goals* over both FD and LAMA.

A more detailed comparison on a per domain basis shows that performance depends in a significant number of cases on the domain. For instance, RPT is notably better in *Airport*, *Floortile*, *Sokoban-learning* and *Storage*, whereas FD and LAMA are better in *Barman* and *N-puzzle*. The domains in which RPT performs better are often those in which there are dead ends undetectable by reachability heuristics, which cause FD and LAMA to explore big sterile plateaus. Note that *Floortile*, *Sokoban-learning* and *Storage* have a similar structure, as achieving the closest goal proposition first often leads to a dead end; in this case the additional exploration induced by the sampling

[1] As of December 2013, revision 3288.

Planners:	FD	LAMA	10k_0.2	10k_0.5	10k_0.8	100k_0.2	100k_0.5	100k_0.8	1k_0.5	goals
Airport(50)	35	32	45±1.41	45.6±0.55	**46**±0	41.6±1.67	43.8±0.84	43±2.12	**46**±0	40.4±1.52
Barman(20)	**20**	**20**	3±0.71	3±1.58	2.4±1.14	9.8±0.84	8.2±1.1	8.2±1.1	1.4±0.89	19.8±0.45
Blocks(35)	**35**	**35**	**35**±0	**35**±0	**35**±0	**35**±0	**35**±0	**35**±0	**35**±0	**35**±0
Depot(22)	18	21	20.2±0.84	20.8±0.84	20±1.22	19.6±0.89	19.6±0.89	19.8±0.84	18.2±0.84	**22**±0
Driverlog(20)	**20**	**20**	**20**±0	**20**±0	**20**±0	**20**±0	**20**±0	**20**±0	18±0	19±0
Elevators(20)	19	**20**	**20**±0	**20**±0	**20**±0	19.8±0.45	19.8±0.45	19.8±0.45	19.4±0.89	**20**±0
Floortile(20)	7	6	**20**±0	**20**±0	**20**±0	**20**±0	**20**±0	**20**±0	19.8±0.45	**20**±0
Freecell(80)	79	79	**80**±0	**80**±0	**80**±0	**80**±0	**80**±0	**80**±0	**80**±0	77.8±1.3
Gold-miner(30)	**30**	**30**	**30**±0	**30**±0	**30**±0	**30**±0	**30**±0	**30**±0	**30**±0	**30**±0
Grid(5)	**5**	**5**	**5**±0	**5**±0	**5**±0	**5**±0	**5**±0	**5**±0	**5**±0	**5**±0
Gripper(20)	**20**	**20**	**20**±0	**20**±0	**20**±0	**20**±0	**20**±0	**20**±0	**20**±0	**20**±0
Logistics(35)	34	**35**	**35**±0	**35**±0	**35**±0	34.4±0.55	34.6±0.55	34.8±0.45	34.6±0.55	**35**±0
Matching-bw(30)	20	25	**30**±0	**30**±0	**30**±0	**30**±0	**30**±0	**30**±0	**30**±0	**30**±0
Miconic(150)	**150**	**150**	**150**±0	**150**±0	**150**±0	**150**±0	**150**±0	**150**±0	**150**±0	**150**±0
Movie(30)	**30**	**30**	**30**±0	**30**±0	**30**±0	**30**±0	**30**±0	**30**±0	**30**±0	**30**±0
Mprime(35)	**35**	**35**	**35**±0	**35**±0	**35**±0	**35**±0	**35**±0	**35**±0	**35**±0	34.8±0.45
Mystery(20)	16	**19**	**19**±0	**19**±0	**19**±0	**19**±0	**19**±0	**19**±0	**19**±0	17.2±0.45
Nomystery(20)	9	**13**	12.2±0.84	11±1	10.8±0.84	10.6±0.89	10.8±0.84	10±1	12.4±1.14	7±0.71
N-puzzle(30)	28	**30**	20.6±1.34	21.2±0.84	20.4±0.89	25.4±0.55	25.8±0.84	26±1	10.2±1.92	30±0
Openstacks(20)	**20**	**20**	19.8±0.45	**20**±0	**20**±0	19.6±0.89	19.2±0.84	17.6±1.34	**20**±0	**20**±0
Parcprinter(20)	**20**	**20**	**20**±0	**20**±0	**20**±0	**20**±0	**20**±0	**20**±0	**20**±0	**20**±0
Parking(20)	**20**	**20**	19.8±0.45	19.8±0.45	**20**±0	19.8±0.45	19.6±0.55	19±1.22	**20**±0	**20**±0
Pathways-noneg(30)	**30**	**30**	**30**±0	**30**±0	**30**±0	**30**±0	**30**±0	**30**±0	29.8±0.45	29.6±0.55
Pegsol(20)	**20**	**20**	**20**±0	**20**±0	**20**±0	19.8±0.45	**20**±0	**20**±0	**20**±0	19.4±0.55
Pipesworld-notank(50)	43	44	43.2±0.45	43.4±0.55	**44.2**±1.3	43.8±0.45	43.4±0.55	43.4±0.89	**44.2**±1.1	43.4±1.14
Pipesworld-tank(50)	39	41	40.2±0.84	40.4±0.89	40.4±1.52	41.2±0.84	39.8±1.3	39.6±1.34	**41.4**±0.89	38.4±1.67
PSR-small(50)	**50**	**50**	**50**±0	**50**±0	**50**±0	**50**±0	**50**±0	**50**±0	**50**±0	**50**±0
Rovers(40)	**40**	**40**	**40**±0	**40**±0	**40**±0	**40**±0	**40**±0	**40**±0	39.8±0.45	**40**±0
Satellite(36)	**36**	**36**	35.6±0.55	35.4±0.55	35.4±0.55	35.8±0.45	35.2±0.84	34.4±0.89	35.2±0.45	**36**±0
Scanalyzer(20)	19	**20**	19.8±0.45	**20**±0	**20**±0	19.2±0.45	**20**±0	19.8±0.45	19.8±0.45	**20**±0
Sokoban(20)	**19**	17	12.8±1.1	14.2±0.84	14.6±1.14	16.4±1.14	16.4±1.14	17.2±0.84	12±1	5.4±0.55
Sokoban-learning(30)	24	21	**30**±0	**30**±0	**30**±0	**30**±0	**30**±0	**30**±0	**30**±0	11±1.22
Spanner(30)	0	0	0±0	0±0	0±0	0±0	0±0	0±0	0±0	0±0
Storage(30)	20	19	25.8±0.84	26±1	**26.6**±1.14	24.2±1.3	24.4±1.14	25.6±0.89	25.6±0.89	19.4±0.89
Tidybot(20)	13	14	18±1	17.4±0.55	18±0	17.6±0.55	17.4±1.14	17.6±0.55	17.8±0.45	**19.2**±0.84
Tpp(30)	**30**	**30**	**30**±0	**30**±0	**30**±0	**30**±0	**30**±0	**30**±0	**30**±0	**30**±0
Transport(20)	11	17	13±0.71	13.2±1.1	14±0.71	11.8±0.84	11.2±0.84	9.6±1.67	12.8±1.3	**17.6**±1.34
Trucks-strips(30)	19	18	22±1	**23.2**±0.45	23±1	20.8±1.48	21.8±1.1	23±1	21.2±0.84	9.4±0.55
Visitall(20)	3	**20**	14.6±1.14	13.4±1.52	10.8±0.45	11.8±0.45	10.8±0.45	7.6±0.55	2.6±0.55	13.8±0.45
Woodworking(20)	**20**	**20**	**20**±0	**20**±0	**20**±0	**20**±0	**20**±0	**20**±0	**20**±0	**20**±0
Zenotravel(20)	**20**	**20**	**20**±0	**20**±0	**20**±0	**20**±0	**20**±0	**20**±0	**20**±0	**20**±0
Total	1125	1162	1174.6	**1177**	1175.6	**1177**	1175.8	1170	1146.2	1145.6

Table 1

Comparison between FD, LAMA and the different configurations of RPT. Numbers in parentheses represent the total number of problems in the domain. Mean and standard deviation over 5 runs reported.

method and the limit on the number of expanded nodes by the local search ε prove to be very beneficial.

Some domains in which RPT is outperformed by LAMA are domains in which state invariants do not suffice to avoid spurious states. *Barman* is the most notable example, because the current definition allows some unintended things to happen (such as a glass containing a drink and being clean at the same time), which reduces the number of mutexes found by invariant computation methods. Another very relevant case is *Visitall*, in which FD is able to solve only 3 problems, LAMA solves the whole set of problems and RPT is somewhere in the middle. The difference between FD and LAMA is explained by the heuristics they use: *Visitall* is a domain designed to induce plateaus when the FF heuristic is used, but it is otherwise easily solved with heuristics that employ additive schemes such as the goal-counting and the landmark-counting heuristics. However, this does not explain why RPT behaves like it does. An important fact is that sampling in *Visitall* is very problematic, as it is possible to sample a state in which a visited cell can be surrounded by not-visited cells, which is unreachable from I. However, RPT still solves more problems than FD, which shows that the possibility of recovering from exploring plateaus that RPT has compensates the added difficulty of sampling the search space adequately.

In a few domains RPT is also significantly helped by pruning spurious actions. *Floortile*, *Matching-bw* and *Tidybot* are such domains. In a modified version of FD that performs the same preprocessing as RPT, pruning spurious actions reduces the difference in coverage in these domains by a large margin even if state invariants are not exploited explicitly by FD during search.

6.2. Parameters of RPT

In this work we experiment with the two parameters of RPT ε and p to analyze their impact. Higher values of ε mean that the local searches are larger, whereas p determines the balance between exploration and exploitation. Prior to the experimentation we expected to have very different results depending on the parameters of RPT; however, Table 1 shows that in reality the differences are not that big. Although different configurations of RPT have different coverage, their overall behavior compared to FD and LAMA is consistent. Also, there is no overall winner configuration, with different values of ε and p being more appropriate in some domains than in others.

The exception to this phenomenon is the *Barman* domain. As mentioned before, sampling in this domain is more complicated due to the frequent generation of spurious states. Because of this, larger values of ε and smaller values of p are preferable, since they reduce the sampling tendency of RPT and allow reproducing a behavior closer to FD's and LAMA's.

The lower coverage of the configuration with $\varepsilon = 1k$ shows that our choice of parameters is overall adequate. Nevertheless, even inadequate values of ε in terms of total coverage may still be good for some specific domains, such as *Airport* and both versions of *Pipesworld*, in which the $\varepsilon = 1k$ configuration achieves the maximum coverage. This suggests that the choice of parameters is a complicated problem on itself and should be analyzed for specific cases.

6.3. Tree Size

RPT builds the solution plans by concatenating the tree edges (sequences of actions that lead from one node of the tree to one of its children) that compose a path from the initial node to a goal node. Table 2 shows the number of tree edges that the solution with the highest number of tree edges in that domain has. This information is important because it is representative of the size of the tree in the cases in which it is able to scale up to bigger problems. The most obvious conclusion is that RPTs are significantly smaller than the RRTs used in motion planning, as the local searches are computationally much costlier in automated planning. However, in some domains in which the heuristic evaluation is not as costly, like *Visitall*, RPT is able to solve problems by building trees with hundreds of edges. A notable case is the reported 587 edges that RPT_10k_0.5 has to trace back to recover the solution of the hardest instance that it is able to solve in *Visitall*. Actually, several solution plans returned by RPT in *Visitall* have well over 10000 actions, which highlights the robustness and scalability of RPT.

As expected, the size of the tree appears to be inversely proportional to ε. The version with $\varepsilon = 1k$ has consistently bigger trees than the rest of the configurations, barring some domains in which the $\varepsilon = 10000$ version could not solve bigger instances than other configurations could.

Another relevant fact is that many problems are solved with a single extension even with $\varepsilon = 10k$. For example, a total of 946 problems were solved by RPT_10k_0.5 with a single extension out of the 1177 that it can solve overall. This is due to two reasons:

Planners:	10k_0.2	10k_0.5	10k_0.8	100k_0.2	100k_0.5	100k_0.8	100k_0.5	goals
Airport(50)	5	7	4	3	3	3	14	4
Barman(20)	2	2	2	11	21	13	-	45
Blocks(35)	1	1	1	1	1	1	3	2
Depot(22)	28	32	16	9	6	7	50	12
Driverlog(20)	5	13	10	5	7	6	45	9
Elevators(20)	10	9	4	5	5	4	32	8
Floortile(20)	22	12	4	2	2	1	35	83
Freecell(80)	5	2	2	2	2	3	7	5
Gold-miner(30)	1	1	1	1	1	1	2	1
Grid(5)	2	2	2	1	1	1	5	3
Gripper(20)	1	1	1	1	1	1	2	2
Logistics(35)	6	7	3	3	2	5	18	7
Matching-bw(30)	1	1	1	1	1	1	5	7
Miconic(150)	1	1	1	1	1	1	2	2
Movie(30)	1	1	1	1	1	1	2	2
Mprime(35)	1	1	1	1	1	1	2	2
Mystery(20)	2	2	2	2	2	2	6	2
Nomystery(20)	4	5	4	4	4	4	10	2
N-puzzle(30)	5	4	9	4	4	9	28	40
Openstacks(20)	4	3	2	1	1	1	9	5
Parcprinter(20)	1	1	1	1	1	1	2	2
Parking(20)	4	4	5	2	2	2	5	9
Pathways-noneg(30)	1	1	1	1	1	1	5	2
Pegsol(20)	3	3	2	2	3	2	5	6
Pipesworld-notankage(50)	29	4	4	5	5	5	14	4
Pipesworld-tankage(50)	27	7	10	6	6	7	16	4
PSR-small(50)	1	1	1	1	1	1	7	2
Rovers(40)	1	1	1	1	1	1	8	4
Satellite(36)	4	4	2	1	1	1	10	3
Scanalyzer(20)	2	3	3	2	2	2	6	9
Sokoban(20)	11	12	12	6	5	4	20	11
Sokoban-learning(30)	5	5	5	4	3	4	13	2
Spanner(30)	-	-	-	-	-	-	-	-
Storage(30)	9	10	10	10	6	5	25	4
Tidybot(20)	5	4	3	3	3	5	14	17
Tpp(30)	1	1	1	1	1	1	6	2
Transport(20)	13	8	10	3	3	5	117	18
Trucks-strips(30)	7	7	9	5	3	4	9	2
Visitall(20)	519	587	203	137	97	42	494	225
Woodworking(20)	6	5	4	2	2	2	46	9
Zenotravel(20)	1	1	1	1	1	1	3	2

Table 2

Number of tree edges that the solution with the highest number of tree edges in that domain has.

first, many of the domains come from old IPCs and thus are relatively easily solved by current state-of-the-art planners; second, most problems in planning are solved either very quickly or not at all due to the exponential blow up of the requirements in time and space as the size of the problems increase, which is also reflected by the size of the tree.

6.4. Sampling

As described in Section 5.1.3, sampling a state in an implicit search space requires solving a CSP. This CSP tends to be very small and the time spent solving it is on average negligible. Some exceptional cases occur though, when sampling a state takes a considerable amount of time. In the first run, in two problems of *Airport*, two problems of *Tidybot* and in the whole set of *Barman* problems sampling requires more than one second. In particular, for the first four cases it requires 699.55s (problem 46 of *Airport*), 21.88s (problem 47 of *Airport*), 152.53s (problem 19 of *Tidybot*) and 301.75s (problem 20 of *Tidybot*). In the *Barman* domain times range from 0.8s to not being able to sample a state under the time limit of 1800 seconds. However, in all these cases the problem is not the time required to solve the CSP, but rather that G is unreachable from the sampled states - which is detected by the reachability analysis from the sampled state. For example, in problem 46 of *Airport* more than 18000 states had to be sampled before a state from which G could be reached was found, which explains the long time spent sampling.

To ascertain that the heuristics proposed to solve the CSP are indeed necessary, we did some informal experimentation without them. After disabling forward checking and using a random ordering of the *"exactly-1"* invariant groups, in some domains solving the CSP was just not possible. For example, in many instances of *Sokoban* the backtracking algorithm was not able to sample a random state under the time limit of 1800 seconds. Such cases tend to occur in domains in which there are *"exactly-1"* invariant groups highly constrained by mutexes and other *"exactly-1"* invariant groups.

Finally, the relatively worse performance of the *goals* configuration, which implements RRT-Plan's sampling method, shows that random trees benefit more from random sampling than from choosing subsets of goals, even if the latter overcomes most problems derived from sampling in implicit search spaces. Again, there are exceptions to this, which means that RPT may benefit from switching from one method to the other randomly or when sampling by solving a CSP is not feasible.

6.5. Other Measurements

Although memory is usually not a bottleneck in satisficing planning, it is interesting to test whether the claims about the efficiency in terms of memory of RPT are true. In the experimentation we measured the memory necessary to store the tree and the auxiliary structures that allow a faster computation of the heuristic from the nodes of the tree. In all the problems, the amount of memory is smaller than the memory necessary to ground the problem and to compute h^2. This means that the memory needed during search by RPT is in practice determined by ε. As a final note and to confirm this fact, during the experimentation FD and LAMA run out of memory before running out of time in 40 and 25 problems respectively, whereas this never happened with RPT.

Regarding quality, the plans returned by RPT were consistently longer than the ones returned by FD and LAMA whenever more than one edge was needed to trace back the solution in RPT. Such is the case in both versions of *Sokoban*, in *Visitall* and in *N-puzzle*. For example, and using the quality score from the last IPC, RPT_10k_0.5's quality score in the first run is 8 points lower than FD's in *Sokoban* and very similar in *Sokoban-learning* despite it being able to solve 6 problems more in the latter domain. This was to be expected though, as the added exploration and the uniform sampling often cause RPT to take detours on its way to a goal state. The same case was observed with time. Besides, the preprocessing made by RPT can in some cases be longer than the time spent by FD and LAMA during search, which further skews the time score in favor of the latter.

7. Related Work

Stochastic search has also been employed by other planners. A prominent example is LPG [17], a planner inspired by random walk search algorithms. LPG searches in the space of plans employing a structure known as the action graph, choosing neighbor graphs based on a parametrized heuristic function. The main relation between LPG and RPT is that LPG tries to balance exploration and exploitation by performing ran-

dom restarts. These restarts help LPG to avoid exploring plateaus and local minima.

Random exploration has also been proposed in the context of forward-chaining planning. This includes the use of Monte-Carlo Random Walks to select a promising action sequence [34], local search combined with random walks [42] and the triggering of random walks when the planner detects that it has fallen into a heuristic plateau [31,41].

Lastly, inspired by real-time versions of RRTs [8], an RRT-like algorithm that stochastically interleaves search and plan reuse [39] has been proposed in [7]. In this case, the increased exploration comes from employing information from past plans instead of from sampled states. This includes reusing both plans and goals from previous searches.

8. Conclusions and Future Work

In this paper, we have analyzed how to adapt RRTs for their use in automated planning. Previous work has been studied and the challenges that the implementation of RRTs in contexts other than motion planning posed have been presented. In the experimentation we have shown that this approach has much potential, being able to outperform the state of the art. Besides, we have identified the major flaws of this approach, which may allow to obtain better results in the future.

The main challenge of the use of RRTs in automated planning, the design of an algorithm able to sample an implicit search space randomly, has been thoroughly studied. This novel problem has been tackled by exploiting state invariants of the problem extensively and formulating it as a CSP. The results show that, in domains in which current methods can find most relevant invariants, sampling is both efficient and useful.

Another of the drawbacks of RRTs, the estimation of the closest node, has also been analyzed. In this case we have generalized the cost-caching scheme previously proposed by the authors of RRT-Plan. Here we employ a more general definition of caching of reachability heuristics, inspired by recent work on regression in automated planning.

We have also examined the characteristics of the local planners that can be used along with RPT. We identified their requirements and defined the limit ε of the local search in terms of state-space planning.

In the experimental evaluation we have run tests over a broad set of benchmarks. We focused on both overall and *per domain* coverage, with an special emphasis on the defining features of the domains that may affect the performance of RPT in comparison with other state-of-the-art planners. Other measures have been presented, including some specific to RRT-like algorithms like tree size and sampling performance.

After this analysis, several lines of research remain open. First, some approaches like growing two trees at the same time in a bidirectional manner and the implementation of a proper *Connect* phase are still unexplored. Additionally, given how the time to compute reachability heuristics per node varies greatly depending on the problem in automated planning, using a time limit as ε may be an interesting approach. Besides, an anytime version of RPT that gradually covers the search space improving the quality of the best solution found so far may prove useful for setting in which both coverage and quality matters, as in the IPC.

In terms of sampling, several improvements are left as future work. First, using landmarks [24] to bias the sampling process could yield more representative sampled states, allowing a faster convergence towards a solution. This in fact could be seen as an intermediate step between sampling subsets of goals and our sampling method, as goals are landmarks themselves. Another interesting possibility is biasing how propositions are chosen using the distance to I and to G in a delete-free formulation of the problem, or learning on the fly which sampled propositions could not be easily reached in previous iterations of the tree construction.

From a planning perspective, techniques like caching the heuristic value of explored states to avoid recomputation when they are expanded several times [35] may prove interesting for this kind of algorithms. Another interesting possibility is the usage of portfolios of search algorithms or portfolios of heuristics combined with RRTs in order to compensate the flaws of both best-first search algorithms and RRTs.

As a last remark, another possible future line of research includes adapting this algorithm for a dynamic setting in which interleaving of planning and execution is necessary. This approach looks promising for domains in which exogenous events and partial information may force the planner to replan in numerous occasions.

Acknowledgements

This work has been partially supported by a FPI grant from the Spanish government associated to the MICINN project TIN2008-06701-C03-03, and it has

also been supported by the project TIN2011-27652-C03-02.

The fourth author was partially funded by the Office of Naval Research under grant number N00014-09-1-1031. The views and conclusions contained in this document are those of the authors only.

References

[1] V. Alcázar, D. Borrajo, S. Fernández, and R. Fuentetaja. Revisiting regression in planning. In *International Joint Conference on Artificial Intelligence*, pages 2254–2260, 2013.

[2] V. Alcázar, M. M. Veloso, and D. Borrajo. Adapting a Rapidly-Exploring Random Tree for automated planning. In *Symposium on Combinatorial Search*, pages 2–9, 2011.

[3] F. Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, Sept. 1991.

[4] J. Bibai, P. Savéant, M. Schoenauer, and V. Vidal. An evolutionary metaheuristic based on state decomposition for domain-independent satisficing planning. In *International Conference on Automated Planning and Scheduling*, pages 18–25. AAAI, 2010.

[5] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

[6] D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, Apr. 1979.

[7] J. Bruce and M. Veloso. Real-time randomized path planning for robot navigation. In *International Conference on Intelligent Robots and Systems*, pages 288–295, October 2002.

[8] J. Bruce and M. M. Veloso. Real-time randomized motion planning for multiple domains. In G. Lakemeyer, E. Sklar, D. G. Sorrenti, and T. Takahashi, editors, *RoboCup*, volume 4434 of *Lecture Notes in Computer Science*, pages 532–539, 2006.

[9] D. Burfoot, J. Pineau, and G. Dudek. RRT-Plan: A randomized algorithm for STRIPS planning. In *International Conference on Automated Planning and Scheduling*, pages 362–365, 2006.

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

[11] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA, June 2005.

[12] M. Christie, R. Machap, J.-M. Normand, P. Olivier, and J. Pickering. Virtual camera planning: A survey. In A. Butz, B. Fisher, A. Krger, and P. Olivier, editors, *Smart Graphics*, volume 3638 of *Lecture Notes in Computer Science*, pages 40–52. Springer Berlin Heidelberg, 2005.

[13] J. Cortés, T. Siméon, V. R. De Angulo, D. Guieysse, M. Remaud-Siméon, and V. Tran. A path planning approach for computing large-amplitude motions of flexible molecules. *Bioinformatics*, 21(suppl 1):i116–i125, 2005.

[14] J. C. Culberson and J. Schaeffer. Pattern databases. *Comput. Intell.*, 14(3):318–334, 1998.

[15] L. Da Xu, C. Wang, Z. Bi, and J. Yu. AutoAssem: an automated assembly planning system for complex products. *Industrial Informatics, IEEE Transactions on*, 8(3):669–678, 2012.

[16] R. Fuentetaja, D. Borrajo, and C. Linares López. A unified view of cost-based heuristics. In *Proceedings of the "2nd Workshop on Heuristics for Domain-independent Planning". Conference on Automated Planning and Scheduling (ICAPS'09)*, Thessaloniki, Greece, 2009.

[17] A. Gerevini and I. Serina. LPG: A planner based on local search for planning graphs with action costs. In *Conference on Artificial Intelligence Planning Systems*, pages 13–22. AAAI, 2002.

[18] R. M. Haralick and G. L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artif. Intell.*, 14(3):263–313, Oct. 1980.

[19] P. Haslum. Additive and reversed relaxed reachability heuristics revisited. *Proceedings of the 6th International Planning Competition*, 2008.

[20] P. Haslum. $h^m(P) = h^1(P^m)$: Alternative characterisations of the generalisation from $h^{\max}$ to h^m. In *International Conference on Automated Planning and Scheduling*, pages 354–357, 2009.

[21] M. Helmert. The Fast Downward planning system. *J. Artif. Intell. Res. (JAIR)*, 26:191–246, 2006.

[22] M. Helmert. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173(5-6):503–535, 2009.

[23] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res. (JAIR)*, 14:253–302, 2001.

[24] J. Hoffmann, J. Porteous, and L. Sebastia. Ordered landmarks in planning. *J. Artif. Intell. Res. (JAIR)*, 22:215–278, 2004.

[25] L. E. Kavraki, P. Svestka, L. E. K. P. Vestka, J. claude Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12:566–580, 1996.

[26] J. Koehler and J. Hoffmann. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *J. Artif. Intell. Res. (JAIR)*, 12:338–386, 2000.

[27] J. J. Kuffner and S. M. LaValle. RRT-Connect: An efficient approach to single-query path planning. In *ICRA*, pages 995–1001. IEEE, 2000.

[28] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[29] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In *International Conference on Robotics and Automation*, pages 473–479, 1999.

[30] C. Linares López and D. Borrajo. Adding diversity to classical heuristic planning. In *Proceedings of the Third Annual Symposium on Combinatorial Search (SOCS'10)*, pages 73–80, Atlanta, USA, 2010.

[31] Q. Lu, Y. Xu, R. Huang, and Y. Chen. The Roamer planner: random-walk assisted best-first search. In *The 2011 International Planning Competition*, 2011.

[32] B. Massey. *Directions In Planning: Understanding The Flow Of Time In Planning*. PhD thesis, Computational Intelligence Research Laboratory, University of Oregon, 1999.

[33] S. Morgan and M. S. Branicky. Sampling-based planning for discrete spaces. In *International Conference on Intelligent Robots and Systems*, pages 1938 – 1945, 2004.

[34] H. Nakhost and M. Müller. Monte-Carlo exploration for deterministic planning. In *International Joint Conference on Artificial Intelligence*, pages 1766–1771, 2009.

[35] S. Richter, J. T. Thayer, and W. Ruml. The joy of forgetting: Faster anytime search via restarting. In R. I. Brafman, H. Geffner, J. Hoffmann, and H. A. Kautz, editors, *International Conference on Automated Planning and Scheduling*, pages 137–144, 2010.

[36] S. Richter and M. Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *J. Artif. Intell. Res. (JAIR)*, 39:127–177, 2010.

[37] J. Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 193:45–86, 2012.

[38] G. Röger and M. Helmert. The more, the merrier: Combining heuristic estimators for satisficing planning. In *International Conference on Automated Planning and Scheduling*, pages 246–249, 2010.

[39] M. M. Veloso. *Learning by Analogical Reasoning in General Problem Solving*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, August 1992.

[40] V. Vidal. A lookahead strategy for heuristic search planning. In S. Zilberstein, J. Koehler, and S. Koenig, editors, *International Conference on Automated Planning and Scheduling*, pages 150–160, 2004.

[41] F. Xie, M. Müller, and R. Holte. Adding local exploration to greedy best-first search in satisficing planning. In *AAAI Conference on Artificial Intelligence*, pages 2388–2394, 2014.

[42] F. Xie, H. Nakhost, and M. Müller. Planning via random walk-driven local search. In *International Conference on Automated Planning and Scheduling*, pages 181–189, 2012.

[43] H. L. S. Younes and R. G. Simmons. VHPOP: Versatile heuristic partial order planner. *J. Artif. Intell. Res. (JAIR)*, 20:405–430, 2003.

[44] S. Zilles and R. C. Holte. The computational complexity of avoiding spurious states in state space abstraction. *Artificial Intelligence*, 174(14):1072–1092, Sept. 2010.