Adapting a Rapidly-Exploring Random Tree for Automated Planning

Vidal Alcázar
Universidad Carlos III de Madrid
Avenida de la Universidad, 30
28911 Leganés, Spain
valcazar@inf.uc3m.es

Manuela Veloso
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, USA
veloso@cmu.edu

Daniel Borrajo
Universidad Carlos III de Madrid
Avenida de la Universidad, 30
28911 Leganés, Spain
dborrajo@ia.uc3m.es

Abstract

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 as they offer a great degree of flexibility and reliability. However, their use in other search domains 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 that this approach has over previously used search algorithms and the challenges of adapting RRTs for implicit and discrete search spaces.

Introduction

Automated planning is an area in Artificial Intelligence that deals with the realization of plans or sequences of actions to achieve some goal. Problems in automated planning are formalized as an initial state and a set of goals that must be made true. Actions are defined as a set of operators with preconditions and effects. In the particular case of propositional planning, propositions are modeled with predicates related to objects, states and goals are represented by sets of propositions that are true and actions make true or false the propositions appearing in their effects.

Currently most of the state-of-the-art planners are based on the heuristic forward search paradigm first employed by HSP (Bonet and Geffner 2001). While this represented a huge leap in performance compared to previous approaches, this kind of planners also suffer from a series of shortcomings. In particular, certain characteristics of the search space of planning problems hinder their performance. Large plateaus for both g and h values, numerous transpositions in the search space and areas in which the heuristic function is misleading represent the main challenges for these planners. Besides, the most successful planners use techniques that increase the greediness of the search, which often exacerbates this problem. A couple of examples of these approaches are pruning techniques like helpful actions introduced by FF (Hoffmann 2001) and greedy search algorithms like EHC used by FF and greedy best-first search used by HSP and Fast Downward (Helmert 2006).

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 configuration. Some examples of motion planning problems are robotics, animated simulations, drug design and manufacturing. A broad range of techniques have been used in this area, although the current trend is to use algorithms that sample randomly the search space due to their reliability, simplicity and consistent behavior. Probabilistic roadmaps (PRMs) (Kavraki et al. 1996) and rapidly-exploring random trees (RRTs) (LaValle and Kuffner 1999) 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, 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 belonging to the opposite tree. This keeps a nice balance between exploitation and exploration and is on average 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 similarities. 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 search space of these problems is a bi-dimensional or three-dimensional explicit continuous space, whereas in automated planning is a multi-dimensional implicit discrete space. This has lead to both areas being developed without much interaction between them 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 plan-

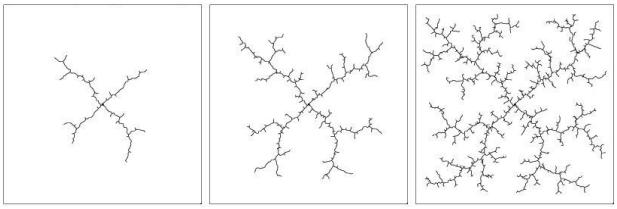

Figure 1: Progressive construction of an RRT.

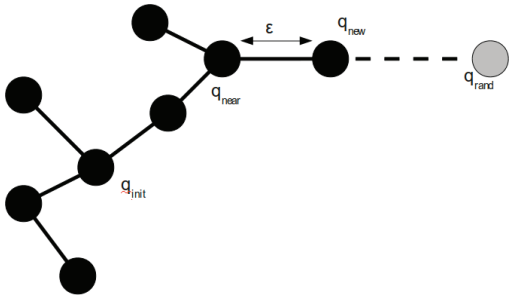

Figure 2: Extend phase of an RRT.

ners have while keeping most of their good properties. First, some background and an analysis of previous works will be given. Next, the advantages of using RRTs in automated planning will be presented and a description of how to overcome some problems regarding their implementation will be given. Some experimentation will be done to back up our claims and finally some conclusions and future lines of research will be added.

Background

In this paper we deal only with propositional planning, which we formalize. We then overview the use of RRTs in motion planning.

Propositional Planning

Automated Planning is described as the task of generating an ordered set of actions or plan that achieves a given set of goals from an initial state. In this work the standard formalization of a planning problem is represented by 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 and $G \subseteq S$ the set of goal propositions. The actions that are applicable depend on several propositions being true and their effects can make propositions true of false, which is commonly known as *adding* and *deleting* the propositions respectively. Thus, in terms of a planning instance an action would be defined as a triple $\{pre(a), add(a), del(a)\}$ in which $a \in A$ and $pre(a), add(a), del(a) \subseteq S$. Actions can have non-unitary costs, but in this work we will disregard this fact and consider all the actions as having a cost of 1, which makes the quality metric the plan length. A distinction must be made between satisficing planning, which revolves about finding any solution plan, and optimal planning, in which an optimal plan regarding some metric must be found. Again, in this work we will focus only on satisficing planning.

Rapidly-exploring Random Trees

RRTs (LaValle and Kuffner 1999) 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 as shown in Figure 1. At every step a random q_{rand} configuration is chosen and for that configuration the nearest configuration already belonging to the tree q_{near} is found. 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 of the local search is added to the tree in the same way as long as there was no collision with an obstacle during the search. This operation is called the *Extend* step, illustrated in Figure 2. This process is repeated until some criteria is met, like a limit on the size of the tree. Algorithm 1 gives an outline of the process.

Once the RRT has been built, multiples queries can be issued. For each query, the nearest configurations belonging to the tree to both the initial and the goal configuration 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 in the tree structure.

The key advantage of RRTs is that in their building process they are intrinsically biased towards regions with a low density of configurations. This can be explained by looking at the Voronoi diagrams at every step of the building process. The Voronoi regions of every node are larger when the area around that node has not been explored. This way, the probability of a configuration being sampled in an unexplored region are higher as larger Voronoi regions will be more likely to contain the sampled configuration. This has the advantage of naturally guiding the tree while just performing uniform sampling. Besides, the characteristics of the Voronoi diagram are an 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 ch aracteristic is that RRTs are probabilistically complete, as they will cover the whole search space as the number of sampled configurations tends to infinity.

RRT-Connect

After seeing how successful RRTs were for multi-query motion planning problems, a variation for single-query problems called RRT-Connect was proposed (Kuffner and LaValle 2000). The modifications introduced were the following:

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**
    $q_{rand} \longleftarrow sampleSpace(S)$
    $q_{near} \longleftarrow findNearest(tree, q_{rand}, S)$
    $q_{new} \longleftarrow join(q_{near}, q_{rand}, \epsilon, S)$
    if $reachable(q_{new})$ **then**
      $addConfiguration(tree, q_{near}, q_{new})$
  return $tree$

- Two trees are grown at the same time by alternatively expanding them. The initial configuration of the trees are the initial and the goal configuration respectively.
- The trees are expanded not only towards randomly sampled configurations but also towards the nearest node of the opposinte tree with a probability *p*.
- 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 configuration gives the algorithm its characteristic single-query approach. The *Connect* phase was added after empirically testing that the additional greediness that it introduced improved the performance in many cases. A common variation is also trying to extend the tree towards the opposing tree after every q_{new} is added from that q_{new} configuration when sampling randomly. This helps in cases in which both trees are stuck in regions of the search space which are close as per the distance measure but in which local searches consistently fail due to obstacles.

Previous Work

Although RRTs have not been frequently used in areas other than motion planning, there is previous work in which RRTs 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.

RRTs in Discrete Search Spaces

Although RRTs were designed for continuous search spaces, an attempt to implement them for search problems in discrete search spaces has been proposed (Morgan and Branicky 2004). The main motivation of this work was adapting the RRTs for grid worlds and similar classical search problems. In this work the main challenges of adapting the RRT were analyzed, in particular the need of defining an alternative measure of distance when finding the nearest node of the tree to a sampled state and the issues related to adapting a local planner that substitutes the *Expand* phase of the traditional RRTs. The alternative to the widely used euclidean distance that was proposed 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 planners, 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.

RRT-Plan

Directly related to automated planning, the planner RRT-Plan (Burfoot, Pineau, and Dudek 2006) was proposed as a randomized planning algorithm inspired by RRTs. In this case the EHC search phase of FF (Hoffmann 2001), 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 (Bonet and Geffner 2001) and the relaxed plan heuristic introduced by FF (Hoffmann 2001). 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 heuristic from every node in the tree every time a new local search toward s a state is done the authors propose caching the cost of achieving every proposition whenever a new 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 the reachability analysis.

Regarding sampling, RRT-Plan does not sample the search state uniformly but rather chooses a subset of propo-

sitions from the goal set and uses it 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 is reachable and whether the goal is reachable from the sampled 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, the set of goal propositions is reachable and hence so it is any of its possible subsets. In addition, RRT-Plan performs goal locking, i.e., when a goal proposition that conformed a given sampled state is achieved any subsequent searches from the added q_{new} node and its children nodes are not allowed to delete it.

Whereas RRT-Plan effectively addresses the problem of sampling states in implicit search spaces, the way of doing it limits most of the advantages RRTs have to offer. By choosing subsets of the goal set instead of sampling uniformly the search space the RRT does not tend to expand towards unexplored regions and thus loses the implicit balance between exploration and exploitation during the 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 similar method to the problem partitioning techniques that the authors that discovered landmarks proposed (Hoffmann, Porteous, and Sebastia 2004) (even more so taking into account that goals are landmarks themselves) albeit with the possibility in this case to recover from wrong orderings.

RRTs in Automated Planning

Motivated by the success of RRTs in motion planning, in this work we propose an implementation for their use in automated planning. We show the advantages they have over traditional forward search planners and analyze the difficulties that must be overcome when adapting them for planning problems.

Motivation

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 search algorithms along with reachability heuristics and other associated techniques. However, forward 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 present. Search spaces in automated planning tend to have big plateaus both in terms of *g* and *h* value. The high number of transpositions and the high branching factor that appear in many domains aggravate this fact. Forward 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 du ring the search. It has been demonstrated that techniques that increase the greediness of the search algorithm like helpful actions from FF and look-ahead states from YAHSP (Vidal 2004) 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 and thus this increased greediness can be disadvantageous at times.

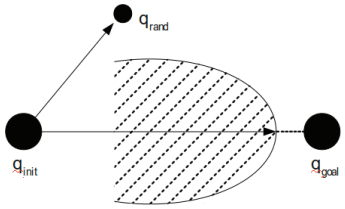

Figure 3: Simple example of a best-first search algorithm greedily exploring an *h plateau* due to the heuristic ignoring the obstacles. Advancing towards some randomly sampled state like q_{rand} can alleviate this problem.

Figure 3 shows a typical example of a best-first search algorithm getting stuck in a *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 shows the imbalance between exploitation and exploration this kind of approaches have. This problem has been previously analyzed (Linares López and Borrajo 2010), but it is still one of the main shortcomings of best-first search. To partially address this issue it is interesting to 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.

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

- They keep a better balance between exploration and exploitation during the search.
- Local searches minimize exploring plateaus, as the maximum size of the search is limited.
- 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.

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, but instead search is performed over several areas at the same time. This way they tend to recover better than best-first search from falling into local minima.

Besides, even though the local search may fall in a plateau, the number of nodes that will be expanded is limited and thus not much effort will be wasted in those cases.

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.

Implementing the RRT

Because of 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.

Sampling The main reason why RRTs have not been studied in automated planning is probably the difficulties of sampling in the search space. RRT-Plan circumvented this fact by substituting uniform sampling with subsets of goals. However, this negates some of the advantages that RRTs have. Here we propose the use of uniform sampling along with two techniques to reduce the odds of obtaining an unreachable state.

When sampling a state, two facts must be considered: first, the sampled state must be reachable from the initial state, and second, the goal must be reachable from the sampled state. Checking whether this is true is as hard as solving the problem itself, so an approximative approach must be used instead. One of the previously used techniques for this purpose is the concept of mutual exclusivity between propositions (Haslum and Geffner 2000). A static relation of mutual exclusivity between propositions (or static mutex) is a set of propositions $M = (p_0, \ldots, p_n)$ for which there is no reachable state $s \subseteq S$ such that all the elements in M are true. Static mutexes may not suffice to detect all the unreachable sets of propositions in the search space but in many domains they are able to prune most unreachable states. This is a similar approach to the one used by evolutionary planners that decompose the problem using sampling techniques (Bibai et al. 2010).

Again, computing all the static mutexes for a given problem is PSPACE-complete. Nevertheless, there are two methods that allow the computation of a subset of the mutexes of the problem in reasonable time: invariant synthesis (Helmert 2009) and h^m (Haslum and Geffner 2000). Invariant synthesis uses information from the operators of the problem to find groups of monotonicity invariants. These are sets of propositions from which only one proposition can be true at a time, which means that they are mutex between them. On the other hand, h^m gives a lower bound of the distance from the initial state to a state in which any set of *m* propositions are true. If this bound is infinite, those propositions are mutex. The complexity of the computation of these sets grows exponentially with the number of propositions in the set, so usually only sets of 2 propositions are used to find mutexes. These methods yield different mutexes. The invariant synthesis is usually much faster, as it only uses information from the domain definition, although h^2 exploits information from the initial state, which may prove essential to find additional mutexes.

Even after using mutexes there are cases in which a state is not reachable or the goal is not reachable from that state. For example, a sampled state in Sokoban may contain a block in a corner which is not a goal location. While this sampled state may not violate any mutex, it is a dead end as the block is effectively unmovable and thus the goal is not reachable. To address this problem a regular reachability analysis can be done both from the initial state and from the sampled state. If some proposition in either the sampled state or the goal is unreachable then the sampled state can be safely discarded. This is again a non-complete method, but in cases such as the aforementioned one it may be useful.

In this work we use a SAS+ representation of the problem (Bäckström and Nebel 1995). Taking this into account, the sampling method works as follows: first, a SAS+ invariant is chosen at random; then, all the propositions mutex with previously chosen propositions are discarded; after that, a proposition from those belonging to the SAS+ invariant is randomly chosen and added to the state. This is repeated until all SAS+ invariants have been chosen. If no propositions are left when taking out propositions of some invariant that were mutex with the already chosen propositions it means that the partial state is unreachable and the process is restarted from scratch. Once a complete state is obtained, a reachability analysis is done from the initial state towards the sampled state and from the sampled state towards the goal state. If some proposition is unreachable then the state is discarded.

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 proposition from every node of the tree and using that information to compute h_{add}. Despite being an elegant and efficient solution, this has the problem that 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 relaxed plan heuristic used by FF, are on average more accurate (Fuentetaja, Borrajo, and Linares López 2009). Because of that, in our implementation *best supporters*, that is, actions that first achieve a given proposition in the reachability analysis, are cached. This allows to compute not only h_{add} by adding the costs of the best supporters of the sampled propositions but also other heuristics like the relaxed plan 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, so this approach allows to get more accurate (or diverse) heuristic estimates without incurring in a significant over-

head.

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 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 set 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; 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; ϵ limits the number of expanded nodes in every local search; there is a probability *p* of advancing towards the goal and a probability *1-p* of advancing towards a sampled state; when performing a local search, only the closest node to the goal state (a sampled state or the original goal itself) is added to the tree when the goal state has not been reached; after adding a new node q_{new} from local search towards a sampled state, a new local search from q_{new} to q_{goal} is performed. Algorithm 2 describes the whole process.

Algorithm 2: Search process of the proposed RRT

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

Experimentation

We run some experiments on a Dual Core Intel Pentium D at 3.40 Ghz running under Linux. We called our planner Random Planning Tree (RPT). The maximum available memory for RPT was set to 1GB and the time limit was 1800 seconds. RPT was implemented on top of Fast Downward (Helmert 2006). Fast Downward itself was used as the local planner. It was configured to use greedy best-first search with lazy evaluation as its search algorithm. The heuristic is the relaxed plan heuristic used by FF (Hoffmann 2001). Preferred operators and boosting were enabled. Mutexes were computed by the invariant analysis Fast Downward's translator performs. Since RRTs are stochastic algorithms all the experiments were done using the same seed for the pseudorandom number generator so results would be reproducible. The base planner we compare against is LAMA-2008 (Richter and Westphal 2010), the winner of the International Planning Competition (IPC) held in 2008.

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 of expanding towards the original goal instead of towards some sampled state *p*. In the experimentation we tried three different combinations, one with $\epsilon = 10000$ and $p = 0.5$, a second one with $\epsilon = 1000$ and $p = 0.5$ and a third one with $\epsilon = 10000$ and $p = 0.2$. These configurations were called RPT1, RPT2 and RPT3 respectively. For the case in which $\epsilon = 1000$ (RPT2) we changed the default boosting value of Fast Downward, which is 1000, to 100, as otherwise this feature would never be used by the local planner. By experimenting with different versions we aim to understand which are the factors that can have an impact on the performance of the algorithm. In particular, RPT1 makes relatively large local searches and tends to expand frequently towards the goal, RPT2 makes smaller local searches with the same *p* value and RPT3 makes large local searches but expands most of the time towards sampled states instead of towards the goal.

The focus of this experimentation is coverage, that is, the number of solved problems in each domain. Since RRTs expand towards random states it is logical to expect worse results in terms of quality than with a more traditional approach, so we did not consider quality as an important criteria. The domains were the ones used in the last IPCs: IPC-06, IPC-08 and IPC-08 learning track. Since there are two versions of Sokoban (deterministic track and learning track) we have called the one from the learning track Sokoban-L. Some domains were left out due to some bugs. Table 1 summarizes the obtained results.

As shown by the results, the performance of RPT varies greatly depending on the domain. On average LAMA still solves more problems; however, we found that the main reason why RPT behaves much worse in some domains is that the mutexes obtained from the invariant analysis are not representative enough. A notable case is Sokoban, which has two versions despite being essentially the same domain. In the deterministic track version, RPT fares much worse than LAMA, whereas in the learning track version the situation is the opposite. This is due to the fact that in the deterministic track version, the propositions obtained from instantiating the predicate *(at-goal ?s - stone)* do not appear as mutex with the stone being at a non-goal location, which leads to frequently sampling unreachable states. Similar cases occur in other domains. For example, in Matching-Blocksworld the propositions obtained from instantiating the predicate *(solid ?b - block)* do not appear as mutex with some other block being on top of the non-solid one. h^2 is able to detect these mutexes because it uses information from the initial state, so our guess is that combining both techniques (using Fast Downward means basically that the invariant mutexes come for free) would greatly improve the overall performance in those cases. All in all, this confirms the importance that proper

Planners:	LAMA	RPT1	RPT2	RPT3
Openstacks(30)	30	30	30	30
Pipesworld(50)	27	42	39	41
Rovers(40)	40	40	39	40
Storage(30)	18	22	24	25
TPP(30)	30	28	25	25
Trucks(30)	25	17	7	9
Elevators(30)	25	26	22	23
Parcprinter(30)	17	11	11	11
Pegsol(30)	30	30	30	30
Scanalyzer(30)	29	30	29	30
Sokoban(30)	22	10	5	8
Transport(30)	30	30	27	28
Woodworking(30)	29	16	16	16
Gold-Miner(30)	29	30	30	30
M-Blocksworld(30)	24	19	4	5
N-Puzzle(30)	26	26	15	15
Parking(30)	25	12	6	7
Sokoban-L(30)	18	30	30	30
Total	474	449	389	403

Table 1: Comparison between LAMA and the different configurations of RPT. Numbers between parenthesis represent the total number of problems.

sampling has in implicit search spaces.

Regarding the different configurations of RPT, a noteworthy fact is that larger local searches dominate smaller ones in all the domains but Storage. The reason behind this is that by expanding only 1000 nodes some local searches do not have enough margin to make a significant improvement. Actually, with this ϵ the new added nodes tend to be close to the node that was used as the initial state for the local search, which means that the tree explores the space at a much lower rate. This may be related to the reasons that lead to the authors of RRT-Connect to implement the *Connect* phase, which essentially lets the local search continue as long as there is an improvement.

Another important factor is the value of p. For the same ϵ the configuration that tends to expand towards the goal more frequently obtains better results in a consistent way. In fact the same case as with the ϵ value occurs: RPT1 dominates RPT3 in all the domains but Storage. The additional greediness of higher values of p seems to be useful in cases in which an excessive exploration of the search space leads to expanding a higher number of nodes, which causes less greedy configurations to time out due to the time spent evaluating the expanded nodes.

In terms of expanded nodes in most cases the RPTs expand a higher number of nodes. This is because the random sampling often makes them explore regions of the search space irrelevant to the solution. This situation is reversed in those instances in which the best-first search algorithm of LAMA gets stuck in big plateaus. A straightforward comparison is difficult to do for two reasons though: first, in many domains LAMA and RPT1 tend to solve different instances, and second planners usually solve the problems either very quickly or not at all, which means that for most instances solved by both approaches LAMA solves the problem too quickly for the additional exploration of RPT to pay off.

A relevant piece of information that can help to understand the behavior of RPT is the number of branches that were traced back when retrieving the solution plan. In most cases the number of branches was low, ranging from 2 to 5 in most instances (in the smallest instances the local search was able to reach the goal from the initial state for the configurations with $\epsilon = 10000$). The size of the final tree is also a good indicative of the behavior of the algorithm. Due to the computational cost of evaluating the nodes, only a limited number of local searches can be performed before the time limit is reached. This means that the size of the tree is very small compared to the trees built in motion planning. In our experiments only a few times the resulting tree contained more than a hundred nodes when the instance was solved.

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 potential, being able to outperform the state of the art in some domains. Besides, we have identified the major flaws of this approach, which may allow to obtain better results in the future.

After this analysis several lines of research remain open. First, some approaches like growing two trees at the same time like in bidirectional search and the implementation of a proper *Connect* phase are still unexplored. Besides, there is abundant research currently done on RRTs related to avoiding pathological cases and introducing biases that allow a more advantageous sampling. These techniques can also be studied for their use in automated planning so the overall performance is improved.

From a more planning perspective, techniques like caching the heuristic value of explored states to avoid recomputation when they are expanded several times (Richter, Thayer, and Ruml 2010) may prove interesting for this kind of algorithms. Other interesting possibility is the usage of portfolios of search algorithms or heuristics combined with RRTs to compensate the flaws of 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. Inspired by real-time versions of RRTs (Bruce and Veloso 2006), this approach looks promising for domains in which exogenous events and partial information may force the planner to replan in numerous occasions.

Acknowledgments

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

The second 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

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS+ planning. *Computational Intelligence* 11:625–656.

Bibai, J.; Savéant, P.; Schoenauer, M.; and Vidal, V. 2010. An evolutionary metaheuristic based on state decomposition for domain-independent satisficing planning. In Brafman, R. I.; Geffner, H.; Hoffmann, J.; and Kautz, H. A., eds., *ICAPS*, 18–25. AAAI.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artif. Intell.* 129(1-2):5–33.

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

Burfoot, D.; Pineau, J.; and Dudek, G. 2006. RRT-Plan: A randomized algorithm for STRIPS planning. In *ICAPS*, 362–365. AAAI.

Fuentetaja, R.; Borrajo, D.; and Linares López, C. 2009. 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)*.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In *AIPS*, 140–149.

Helmert, M. 2006. The fast downward planning system. *J. Artif. Intell. Res. (JAIR)* 26:191–246.

Helmert, M. 2009. Concise finite-domain representations for pddl planning tasks. *Artif. Intell.* 173(5-6):503–535.

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

Hoffmann, J. 2001. FF: The Fast-Forward planning system. *AI Magazine* 22(3):57–62.

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

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

LaValle, S. M., and Kuffner, J. J. 1999. Randomized kinodynamic planning. In *ICRA*, 473–479.

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

Morgan, S., and Branicky, M. S. 2004. Sampling-based planning for discrete spaces. In *ICRA*.

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

Richter, S.; Thayer, J. T.; and Ruml, W. 2010. The joy of forgetting: Faster anytime search via restarting. In Brafman, R. I.; Geffner, H.; Hoffmann, J.; and Kautz, H. A., eds., *ICAPS*, 137–144. AAAI.

Vidal, V. 2004. A lookahead strategy for heuristic search planning. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *ICAPS*, 150–160. AAAI.