RRT-Plan: a Randomized Algorithm for STRIPS Planning

Abstract

We propose a randomized STRIPS planning algorithm called RRT-Plan. This planner is inspired by the idea of Rapidly exploring Random Trees, a concept originally designed for use in continuous path planning problems. Issues that arise in the conversion of RRTs from continuous to discrete spaces are discussed, and several additional mechanisms are proposed to improve performance. We pose several problems that planners based on the relaxed plan length heuristic cannot solve but RRT-Plan can. Our experimental results indicate that RRT-Plan is competitive with the state of the art in STRIPS planning, and that the use of randomization does not significantly worsen plan quality. The success of RRT-Plan indicates that similar randomization techniques could be effective in more advanced planning domains.

Introduction

In this paper we propose a new approach to planning in discrete domains. We demonstrate this in the context of STRIPS planning, due to its prevalence and generality. The STRIPS language offers a compact and expressive means to represent planning problems where the state can be defined by a list of positive literals (Fikes & Nilsson 1971). Such a simple language can capture a surprisingly complex set of planning domains, and new planning algorithms for STRIPS-type planning problems are an active area of research (Vidal 2004; Gerevini & Serina 2002; Helmert 2004).

Several recent successful algorithms for STRIPS planning are in the class of *heuristic* planners, e.g. HSP (Bonet & Geffner 1999) and FF (Hoffmann & Nebel 2001), which define a state-specific heuristic evaluation function to guide search through the state space. Despite their success, heurstic planners, due to their deterministic nature, are subject to a number of weaknesses. In particular, in many domains the heuristic function can give bad estimates, or can fail to adequately distinguish neighbouring states. Randomization is a standard approach that can help escape local minima and to mitigate the impact of systematically bad information.

The use of randomization in planning has generated significant attention in the field of robot path planning, where (LaValle 1998; LaValle & Kuffner 2000) introduced the notion of Rapidly exploring Random Trees (RRTs). RRTs interleave large scale stochastic exploration with deterministic but incomplete local goal searches. RRTs are known to be particularly useful in robotics path planning domains where the state space is continuous, highly dimensional, and there is a relatively large goal area. With the notable exception of (Morgan & Branicky 2004), RRTs have not been extended to discrete-space planners. These authors considered only the general case of discrete search, and did not leverage any of the ideas or techniques that have been developed for STRIPS planning.

The primary contribution this paper is a novel algorithm for solving STRIPS problems through the use of randomized search. We call this algorithm RRT-Plan. There are a variety of difficulties that arise with attempting to translate the RRT concepts from the continuous case to the discrete case. First, RRTs require the ability to randomly sample a state from the state space. While this is relatively straight-forward in continuous spaces, sampling feasible states can be challenging in discrete domains. Second, given a random state, RRTs require the ability to identify a "nearest neighbour" (in state space). Again, this has a straight-forward interpretation in continuous domains, but is less clear in discrete planning. Finally, a sub-planner must be invoked to try to connect the current state to the goal. This paper discusses how RRT-Plan is able to overcome these problems specifically for the STRIPS representation, and argues that many of these ideas extend nicely to more general discrete planning problems.

In addition to presenting the core RRT-Plan algorithm, we present results validating performance of RRT-Plan compared to the state-of-the-art planners FF (on whose technology we rely significantly), and LPG (Gerevini & Serina 2002). We present several techniques which allow RRT-Plan to effectively focus on solving subgoals as well as to adapt its search parameters based on information from the growth of the tree. Finally, we describe a class of problems which are known to be particularly challenging for heuristic planners, and show that these can be easily overcome by RRT-Plan due to its randomized searching.

STRIPS planning

A STRIPS problem is defined by a set of atoms A, a set of operators O, an initial set of atoms I, and a set of goal

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

atoms G. Associated with each operator $o \in O$ is a set of preconditions $prec(o)$, add effects $add(o)$ and delete effects $del(o)$. A state s is a set of atoms. An operator can be applied to a state if all of the operator's preconditions are asserted in that state. The resulting state $s' = Result(s, o)$ is the same as s, but with the atoms in $add(o)$ added and those in $del(o)$ removed.

The canonical example of a STRIPS planning problem is Blocks World (McAllester & Rosenblitt 1991). This classic domain models the arrangement of a collection of building blocks. In each problem there is some number of blocks in an initial configuration, and the goal is to achieve a different specified configuration. Another standard STRIPS planning problem is Logistics, in which the agent must direct the shipment of packages by trucks and airplanes.

STRIPS planning has been studied extensively in the literature (Bylander 1994; Blum & Furst 1995). We now describe two very successful heuristic planners.

HSP - Heuristic Search Planner

(Bonet & Geffner 1999) first introduced the Heuristic Search Planner (HSP). The algorithm formulates the planning problem as search through a discrete state space, where a heuristic estimate of the distance (# of actions) between any state and the goal is used to guide the search. Given the heuristic, a refinement of Best-First Search is used until the goal is reached.

The heuristic estimates of goal distance are obtained by considering a *relaxed* version of the problem, where the delete effects are ignored. It is impossible to reach the goal more quickly with delete effects than without, therefore whenever the length of the optimal relaxed plan can be calculated, the heuristic estimate is *admissible*. Unfortunately, solving the relaxed planning problem is itself NP-hard (Bylander 1994), therefore in practice HSP uses an approximation of the relaxed costs. This is generally quite effective and HSP can successfully solve a large number of STRIPS planning problems.

HSP approximates the minimum path length to the goal, denoted $h^+(s)$ by summing the (estimated) minimum path length (or cost) to each atom in the goal. For a given state s, a copy s^* is made. The set of atom costs $g(p)$ is initialized to zero for all $p \in s$, and ∞ otherwise. We then find all operators that are applicable in s^*, and add all atoms asserted by those operators. The costs for atoms are updated according to the rule:

$$g(p) := min[g(p), 1 + g(C)] \quad (1)$$

Where C is the set of preconditions of the operator that adds p. The process continues until a steady state is reached. At this point all atoms with $g(p) = \infty$ are unachievable, and if the goal contains any such atoms it is an impossible problem. The function $g(C)$ can be either the sum or the maximum of $g(p)$ for the atoms in C. $h^+(s)$ is just $g(G)$, where G represents the atoms in the goal state.

For the purposes of this paper, it is important to note that for a given state, every atom has a specific cost. Thus if one were to change the goal definition, the atom costs do not need to be recalculated to obtain a new heuristic estimate.

FF - Fast Forward Planner

The Fast Forward Planner by (Hoffmann & Nebel 2001) is similar to HSP in that it uses heuristic search, and the heuristic is based on the length of the relaxed plan. However FF introduces several improvements.

First, a more sophisticated technique is used to estimate the length of the relaxed plan. This technique uses the *planning graph*, from Blum and Furst's Graphplan procedure (Blum & Furst 1995), to get an improved heuristic estimate. This approximation is more precise because the planning graph is able to take into account positive interactions between goal atoms, which HSP's technique ignores.

In addition, FF features several pruning devices to further refine the search. The two main ones are called *helpful actions* and *added goal deletion*. Helpful actions are those which are applicable in the first action level of the relaxed planning graph. The intuition is that these actions provide the most direct means of achieving the goal atoms, and therefore other (i.e. non-helpful) actions can be pruned from the search tree. Added goal deletion instructs the algorithm not to expand nodes in which goal atoms have apparently been achieved too early, as indicated by the fact that they are later deleted in the relaxed plan solution.

Finally, FF uses two stages of search. The first is called *Enforced Hill Climbing*. During this stage FF employs the above-mentioned aggressive search-pruning techniques, and moves monotonically in the direction of improving heuristic evaluation. The second phase is a *Greedy Best First Search*, similar to HSP, which is slower but complete. As a result, FF is guaranteed to find a solution if one exists. In pracice, Hoffmann has shown that a significant number of planning domains exhibit topologies that allow FF to run in polynomial time (Hoffmann 2002).

The net effectiveness of the several innovations discussed above make FF one of the most successful STRIPS planners. However, there are a number of simple domains in which the h^+ heuristic or FF's pruning techniques break down.

We illustrate some such situations later in this paper, and demonstrate that by injecting randomization in the search procedure, these problems can be easily overcome.

Rapidly-exploring Random Trees

Rapidly-exploring Random Trees (RRTs) were first introduced by (LaValle 1998; Kuffner & LaValle 2000) to solve multidimensional path planning problems. The key concept of RRTs is to use a randomized search process to grow a tree of paths through the state space, until one of the nodes in the tree is sufficiently close to the goal that the goal can be found using a short deterministic search procedure. A good example of the kinds of problems that can be solved easily by RRTs is that of a humanoid robot picking up an object (Kuffner *et al.* 2003). The robot has many degrees of freedom, and thus the planning problem has high dimensionality. The size of the search space is prohibitive for most traditional deterministic planners, but RRTs are able to find a solution in reasonable time.

RRTs were designed for continuous metric (but not necessarily Euclidean) state spaces and are constructed by in-

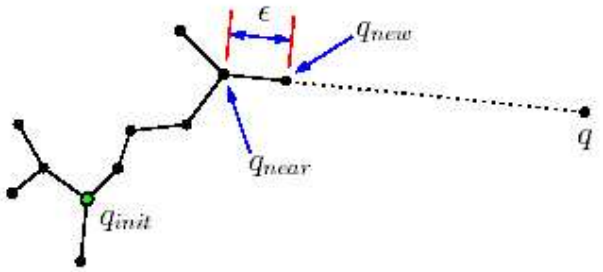

Figure 1: RRT Expansion process. q_{init} is the initial state, q is a randomly chosen point, q_{near} is its nearest neighbor in the tree. A local planner is invoked to connect q_{near} to q, but is only allowed to progress a distance ϵ. From (Kuffner & LaValle 2000).

crementally growing a search tree. Initially the tree has only one node located at the starting state. Then we choose a point q_{rand} at random in the state space. We find q_{rand}'s nearest neighbor in the tree, that state is denoted q_{near}. Next we try to connect q_{rand} and q_{near} using a simple local planner, which essentially just moves in a straight line from one point to the other until it encounters an obstacle. If the local planner succeeds in reaching q_{rand}, it is added as a new node in the tree. Typically the search is only allowed to proceed for some distance ϵ, in which case the location reached (called q_{new}) is added instead. This process is illustrated in figure 1. The distance ϵ is a parameter that must be set; if it is too low, the tree will require many iterations to grow.

When the RRT is grown in this way, it exhibits the following property. If the random points are chosen *uniformly* over the state space, then the tree expands to sample the configuration space uniformly. If the points are chosen according to some other distribution $p(x)$, then the tree will converge to a sampling of the space by $p(x)$ (LaValle 1998).

(Kuffner & LaValle 2000) extended the basic RRT concepts to create an algorithm called RRT-Connect. This simple but powerful algorithm works by growing an RRT from the start state, and after every expansion phase attempting to connect to the goal. Because of the uniform sampling property of the tree, the algorithm is probabilistically complete, since eventually a point will be reached from which the goal is directly achievable with the local planner. A more sophisticated method grows two trees, one from the start and one from the goal.

RRTs provide no guarantee over the quality of the final plan (e.g. minimum length). In high-dimensional robot path planning, it is often the case that finding some acceptable plan is the overriding concern, while plan optimality is much less important. RRTs are particularly useful when solutions are not scarce, and there are many acceptable paths to the goal. Intuitively, RRTs allow us to find one solution out of many in high dimensional search spaces. Conversely, RRTs are not very good at finding a solution when there is only a small bundle of them, for example in the case where the robot must navigate a narrow corridor.

- **Select random goal subset RGS**
- **Find "Nearest Neighbor" qNear to RGS**
- **Invoke planner to connect qNear to RGS**
- **If state qNew is found that satisfies RGS,**
  - Add qNew to tree as child of qNear
  - Calculate atom costs for qNew
  - Attempt to connect qNew to final goal

Figure 2: One iteration of tree expansion.

In STRIPS planning domains, the goal is specified as a subset of state atoms. Thus in many cases there are a large number of acceptable goal states and many paths to the goal states. RRTs exploit the existence of many possible solutions, and we will attempt to do so as well in the algorithm we propose below.

The RRT-Plan Algorithm

RRT-Plan is a randomized planning algorithm for discrete state spaces, in particular those that can be represented using the STRIPS planning language. The algorithm extends well-known heuristic planners such as HSP and FF, by introducing randomization in the exploration of the state space, therefore providing the ability to escaple plateaus in the heuristic function.

At a high-level, RRT-Plan contains much the same steps as the RRT-Connect algorithm described above. An outline is provided in Figure 2. The basic idea is to expand a tree over the discrete state space, in random directions (guided by the sampling of states), until a node is found that is sufficiently close to the goal that they can be connected by a short deterministic search.

There are several obstacles when trying to extend the concepts of RRTs to the discrete planning case. First, RRTs require the ability to randomly sample from the state space. Second, given a random state, RRTs require the ability to find its *nearest-neighbour* in the tree. Finally, a deterministic planner must be invoked as a sub-routine to try to connect nodes to random states and to the goal. We now discuss each step of our algorithm in detail.

Select random state (q_{rand}). The first step relies on the ability to sample points randomly from the space. While it is not difficult to sample randomly, it is practically impossible to sample the reachable space uniformly. This is because determining if any given state should be assigned a non-zero probability (i.e. determining if it is reachable) is equivalent to solving the planning problem itself.

Because of this difficulty, RRT-Plan generates target states q_{rand} by taking random subsets from the goal atoms. This approach has several advantages. It is easy to compute. If the problem is solvable, then every goal subset must itself be reachable. Finally, it tends to bias the search towards the

goal. In the following we abbreviate "random goal subset" as RGS.

Some domains have natural goal orderings, meaning that some goal atoms must be achieved before others. The classic example of this is again Blocks World. Several methods exist to discover goal orderings (Korf 1985; Knoblock 1990); we use the heuristic technique of (Koehler & Hoffmann 2000). By utilizing this information, we can improve the selection of random goal subsets by respecting the ordering relationships between goal atoms. Specifically, no goal atom is included in the RGS unless all goal atoms which must be achieved before it are also included. While the ordering would eventually be discovered through random trial and error, using the computed goal agenda speeds up the process.

Find Nearest Neighbor. The second step in the expansion process requires finding the node in the tree that is closest to the random target. While this distance is well-defined in terms of the number of actions required to get from one state to the other, finding the precise value is again equivalent to the planning problem itself.

However this specific problem has been considered at length by the heuristic planners, which try to estimate distance between a state and the goal. Therefore we turn to them for inspiration on this issue.

FF and HSP both use the relaxed plan length heuristic, called h^+, but use different methods to estimate this function. We adopt the technique proposed by HSP (Eqn 1) because the distance is evaluated as a per atom cost, which can be easily reused for different target states (i.e. different q_{rand}). FF's heuristic on the other hand requires a relaxed plan graph to be constructed every time a new target is considered. In RRT-Plan, each tree expansion phase requires one distance computation for each node in the tree, therefore this would become prohibitively expensive as the tree grows larger.

Note that it is possible that the nearest neighbor node in the tree actually contains the RGS target. In this case extra goal atoms are added to the RGS until it is no longer contained by the nearest neighbor.

Invoke sub-planner to connect q_{near} to q_{rand}. RRT-Plan requires a sub-planner at two steps - first to connect the nearest neighbor q_{near} to the target q_{rand}, and second to connect the new node q_{new} to the goal. In both cases, we propose to use FF, with some modifications.

Recall that FF features two stages of search: a fast phase of *Enforced Hill Climbing*, and a slow (but complete) phase of *Greedy Best First Search*. When applying FF as a sub-planner in RRT-Plan, we only use only the first of these phases. In addition, we only expand a limited number of nodes. The search is restricted in these ways because the connection attempt might be impossible if q_{near} is a dead end, though significant effort is applied to prevent dead ends from being added to the tree (see below). If a certain point cannot be reached easily, we move on and do another expansion iteration.

When using FF to perform local search, the added goal deletion technique is turned off. A similar method called *goal subset locking* is used in RRT-Plan (see below). The helpful actions technique is used, but the stringency of action pruning is modulated as certain atoms begin to appear difficult to achieve.

Add q_{new} to tree. We maintain a simple tree structure in memory. When a goal subset q_{rand} is reached from q_{near}, a new node is added to the tree as a child of q_{near}, and the actions required to reach q_{rand} are stored. If the full goal is reached from a node in the tree, we can construct a full solution by following the path from the root to the node and then to the goal. In contrast to the continuous formulation, if the sub-planner fails to connect to the target, no new node is added to the tree.

Calculate atom costs for q_{new}. We use the HSP heuristic defined in Eqn 1 to calculate atom costs for any new node. These reflect the estimated cost of achieving the goal from state q_{new}.

Attempt to connect q_{new} to goal. Again, we use the *Enforced Hill Climbing* phase of FF, with bounded node expansion. Normally the connection attempt does not succeed. However, the algorithm has invested time in finding a route to this new state, and should use the knowledge that the new state can be reached. Thus the best (smallest heuristic value) state discovered in the search is added to the tree.

Importantly, this allows is for the node expansion limit to be effectively bypassed. Consider the following scenario. A new RRT node is created, and then an attempt is made to reach the goal. This attempt fails because of the node expansion limit, but would have succeeded if it had been allowed to continue. Because the resulting state is also added as a new node, it can be selected as nearest neighbor on the next iteration, and the search can be continued from the point it was halted due to the expansion limit.

One common occurrence is as follows. The algorithm will perform a goal connection attempt, and make good progress, achieving many goal atoms and obtaining a low heuristic value. However due to some subtlety of the problem it will fail to achieve the goal. The best state is added to the tree, and on the following iteration it is selected as nearest neighbor. The search moves to the random goal state, which constitutes a slight detour which puts the goal in direct reach. A good example of this is the DriverLog domain. On the first goal attempt, all of the packages will be delivered to the correct locations, but getting the drivers and trucks to their destinations is more difficult. The tree continues to grow from this near goal state, and the problem is solved after a bit of trial and error.

Goal Subset Locking Several of the problems encountered by $h+$ planners are caused by the fact that the relaxed plan length heuristic does not penalize the deletion of goal atoms. If a goal atom can be achieved in an "easy" way through the deletion of an already asserted goal atom and in a "hard" way (in which other goal atoms are not deleted), the heuristic value is low, corresponding to the "easy" way. Furthermore, and perhaps more problematically, states which are closer to or further from the solution along the hard path are not accorded correspondingly better or worse values (see the discussion of the PUSH-BLOCK domain below).

To avoid such situtations, when a RGS is achieved in the connection phase of RRT-Plan, the atoms of the RGS are

locked so that any future action which deletes them is not considered. Additionally, any goal atoms which are locked in a parent node are also locked in its children. Importantly, this restriction is taken into account when calculating the atom cost estimates for the node (Eqn 1). States for which the final goal is accorded an infinite heuristic value are discarded. We define $h^+_{gsl}(s, gs)$ to be the length of the relaxed plan from s to the goal where actions which delete atoms in goal subset gs are disallowed. It is clear that:

$$h^+(s) \leq h^+_{gsl}(s, gs) \; \forall gs$$

This follows directly from the fact that the actions allowed in the goal subset locked relaxed plan are a subset of those allowed for the general relaxed plan. Reducing the number of available actions can only increase the length of the resulting plan.

Adapting the Search Parameters

We can extract several important bits of information from the growth (or failure to grow) of the tree. This information is used to adapt the parameters of the sub-planner.

Three pieces of information are collected regarding tree growth. Each is the result of counting connection failures of the sub-planner. First, we keep track of how many times a given node has failed to connect to q_{rand} in the expansion phase. This counter is added to the distance function for a given node, so that a node with a large failure counter is selected less frequently as a nearest neighbor (eventually it is no longer selected at all).

The planner also keeps two similar counters for the goal atoms. These are incremented each time the atom is included in the RGS but cannot be reached. The sub-planner search can fail for two reasons: the node expansion limit is reached, or the list of states to expand is exhausted. We track the number of occurrences of both failure modes for each atom, and the parameters for the local search are chosen based on maximum failure counts for the atoms in the RGS.

When the algorithm has failed several times to reach an atom because of the node expansion tolerance, that parameter is simply increased. When the list of states has been exhausted, the remedy is to decrease the aggressiveness of action pruning. There are currently three levels of action pruning:

- TARGET_HA - consider only actions which are helpful in achieving the current RGS.
- FULL_HA - consider all actions which are helpful in achieving the full goal.
- ALL_APPLICABLE - consider all actions which are applicable in a given state.

Where we mean helpful actions in the sense of (Hoffmann & Nebel 2001).

In the domains we have tested on, there is typically only one atom which is difficult to reach. For example in BlocksWorld, the atom representing the blocks at the base of the tower is often very difficult to achieve, especially in a way such that $h^+_{gsl} < \infty$. As the tree expansion continues, RRT-Plan fails several times to reach that atom, until eventually the search parameters are sufficiently relaxed. After the atom is achieved, the goal can usually be reached rapidly.

Note here the contrast to FF. On such problems, FF will try the fast Enforced Hill Climbing phase, which will fail. Then the slow GBFS phase will run, and eventually succeed in achieving the critical atom in such a way that the full goal is now easy. But the algorithm remains in the slow phase, and the "good" state does not have a particularly low heuristic value, so many more node expansions are required to complete the search.

A precise and optimal strategy for relaxing the search constraints remains a topic for further work. In the following results the node expansion tolerance began at 500 and increased 10% per failure. The action pruning began with TARGET_HA, relaxed to FULL_HA after 10 failures, and relaxed to ALL_APPLICABLE after 50 failures.

Problems Where Randomization Helps

Heuristic planners such as those described above (e.g. HSP and FF) use an approximation of the relaxed plan length as an estimate of the distance to the goal. FF uses a variety of additional techniques to prune the search space. These methods are generally quite effective, but are prone to failure in certain cases.

We now describe some of these situations with specific examples. Similar observations were made by (Helmert 2004). We show how randomization can deal effectively with these issues.

Issue with the h^+ heuristic

The h^+ heuristic function gives length of the relaxed plan from a state to the goal. The relaxed plan ignores the delete effects of actions. Since the goal cannot be achieved more rapidly by discarding delete effects, the h^+ function is *admissible*.

Unfortunately this heuristic is often somewhat uninformative. Indeed, in their original paper on HSP, (Bonet & Geffner 1999) argue that an approximation which is closer to a correct estimate of the true distance to the goal is more helpful than one which provides a strict lower bound. In any event, it often occurs that a large number of states are assigned an identical heuristic value. In the terminology of (Hoffmann 2002), this is called a *plateau*. When the search procedure reaches a plateau, it cannot make progress except by exhaustively examining states.

We can see an example of this in a modified logistics domain. Assume there is a long chain of locations, with one truck in the middle. The truck must deliver a different package to each end of the chain. In this case, the relaxed plan length heuristic assigns the same value to each state where one of the packages is not yet delivered. As a result, the heuristic is completely uninformative in terms of making progress towards either goal (in this case, the problem can still be solved because there are only as many states as links in the chain, but larger instances can be constructed with an exponential number of states).

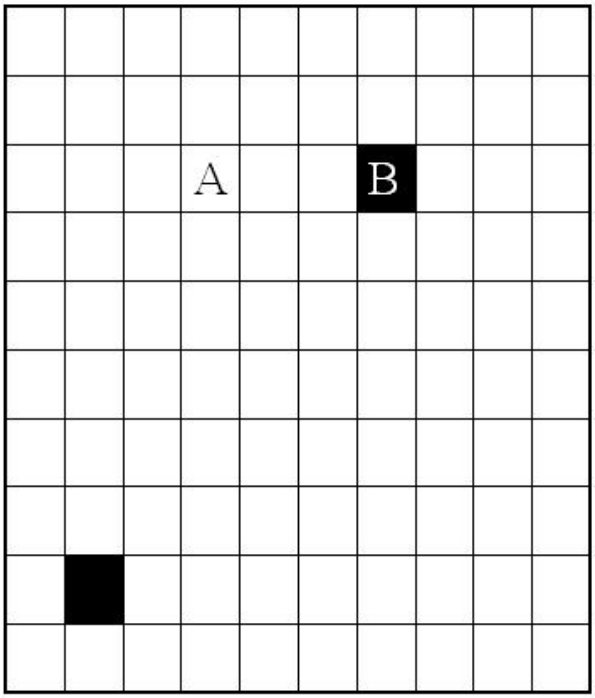

Figure 3: The PUSH-BLOCK domain. The black squares are blocks. A and B denote locations to which the blocks must be pushed. Any state in which the lower block is closer or farther away from A has the same evaluation, because the relaxed plan simply moves the block from B to A.

In this domain, RRT-Plan will succeed immediately: one goal atom or the other will be randomly selected. Achieving either of the goal atoms (while ignoring the other) is easily done, and once the first is achieved, the second is also easy.

A related problem can be understood by looking at a domain we have constructed called PUSH-BLOCK. Here we have a set of blocks positioned on a regular grid. Blocks can be pushed horizontally and vertically, but only one block can occupy a location. There is only one meaningful predicate, (OCCUPIED X Y), which indicates whether a grid position has a block in it (there are other predicates which just encode adjacency relationships between rows and columns). There are two actions, one corresponding to pushing a block vertically and the other to pushing a block horizontally. The goal is a set of locations which must be OCCUPIED.

Consider the situation shown in figure 3. The goal is to have blocks in positions A and B. Position B is already occupied and thus we need only move the block from the lower left corner to A. However, in the relaxed plan a short solution appears: move the block at B to A. Thus every state in which the lower block is nearer or closer to A has the same heuristic value of 3, which is the length of the relaxed plan. In other words, the h^+ heuristic becomes a bad evaluator of states for which the goal is nearby in the relaxed plan.

On the other hand, if we ignored the block at B, the problem would be trivial. The h^+ heuristic would give a perfect evaluation of states, and the lower block would move directly to A. This is exactly what RRT-Plan does by locking the goal atom representing the block at B. In general, the more goals and blocks considered by the heuristic, the worse the quality of the evaluation provided.

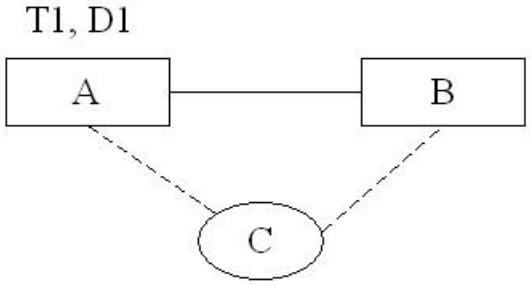

Figure 4: Problem encountered by FF in DriverLog domain. Truck and driver are both at A; goal is to move only the driver to B. Dotted lines indicate walking paths, solid line is unwalkable highway. The only action recognized as helpful by FF is to drive from A to B.

From the two examples above we may make an interesting observation about RRT-Plan. By choosing goal subsets at random and locking goal atoms already achieved, the algorithm is effectively imposing an artificial goal ordering on the problem. While such an imposition cannot make the solution easier in an absolute sense, it can often make the solution more obvious to the planner, by allowing it to prune the search space more aggressively.

Issue with helpful action pruning

FF uses the notion of helpful actions to prune the search space during enforced hill climbing. Helpful actions are those which have add effects which are present in the first fact level of the solution to the relaxed planning problem. Intuitively, these are the actions which provide the most direct means of achieving the most pertinent atoms.

(Hoffmann & Nebel 2001) give an example where helpful actions can prune all of the solutions to the problem. Their example is somewhat abstract; we'll present a situation that arises in one of the actual planning domains, DriverLog.

DriverLog is basically the same as Logistics, except there are only trucks, and the truck drivers can walk around as well as drive. (Hoffmann 2002) showed that FF can handle the logistics problems in polynomial time, so we might expect this domain to be similarly easy. However, consider the situation shown in figure 4.

Here a truck and driver are both at location A. Locations A and B are linked by a highway which can be driven across but not walked. A and B are also linked indirectly through C, and these paths can be walked by the driver. The goal is to have the truck at A, and the driver at B. The solution is clearly (WALK D1 A C) - (WALK D1 C B).

The relaxed plan graph will have only one level, because the first goal atom is achieved in the initial state and the sec-

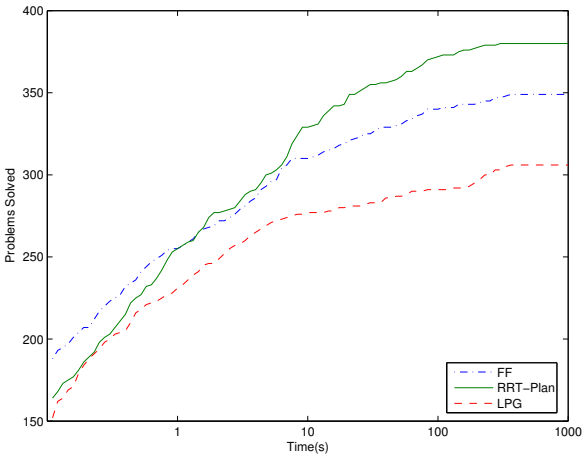

Figure 5: Number of plans solved as a function of time for FF and RRT-Plan. Time is on a logarithmic scale.

ond goal atom can be achieved with only one action. Thus the only helpful action considered is to drive from A to B.

RRT-Plan handles this case by randomly asserting (AT TRUCK A), and locking this atom when it is achieved. Now the relaxed plan solution will not be allowed to consider the DRIVE action, and so will select the correct WALK action instead. Note that RRT-Plan has no way of knowing in advance that (AT TRUCK A) should be selected, it simply chooses it after several iterations of trial and error.

Experimental Results

To validate RRT-Plan, we compared its performance with that of state-of-the-art planners FF and LPG (Gerevini & Serina 2002) on problems from the planning competitions from 1998, 2000, and 2002 (McDermott 2000; Bacchus 2001; Long & Fox 2003). We also used the STRIPS version of the Pipesworld domain from the 2004 competition.

We also used the PUSH-BLOCK domain described above. This is simply a 20x20 grid of positions which can be occupied by blocks. The blocks can be pushed left, right, up or down, but not into a location which already contains another block. The goal is simply a set of locations to which we must move blocks. We generated 20 domains, starting with one block/goal atom and moving up to 20.

Figure 5 shows the number of problems solved by the planners within a given time length, while Table 1 shows the number of problems solved by FF and RRT-Plan on several standard planning domains. These statistics were generated by allowing the planners to run for up to five minutes and recording the time to completion. Experiments were performed on a 3GHz Pentium 4 Linux machine with 2GB of RAM. Looking at the table, we see that RRT-Plan outperforms FF in several domains and is worse in only one, FreeCell. These results can be broken down as follows:

Rovers, Satellite, Logistics . In these domains, FF's fast Enforced Hill-Climbing phase is able to rapidly solve the problem. Because we use this same technique for the sub-planner, RRT-Plan will also rapidly solve the problem, after a small number of iterations. LPG also solves these problems easily.

Mystery, MPrime, FreeCell . The problems in these domains have a small number of goal atoms. This essentially cripples the randomization of RRT-Plan, because there are so few goal subsets to choose from. However, the search constraints will be relaxed until all the effort is expended by the sub-planner, thus RRT-Plan again becomes equivalent (modulo exact parameter settings) to FF. LPG does significantly better on the MPrime domain than both FF and RRT-Plan, and significantly worse on the FreeCell domain.

Blocks-World, DriverLog, Depot, Pipesworld, Push-Block. In these domains RRT-Plan seems to achieve consistently better results than FF. In the case of DriverLog and Push-Block, the reason should be clear from the discussion above.

For Depot and in particular Blocks-World, the cause is a bit more difficult to discern. These domains are characterized by strong goal orderings. However, the goal agenda technique used by FF (Koehler & Hoffmann 2000) is able to discover the goal ordering and provide it to the planner. We speculate that FF does not consider the full goal when achieving goal subsets, which cripples the Goal Added Deletion heuristic. RRT-Plan will attempt to reach the first goal atom and reject several states as unacceptable ($h^+_{gsl} = \infty$). Thereafter the search constraints are relaxed, and finally the difficult atom is achieved acceptably, after which the problem becomes easy.

The goal of RRT-Plan is to find solutions to difficult problems. To that end, we were prepared to accept suboptimal plan lengths. We expected the plans generated to be uniformly worse than those created by other planners. Surprisingly, this was not always the case, as shown by Table 3. Indeed, for some problems the plan generated by RRT-Plan is about half the length of FF's. Figure 6 shows the number of plans generated of a given length.

RRT-Plan is of course a randomized algorithm and therefore can produce different results on different runs. While we have not performed exhaustive testing, the numbers in Table 1 change only slightly (in one test suite we observed only 6 failures on Pipesworld, in another 1 failure in Push-Block). This is probably because the average completion time for some problems is around the 5 minute cutoff and therefore variation can mean the problem sometime registers as a failure.

We have performed systematic tests on the hard DriverLog (16-20) problems, and therefore the table lists means and standard deviations for plan length and completion time. These numbers were obtained by running RRT-Plan 100 times on each problem.

In comparing RRT-Plan to LPG, the most reasonable analysis is to look at the number of domains where one exceeds the other, rather than number of problems. With the exception of Push-Block, the score is seems to be tied: RRT-Plan wins in MPrime and loses in FreeCell, while in other domains performance is roughly equal. A greater number of problems and especially domains is needed to more rigor-

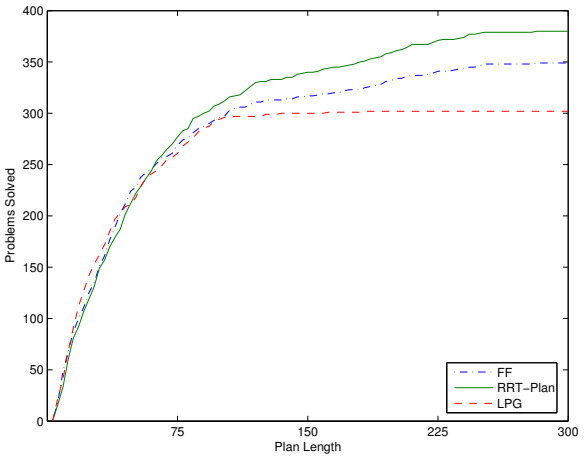

Figure 6: Plan length comparison for FF, LPG, and RRT-Plan.

ously compare performance of the planners.

Table 1: Performance of FF, RRT-Plan, and LPG on various domains. Entries list the number of problems the planner could not solve within five minutes of CPU time.

Domain	FF	RRT-Plan	LPG
Blocks World (35)	3	1	0
Driverlog (20)	5	0	0
Depot (22)	3	0	0
Freecell (80)	8	10	70
Logistics (63)	0	0	0
MPrime (35)	3	3	0
Mystery (30)	14	13	12
Pipesworld (50)	15	8	9
Rovers (20)	0	0	0
Satellite (20)	0	0	0
Push-Block (20)	15	0	19

Discussion

This paper presents a randomized algorithm for STRIPs planning. The strong emphasis on randomization is conceptually novel, compared to current state-of-the-art discrete planners. The algorithm is shown to exhibits competitive empirical performance on a number of standard domains. The algorithm is inspired by Rapidly exploring Random Trees, a concept borrowed from the domain of continuous space path planning. However, significant alteration of the underlying RRT concept must be made for it to work in the discrete domain.

In general, one of the most important advantages of randomized over deterministic algorithms is that they avoid systematic errors. Heuristic planners such as FF use several techniques which are generally powerful and effective, but can sometimes fail completely even in simple situations.

Table 2: Time to completion (seconds) on DriverLog problems 10-20. For problems 16-20, mean and std dev are given for RRT-Plan.

Problem	FF	RRT-Plan	LPG
driverlog-11	0.00	0.00	0.05
driverlog-12	0.41	0.02	0.17
driverlog-13	0.17	0.05	0.47
driverlog-14	0.21	0.05	0.13
driverlog-15	0.06	0.09	0.18
driverlog-16	-	$\mu = 15.7, \sigma = 6.4$	274.79
driverlog-17	-	$\mu = 5.0, \sigma = 1.6$	2.14
driverlog-18	-	$\mu = 2.7, \sigma = 0.6$	38.34
driverlog-19	-	$\mu = 34.4, \sigma = 16.5$	215.41
driverlog-20	-	$\mu = 12.7, \sigma = 5.6$	9.77

Table 3: Plan Length for DriverLog problems 10-20. For problems 16-20, mean and std dev are given for RRT-Plan.

Problem	FF	RRT-Plan	LPG
driverlog-11	24	25	16
driverlog-12	51	48	40
driverlog-13	35	35	49
driverlog-14	37	50	41
driverlog-15	47	49	39
driverlog-16	-	$\mu = 144, \sigma = 17$	181
driverlog-17	-	$\mu = 114, \sigma = 4.6$	95
driverlog-18	-	$\mu = 107, \sigma = 5.3$	83
driverlog-19	-	$\mu = 191, \sigma = 15.2$	159
driverlog-20	-	$\mu = 143, \sigma = 2.9$	94

We have described a number of such situations above, and shown that our randomized algorithm provides better robustness in such cases.

The goal of the original RRT formulation is to expand the tree so that it reaches a uniform sampling of the configuration space. Our original idea was to try to achieve a similar thing for the discrete space search - to grow the tree until it samples the reachable states uniformly. However, this approach met with a variety of difficulties.

The basic problem in the analogy between the continuous and discrete cases is as follows. In the continuous case the region from which the sub-planner can achieve the goal directly is typically of the same dimension as the entire space. In the discrete case, the region can be exponentially smaller, depending of course on the nature of the sub-planner. Therefore even if we achieved uniform coverage of the space - which is itself quite difficult to do - the algorithm would still require exponential time and memory.

Instead of attempting to choose states randomly from the space, RRT-Plan instead randomly samples from goal subsets. As the number of achieved goal subsets in the tree grows, it moves closer and closer to the complete goal. In this sense, it is best to think of RRT-Plan as searching through the space of *artificial goal orderings*. With this reformulation in mind, we can make an interesting analogy. Continuous RRTs are highly effective in problems with high dimensionality but where solutions are not scarce, but encounter difficulty in problems where there is only a thin bundle of solutions (such as navigation in a narrow corridor). Similarly, RRT-Plan is effective at choosing an artificial goal ordering, in problems where there is no natural goal ordering. This artificial ordering can speed up the search significantly. RRT-Plan can also deal with problems with natural goal orderings, when those orderings are discovered using the method of (Koehler & Hoffmann 2000). We imagine that the algorithm will fare poorly in situations with natural goal orderings which are not discovered. However, it would seem that such situations are relatively scarce.

While the selection of random goal orderings is at the heart of our approach, several additional techniques are required. Most prominently is the idea of goal subset locking. By pointing the h^+ heuristic at a goal subset and preventing it from deleting already achieved goals, the topology of the search landscape is simplified. Local minima and plateaus which would otherwise hamper the planner's progress are removed. Also, as the tree grows certain goal atoms are identified as problematic, and more effort is devoted to achieving them.

Striking the correct balance between exploration of the search space and exploitation is a standard question in AI planning. Previous heuristic search planners have focussed almost entirely on exploitation - moving in the direction of decreased heuristic value. Because there are situations where the heuristic function is systematically incorrect, it is often useful to include an element of exploration in the search. On the other hand, due to the size of the space, it would be inefficient to explore haphazardly. The random goal subset selection technique of RRT-Plan is a compromise between the two objectives.

Our results show RRT-Plan to be competitive with leading STRIPS planners. In particular, RRT-Plan can handle several types of problems which FF stumbles on, while performing only slightly worse than FF in other cases, such as when there are very few goal atoms. A main question for future work is how to extend RRT-Plan to deal with such cases. The answer to this question will likely involve finding actions that assert the goal atom(s) and using the preconditions of those actions as a new set of goals, from which to randomly sample. This could even involve growing another RRT from the goal which expands in the regression space (Bonet & Geffner 1999) of the problem.

A large body of other open questions involve how to apply RRT style randomization to the more sophisticated planning problems (such as those which involve temporal constraints, resource allocation, and fluents), and to what extent such techniques are effective. It is reasonable to hope that this line of research will be quite successful, inasmuch as the additional aspects of the planning problem make it more similar to continuous path planning in which RRTs are quite effective.

Acknowledgment

We would like to thank Joerg Hoffmann, Bernhard Nebel, Blai Bonet and Hector Geffner for making the source code to their planners available online.

References

Bacchus, F. 2001. The AIPS '00 planning competition. *AI Magazine* 22(3):47–56.

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, 1636–1642.

Bonet, B., and Geffner, H. 1999. Planning as heuristic search: New results. In Biundo, S., and Fox, M., eds., *Proc. 5th European Conf. on Planning*, 359–371. Durham, UK: Springer: Lecture Notes on Computer Science.

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

Fikes, R., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. In *IJCAI*, 608–620.

Gerevini, A., and Serina, I. 2002. LPG: A planner based on local search for planning graphs with action costs. In Ghallab, M.; Hertzberg, J.; and Traverso, P., eds., *AIPS*, 13–22. AAAI.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In *Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, 161–170.

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

Hoffmann, J. 2002. Local search topology in planning benchmarks: A theoretical analysis. In *Proceedings of*

the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-02). 379-387.

Knoblock, C. A. 1990. Learning abstraction hierarchies for problem solving. In Dietterich, T., and Swartout, W., eds., *Proceedings of the Eighth National Conference on Artificial Intelligence*. Menlo Park, California: AAAI Press.

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

Korf, R. E. 1985. Macro-operators: a weak method for learning. *Artif. Intell.* 26(1):35–77.

Kuffner, J. J., and LaValle, S. M. 2000. RRT-connect: An efficient approach to single-query path planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, 995–1001.

Kuffner, J.; Nishiwaki, K.; Kagami, S.; Inaba, M.; and Inoue, H. 2003. Motion planning for humanoid robots. In *Proc. 11th Intl Symp. of Robotics Research (ISRR 2003)*.

LaValle, S. M., and Kuffner, J. J. 2000. Rapidly-exploring random trees: Progress and prospects. In *Workshop on the Algorithmic Foundations of Robotics*.

LaValle, S. M. 1998. Rapidly-exploring random trees: A new tool for path planning. TR 98-11, Computer Science Dept., Iowa State University.

Long, D., and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *J. Artif. Intell. Res. (JAIR)* 20:1–59.

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, volume 2, 634–639. Anaheim, California, USA: AAAI Press/MIT Press.

McDermott, D. V. 2000. The 1998 AI planning systems competition. *AI Magazine* 21(2):35–55.

Morgan, S., and Branicky, M. S. 2004. Sampling-based planning for discrete spaces. In *Proc. IEEE/RSJ Intl. Conf. Intelligent Robots and Systems*.

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