7-13-2008

R* Search

Maxim Likhachev
University of Pennsylvania, maximl@seas.upenn.edu

Anthony Stentz
Carnegie Mellon University

R* Search

Abstract
Optimal heuristic searches such as A* search are widely used for planning but can rarely scale to large complex problems. The suboptimal versions of heuristic searches such as weighted A* search can often scale to much larger planning problems by trading off the quality of the solution for efficiency. They do so by relying more on the ability of the heuristic function to guide them well towards the goal. For complex planning problems, however, the heuristic function may often guide the search into a large local minimum and make the search examine most of the states in the minimum before proceeding. In this paper, we propose a novel heuristic search, called R* search, which depends much less on the quality of the heuristic function. The search avoids local minima by solving the whole planning problem with a series of short-range and easy-to-solve searches, each guided by the heuristic function towards a randomly chosen goal. In addition, R* scales much better in terms of memory because it can discard a search state-space after each of its searches. On the theoretical side, we derive probabilistic guarantees on the sub-optimality of the solution returned by R*. On the experimental side, we show that R* can scale to large complex problems.

R* Search

Maxim Likhachev
Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104
maximl@seas.upenn.edu

Anthony Stentz
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
axs@rec.ri.cmu.edu

Abstract

Optimal heuristic searches such as A* search are widely used for planning but can rarely scale to large complex problems. The suboptimal versions of heuristic searches such as weighted A* search can often scale to much larger planning problems by trading off the quality of the solution for efficiency. They do so by relying more on the ability of the heuristic function to guide them well towards the goal. For complex planning problems, however, the heuristic function may often guide the search into a large local minimum and make the search examine most of the states in the minimum before proceeding. In this paper, we propose a novel heuristic search, called R* search, which depends much less on the quality of the heuristic function. The search avoids local minima by solving the whole planning problem with a series of *short-range and easy-to-solve* searches, each guided by the heuristic function towards a randomly chosen goal. In addition, R* scales much better in terms of memory because it can discard a search state-space after each of its searches. On the theoretical side, we derive probabilistic guarantees on the sub-optimality of the solution returned by R*. On the experimental side, we show that R* can scale to large complex problems.

Introduction

A* search is a a provably optimal algorithm in terms of both, the solution quality as well as the amount of work (state computations) it has to do to in order to guarantee optimality of the solution (Pearl 1984). The provable optimality of the solution however requires the search to explore too many states for it to be able to solve large complex planning problems. A number of suboptimal heuristic searches have been proposed instead (Furcy 2004; Zhou and Hansen 2002; 2005; Likhachev, Gordon, and Thrun 2003). These searches give up the optimality of the solution guarantee but can often scale to much larger problems (Zhou and Hansen 2002; Rabin 2000; Gaschnig 1979; Likhachev, Gordon, and Thrun 2003; Likhachev and Ferguson 2008). The success of these searches though depends strongly on how well the heuristic function can guide them towards the goal state. This is perfectly explainable since the heuristic function is the only information they have about the direction they have to focuss their search efforts on in order to reach the goal state.

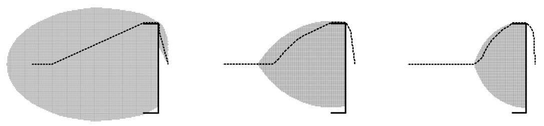

(a) A* (b) wA*, $w = 2$ (c) wA*, $w = 5$

Figure 1: A* and weighted A* dealing with the local minimum

The dependency on the heuristic function leads to a problem when it guides the search into a local minimum - a portion of the state-space from which there is no way to states with smaller heuristics without passing through states with higher heuristics. This problem exists in both discrete as well as discretized but originally continuous domains. It is easier though to visualize it in the latter ones. Figure 1 shows this for a path planning problem in 2D. In solid black are shown obstacles. The world is discretized into a 24-connected grid, meaning that the agent can transition to any cell that is one or two moves away. (This way the agent can move in 16 possible directions.) The cost between any two cells connected by an edge is proportional to the Euclidean distance between them if the transition does not go over an obstacle and is infinite otherwise. In light gray are shown all the states explored by A* (a), A* with heuristics multiplied by a factor w of 2 (b) and A* with heuristics multiplied by a factor w of 5 (c). The last two versions of A* are also often referred to as weighted A* (wA*) - a suboptimal version of A* that trades off the quality of the solution for efficiency. In dashed black are shown the paths found by these algorithms. The figure shows that compared to A*, weighted A* expands much fewer states but returns somewhat suboptimal solutions. However, because the obstacle presents a large local minimum for the heuristic function, all versions of A* have to expand all the states inside the cul-de-sac before they find a way around it. This remains true for any value of w.

In this paper, we propose a randomized version of A*, called R* that depends less on the quality of guidance of the heuristic function. Instead of executing a single search focused towards a goal state, R* tries to execute a series of short-range weighted A* searches towards goal states chosen at random and re-construct a solution from the paths produced by these searches. The main idea R* builds on is that rather than running each of its short-range searches until they find solutions, R* postpones the ones that do not find solutions easily and tries to construct the overall solution using only the results of searches that find solutions easily.

R* schedules and re-schedules short-range searches in such a way as to provide probabilistic guarantees on the sub-optimality of the overall solution it finds. In our theoretical analysis, we prove the existence of these guarantees for a simplified model of the state-space. We also show that the suboptimality guarantees hold with probability equal to 1 if R* does not perform randomized down-select of what short-range searches to schedule. In our experimental analysis, we show that R* can scale to large complex planning problems, can find solutions to such problems much more often than weighted A* search, and can minimize the cost of the found solutions much better than randomized motion planning algorithms, developed specifically for continuous domains.

Related Work

On one hand, R* is related to the family of randomized motion planners (Kavraki et al. 1996; Kuffner and LaValle 2000). These algorithms have gained tremendous popularity in the last decade. They have been shown to consistently solve impressive high-dimensional motion planning problems. In addition, these methods are simple, fast and general enough to solve a variety of motion planning problems. R* differs from these algorithms in several aspects. First, unlike R*, the randomized motion planners were designed specifically for continuous state-spaces. The exceptions are few very recent extensions of randomized motion planning to discrete state-spaces (Burfoot, Pineau, and Dudek 2006; Morgan 2004). Second, the current randomized planners are mainly concerned with finding any feasible path rather than minimizing the cost of the solution. Third, they provide no guarantees on the sub-optimality of the solution. R* tries to find the solutions with minimal cost and provides probabilistic guarantees on the quality of the solution. These two aspects of R* are important when solving planning problems for which the minimization of objective function is important.

On the other hand, R* falls into the category of heuristic searches such as A* (Pearl 1984) and its suboptimal variants (Furcy 2004; Zhou and Hansen 2002; 2005; Likhachev, Gordon, and Thrun 2003). These searches examine states in the search-space thoroughly and in a methodological manner and therefore return solutions that are often of much better quality than those found by randomized planners. In addition, they often provide sub-optimality bounds on the solution and some of them, such as weighted A*, can even be asked to find a solution whose sub-optimality does not exceed the specified bound. Unfortunately though, the performance of these searches depends largely on the quality of the heuristic function. Random scheduling of short-range searches in R* is designed to make the heuristic search less dependent on the quality of the heuristic function.

The most relevant to ours is a very recently and independently developed Randomized A* algorithm (Diankov and Kuffner 2007). Its major difference from our work is that it mainly targets continuous domains and does not provide the analysis of bounds on sub-optimality. Their work contains a number of interesting ideas including the use of statistical learning to learn the heuristic function in order to avoid tweaking its parameters.

```
1  select unexpanded state $s \in \Gamma$ (priority is given to states not labeled AVOID)
2  if path that corresponds to the edge $bp(s) \rightarrow s$ has not been computed yet
3    try to compute this path
4    if failed then label state $s$ as AVOID
5    else
6      update $g(s)$ based on the cost of the found path and $g(bp(s))$
7      if $g(s) > w\, h(s_{\text{start}}, s)$ label $s$ as AVOID
8  else //expand state $s$ (grow $\Gamma$)
9    let $SUCCS(s)$ be $K$ randomly chosen states at distance $\Delta$ from $s$
10   if goal state within $\Delta$, then add it to $SUCCS(s)$
11   for each state $s' \in SUCCS(s)$, add $s'$ and edge $s \rightarrow s'$ to $\Gamma$, set $bp(s') = s$
```

Figure 2: Singe iteration of R*

R* Algorithm

R* operates by constructing a small graph Γ of sparsely placed states, connected to each other via edges. Each edge represents a path in the original graph in between the corresponding states in Γ. In this respect, Γ is related to the graphs constructed by randomized motion planners (Kavraki et al. 1996; Kuffner and LaValle 2000). The difference is that R* constructs Γ in such a way as to provide explicit minimization of the solution cost and probabilistic guarantees on the suboptimality of the solution. To achieve these objectives, R* grows Γ in the same way A* grows a search tree.

At every iteration, R* selects the next state s to expand from Γ (see figure 2). While normal A* expands s by generating all the immediate successors of state s, R* expands s by generating K states residing at some distance Δ from s (lines 8-11). The distance Δ is some metric that measures how far two states are from each other. This metric can be domain dependent or independent such as difference in heuristic values of two states. Either way the metric should be applicable to whatever domain we are trying to solve, be it a discrete one or continuous. If a goal state is within Δ from state s then it is also generated as the successor of s. R* grows Γ by adding these successors of s and edges from s to them.

A path that R* returns is a path in Γ from the start state to the goal state. This path consists of edges in Γ. Each such edge, however, is actually a path in the original graph. Finding each of these (local) paths may potentially be a challenging planning task. R* postpones finding these paths until necessary and tries to concentrate on finding the paths that are easy to find instead. It does this by labeling the states to which it can not find paths easily as AVOID states. Initially, when generating K successors, none of these states are labeled as AVOID - R* does not try to compute paths to *all* of the generated states. Instead, only when state s is selected for expansion does R* try to compute a path from the predecessor of s, stored in the backpointer of s $bp(s)$, to state s

(lines 2-7).

R* uses the weighted A* search with heuristics inflated by w to compute local paths. It stops the search, however, if it fails to find the path easily. (Different heuristics can be used to establish this such as time limit or number of state expansions. In our experiments, we used a threshold of 100 expansions to detect that a path can not be found easily.) If it does fail, then R* labels state s as AVOID state since it assumes that it will be time-consuming to find a path to state s. If the weighted A* search does find a path, then the cost of the found path can be used to assign the cost of the edge $bp(s) \rightarrow s$. The cost of the edge and the cost of the best path from s_{start} to $bp(s)$, stored in $g(bp(s))$, can then be used to update $g(s)$ in the same way A* updates g-values of states.

R* provides probabilistic guarantees on the suboptimality of the solution. The uncertainty in the guarantee is purely due to the randomness of selecting K successors during each expansion. For a given graph Γ, on the other hand, R* can state that the found path is no worse than w times the cost of an optimal path that uses only the edges in Γ.

Suppose $g(s) \leq wh(s_{\text{start}}, s)$. Then the cost of the found path from s_{start} to s via the edges in Γ is clearly no worse than w times the cost of an optimal path, since $h(s_{\text{start}}, s)$ is supposed to be no more than the cost of the optimal path. Suppose now $g(s) > w\,h(s_{\text{start}}, s)$. This means that a path from s_{start} to s may not be w suboptimal. To prove otherwise, similarly to weighted A*, R* needs to expands all the states s' in Γ with $f(s') = g(s') + w\,h(s') \leq g(s) + w\,h(s) = f(s)$. Expanding all of these states, however, is computationally expensive, because some of these states are labeled AVOID and therefore require the computation of hard-to-find local paths to them. Moreover, it may even be unnecessary to use state s. For a given w, there often exist a wide spectrum of solutions that satisfy w suboptimality - some are easier to find than others. Therefore, R* considers the states s with $g(s) > wh(s_{\text{start}}, s)$ as the states it should also avoid expanding. It labels these states as AVOID (line 7).

To provide the suboptimality guarantees and minimize solution costs while avoiding as much as possible the states labeled AVOID, R* selects states for expansion in the order of smaller $f(s) = g(s) + w\,h(s)$), same as in weighted A*. However, it selects these states from the pool of states not labeled AVOID first. Only when there are no more such states left, R* starts selecting AVOID states (in the same order of f-values).

Actual Implementation Details The pseudocode of the R* algorithm is given in figure 3. The algorithm first goes through the initialization of variables. The g-values are estimates of the distance from the start state to the state in question, same as in normal A* search. bp-values are backpointers in graph Γ that can be used to backtrack the solution after the search terminates. k-values are priorities used to select states for expansion from *OPEN* - the list of states in Γ that have not been expanded yet. A state with the minimum priority is always selected for expansion first. Priorities are two-dimensional values and are compared according to the lexicographical ordering (first dimension is compared first, and the second is used to break ties). Whenever a state is

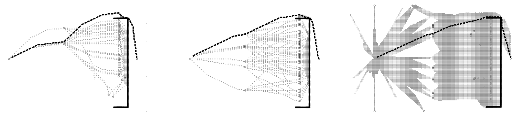
```
1  procedure UpdateState($s$)
2  if ($g(s) > w\,h(s_{\text{start}}, s)$ OR
        ($path_{bp(s),s} = \mathbf{null}$ AND $s$ is labeled AVOID))
3     insert/update $s$ in OPEN with priority $k(s) = [1,\ g(s) + w\,h(s, s_{\text{goal}})]$;
4  else
5     insert/update $s$ in OPEN with priority $k(s) = [0,\ g(s) + w\,h(s, s_{\text{goal}})]$;

6  procedure ReevaluateState($s$)
7  [$path_{bp(s),s}, c_{low}(path_{bp(s),s})$] = TrytoComputeLocalPath($bp(s)$, $s$);
8  if ($path_{bp(s),s} = \mathbf{null}$ OR $g(bp(s)) + c_{low}(path_{bp(s),s}) > w\,h(s_{\text{start}}, s)$)
9     $bp(s) = \arg\min_{s' | s \in SUCCS(s')}(g(s') + c_{low}(path_{s',s}))$;
10    label $s$ as AVOID state;
11 $g(s) = g(bp(s)) + c_{low}(path_{bp(s),s})$;
12 UpdateState($s$);

13 procedure RandomizedAstar()
14 $g(s_{\text{goal}}) = \infty$, $bp(s_{\text{goal}}) = bp(s_{\text{start}}) = \mathbf{null}$, $k(s_{\text{goal}}) = [1, \infty]$;
15 $OPEN = CLOSED = \emptyset$;
16 $g(s_{\text{start}}) = 0$;
17 insert $s_{\text{start}}$ into OPEN with priority $k(s_{\text{start}}) = [0, w\,h(s_{\text{start}}, s_{\text{goal}})]$;
18 while ($k(s_{\text{goal}}) \geq \min_{s' \in OPEN} k(s')$ AND $OPEN \neq \emptyset$)
19    remove $s$ with the smallest priority from OPEN;
20    if $s \neq s_{\text{start}}$ AND $path_{bp(s),s} = \mathbf{null}$
21       ReevaluateState($s$);
22    else //expand state $s$
23       insert $s$ into CLOSED;
24       let $SUCCS$ be the set of $K$ randomly chosen states at distance $\Delta$ from $s$
25       if distance from $s_{\text{goal}}$ to $s$ is smaller than or equal to $\Delta$
26          $SUCCS(s) = SUCCS(s) \cup \{s_{\text{goal}}\}$;
27       $SUCCS(s) = SUCCS(s) - SUCCS(s) \cap CLOSED$
28       for each state $s' \in SUCCS(s)$
29          [$path_{s,s'}, c_{low}(path_{s,s'})$] = [$\mathbf{null}, h(s, s')$];
30          if $s'$ is visited for the first time
31             $g(s') = \infty$, $bp(s') = \mathbf{null}$;
32          if $bp(s') = \mathbf{null}$ OR $g(s) + c_{low}(path_{s,s'}) < g(s')$
33             $g(s') = g(s) + c_{low}(path_{s,s'})$; $bp(s') = s$;
34             UpdateState($s'$);
```

Figure 3: The pseudocode of R*

labeled AVOID, the first dimension of its priority is set to 1. Otherwise, it is 0. This way, the states labeled AVOID are only selected for expansion if there are no states not labeled AVOID left to expand. The second dimension of the priority $k(s)$ is $f(s) = g(s) + w\,h(s)$, where h-values are heuristic values and must be consistent (Pearl 1984).

As in weighted A*, R* expands states until the priority of the goal state is smaller than the smallest priority in *OPEN*. The lines 22-34 correspond to a normal expansion of state s. It generates K random successors of state s, that haven't been expanded (closed) previously and goal state if within Δ from s. For each generated state s', it then sets $path_{s,s'} = \mathbf{null}$ to represent that no path from s to s' has been found yet. The cost of the edge $s \rightarrow s'$ is therefore set to the heuristic estimate of the distance, $c_{low}(path_{s,s'}) = h(s, s')$, which is an admissible estimate. Finally, same as in (weighted) A*, R* tries to decrease the g-value of state s if it has been already generated previously.

If R* selects a state s for expansion and the path to s from its parent $bp(s)$ in Γ has not been computed yet, then R* tries to compute this path first by calling the function ReevaluateState(s) on line 21. If path is found, R* updates the cost $c_{low}(path_{bp(s),s'})$ of the edge $bp(s) \rightarrow s$ based on



(a) R*, $w = 2$ (b) R*, $w = 1.5$ (c) R*, $w = 1$

Figure 4: R* dealing with the local minimum

the cost of the found path. If not found, weighted A* search is supposed to return the smallest (un-inflated) f-value of a state in its queue which can be used to set the edge cost $c_{low}(path_{s,s'})$ to an improved estimate of the path. Depending on whether R* successfully finds the path and how costly the path is, R* labels s as AVOID or not (described above). If it does set the state as AVOID, it re-computes the best predecessor of s (in case there are multiple predecessors) and sets $bp(s)$ accordingly. Afterwards, $g(s)$ is updated based on the g-value of $bp(s)$ and the cost of the edge in between $bp(s)$ and s.

After the while loop of R* terminates, the solution can be re-constructed by following backpointers bp backwards starting at state s_{goal} until s_{start} is reached. The path is given in terms of edges in R*, but each edge is guaranteed to have a local path computed. Thus, the solution in the original graph can be re-constructed.

Example

Figure 4 shows three runs of R* with different values of w on the same 24-connected gridworld as in figure 1. In all three runs, during each expansion, $K = 36$ successor cells were generated at random at a distance $\Delta = 60$ cells (the overall size of the grid was 200 by 200 cells). The small circles in these figures, represent those states in Γ to which paths have been computed. So, these states do not include the states that were labeled AVOID or states to which R* has not tried to compute paths. Thus, for example, the figure shows that R* has never computed a path in between states that lie on the opposite sides of the obstacle, as it would have involved running a weighted A* search for a (relatively) long period of time. All the cells in light gray are states expanded by local weighted A* searches R* executed. These cells are discarded after each local weighted A* search and therefore do not take up any memory (which is important when planning in higher dimensional state-spaces).

Compared to figure 1, figure 4 shows that the exploration done by R* with $w = 2$ and $w = 1.5$ is much sparser than that of the weighted A* search with any w. The quality of the returned solutions, on the other hand, is approximately the same and even better than the quality of the solution returned by weighted A* with $w = 5$. Also, as w decreases, the exploration performed by R* becomes denser in order to satisfy the probabilistic suboptimality bound.

Theoretical Analysis

This section first presents few basic properties about the algorithm. It then presents several theorems that come out of our analysis of the probabilistic guarantees on the suboptimality bound of the solution returned by the algorithm. The full formal analysis of the algorithm can be found in (Likhachev and Stentz 2008).

Basic Properties The first theorem shows that the algorithm is guaranteed to terminate.

Theorem 1 *No state s is expanded (lines 22-34) more than once, and the algorithm is guaranteed to terminate whenever the state-space is finite or the cost of an optimal path from s_{start} to s_{goal} is finite.*

In the next theorem, we use $\pi^{\Gamma}_{bp}(s_{\text{start}}, s)$ to denote the path from s_{start} to s by backtracking the backpointers starting at s. $c(\pi^{\Gamma}_{bp}(s_{\text{start}}, s))$ is used to denote the actual cost of this path. We also use $\pi^{\Gamma}_{opt}(s_{\text{start}}, s)$ to denote the path in graph Γ that is optimal assuming the cost of each edge in Γ is equal to its c_{low} value. Finally, $c^*(\pi^{\Gamma}_{opt}(s_{\text{start}}, s))$ is the cost of this path using costs of optimal transitions between any two consecutive states in this path rather than c_{low} values.

Theorem 2 *At line 18, for any state $s \in \Gamma$ it holds that $c(\pi^{\Gamma}_{bp}(s_{\text{start}}, s)) \leq g(s)$. Moreover, if ($g(s) + w\,h(s, s_{\text{goal}}) \leq g(u) + w\,h(u, s_{\text{goal}})\ \forall u \in$ OPEN), it holds that $g(s) \leq w\,c^*(\pi^{\Gamma}_{opt}(s_{\text{start}}, s))$.*

From the above theorem it follows that when the algorithm terminates the cost of the found path from s_{start} to s_{goal} is no more than $g(s_{\text{goal}})$, which in turn is no more than $w\,c^*(\pi^{\Gamma}_{opt}(s_{\text{start}}, s_{\text{goal}}))$. The reason is that when the algorithm terminates either *OPEN* is empty, in which case $\min_{u \in OPEN} g(u) + w\,h(u, s_{\text{goal}}) = \infty$, or $k(s_{\text{goal}}) < \min_{u \in OPEN} k(u)$. And the latter condition means that either the first element of $k(s_{\text{goal}})$ is 0, which means that $g(s_{\text{goal}}) \leq w\,h(s_{\text{start}}, s_{\text{goal}}) \leq w\,c^*(\pi^{\Gamma}_{opt}(s_{\text{start}}, s_{\text{goal}}))$ or $g(s_{\text{goal}}) + w\,h(s_{\text{start}}, s_{\text{goal}}) < \min_{u \in OPEN}(g(u) + w\,h(u, s_{\text{goal}}))$.

If every time a state is expanded, the algorithm generates all of the states that lie at distance Δ from it, then Γ is guaranteed to contain an optimal state from s_{start} to s_{goal} and therefore the above theorem implies that the algorithm is guaranteed to return a path whose cost is no more than w times the cost $c^*(s_{\text{start}}, s_{\text{goal}})$ of this optimal path. This is summarized in the theorem below.

Theorem 3 *Suppose on line 24 R* always generates all of the states that lie at distance Δ from s. Then upon termination, R* returns a path whose cost is no more than $g(s_{\text{goal}})$ which, in turn, is no more than w times the cost of an optimal path from s_{start} to s_{goal}. That is, $c(\pi^{\Gamma}_{bp}(s_{\text{start}}, s_{\text{goal}})) \leq g(s_{\text{goal}}) \leq w\,c^*(s_{\text{start}}, s_{\text{goal}})$.*

Analysis of the Confidence on suboptimality bound For the purpose of this analysis, we will assume that all edges in the graph have unit costs and each state s has M states lying at distance Δ edges from it. We will also assume that every time K states are generated by search on line 24, they have not been encountered previously by search. This assumption is correct when the search-space is a tree with a single or multiple goal states. The tree model of a search-space has been commonly used for the statistical analysis of A*-like searches (Pearl 1984; Pohl 1977; Gaschnig 1979). Our assumption is also approximately correct if K is negligibly small in comparison to the number of states that lie at distance Δ from state s that is being expanded.

Let us introduce the following tree, denoted by Γ^M. The root of the tree is s_{start} and the successors of each node s' in the tree are all M successors lying at distance Δ edges from state s' in the original graph. We define $N_{l,w}$ to be the number of distinct paths $\pi^{\Gamma^M}(s_{\mathrm{start}}, v)$ in Γ^M such that they satisfy two conditions: (a) a goal state lies within Δ edges from v in the original graph and (b) $c^*(\pi^{\Gamma^M}(s_{\mathrm{start}}, v)) + c^*(v, s_{\mathrm{goal}}) \leq w\, c^*(s_{\mathrm{start}}, s_{\mathrm{goal}})$, where $c^*(\pi^{\Gamma^M}(s_{\mathrm{start}}, v))$ is the cost of the path in Γ^M using optimal transitions between any two consecutive states in this path. The condition (b) means that path $\pi^{\Gamma^M}(s_{\mathrm{start}}, v)$ is w-suboptimal if following optimal paths in between every pair of states on this path. (Remember that path $\pi^{\Gamma^M}(s_{\mathrm{start}}, v)$ is given as a sequence of states from the tree Γ^M.) The probabilistic guarantee on suboptimality bound will use $N_{l,w}$. While $N_{l,w}$ is domain dependent, it can be estimated on smaller instances of the same or simpler (e.g., 2D search) but similar problems. It can also be just set to some small number to give a crude estimate on the probabilistic bound.

We define a K random walk on any graph G starting with any state s_{start} as a process of iteratively building a tree Γ^K of depth m in the following way: its root is state s_{start}; the successors of any state $s' \in \Gamma^K$ are K randomly selected successors of state s' in G. ith step of a K random walk is defined to be a process of generating all states in Γ^K that will reside at the depth of i from the root of Γ^K. Thus, after the 0th step of the K random walk, Γ^K consists of only s_{start}; after the 1st step of the K random walk, Γ^K consists of s_{start} and K randomly chosen successors of s_{start} from the graph G; after the 2nd step, Γ^K is grown further to contain an additional K^2 states, that are randomly chosen K successors of K states added in the previous step. This K random walk is used to analyze the way R* constructs its graph Γ. The next two theorems prove few properties of the K random walk we have just introduced (these results are independent of the R* algorithm).

Theorem 4 *Consider a tree with constant branching factor of M and N_l goal states at depth l distributed uniformly. A K random walk starting at the root s_{start} of this tree generates at least one goal state s_{goal} at lth step with the probability of $1 - \prod_{i=0}^{N_l - 1} \frac{M^l - K^l - i}{M^l - i}$ if $N_l \leq M^l - K^l$ and 1 otherwise.*

The proof to the above theorem is based on the fact that a K random walk executed for l steps generates a tree Γ^K with M^l leaves. We need to compute the probability that at least one of these leaves is one of N_l goal states. This probability is one minus the probability that by selecting at random N_l goal states out of M^l states, none of K^l leaves of Γ^K are selected. The latter probability follows the hypergeometric distribution as given in the theorem. The next theorem extends this result to the case when goal states lie uniformly in between depths m and l, rather than at a single depth l.

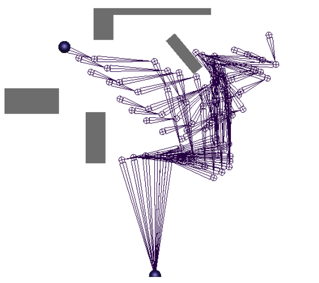
Theorem 5 *Consider a tree with constant branching factor of M and $N_{\leq l}$ goal states distributed uniformly in between depths m and l (including the depths m and l) of the tree. A K random walk starting at the root s_{start} of this tree generates at least one goal state s_{goal} at less than or equal to l steps with the probability of 1 if $K = M$ and $N_{\leq l} > 0$, the probability of at least $1 - e^{-\frac{K^l}{l-m+1}}$ if $K < M$ and $N_{\leq l} > M^l$, and the probability of at least $\min(1 - \prod_{i=0}^{N_{\leq l}-1} \frac{M^l - K^l - i}{M^l - i}, 1 - e^{-\frac{K^{l-1}(M-K)}{l-m}})$ otherwise.*

The theorem above is the main result used to derive the probabilistic guarantees for the suboptimality bound of R* as given in the next theorem.

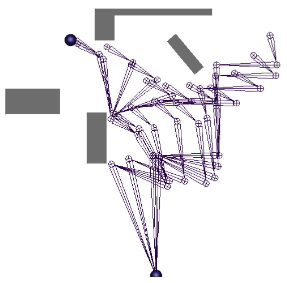
Theorem 6 *The probability that a particular run of R* results in a path whose cost is no more than w^2 times the cost l of an optimal path (that is, $c(\pi_{bp}^{\Gamma}(s_{\mathrm{start}}, s_{\mathrm{goal}})) \leq w^2\, c^*(s_{\mathrm{start}}, s_{\mathrm{goal}})$) is 1 if $K = M$ or $l \leq \Delta$. Otherwise, it is at least $1 - e^{-\frac{K^H}{H-L+2}}$ if $N_{l,w} > M^H$ and $\min(1 - \prod_{i=0}^{N_{l,w}-1} \frac{M^H - K^H - i}{M^H - i}, 1 - e^{-\frac{K^{H-1}(M-K)}{H-L+1}})$ if $N_{l,w} \leq M^H$, where $L = \lfloor \frac{l}{\Delta} \rfloor$ and $H = \lfloor \frac{w\,l}{\Delta} \rfloor$.*

Sketch of the proof: In case $K = M$, the proof follows directly from theorem 3. In case $l \leq \Delta$, the proof is also trivial since a goal state will be generated during the very first expansion, namely the expansion of s_{start}. The proof for the other cases uses theorem 5 to compute a lower bound on the probability that the graph Γ generated by R* contains at least one path $\pi^{\Gamma}(s_{\mathrm{start}}, s_{\mathrm{goal}})$ such that $c^*(\pi^{\Gamma}(s_{\mathrm{start}}, s_{\mathrm{goal}})) \leq w\, c^*(s_{\mathrm{start}}, s_{\mathrm{goal}})$. This bound is also a lower bound on $P(c(\pi_{bp}^{\Gamma}(s_{\mathrm{start}}, s_{\mathrm{goal}})) \leq w^2\, c^*(s_{\mathrm{start}}, s_{\mathrm{goal}}))$ because according to the termination criterion of R* and theorem 2, $c(\pi_{bp}^{\Gamma}(s_{\mathrm{start}}, s_{\mathrm{goal}})) \leq w\, c^*(\pi_{opt}^{\Gamma}(s_{\mathrm{start}}, s_{\mathrm{goal}}))$ and therefore $c(\pi_{bp}^{\Gamma}(s_{\mathrm{start}}, s_{\mathrm{goal}})) \leq w\, c^*(\pi^{\Gamma}(s_{\mathrm{start}}, s_{\mathrm{goal}})) \leq w^2\, c^*(s_{\mathrm{start}}, s_{\mathrm{goal}})$.

The derivation of the lower bound on the probability that Γ contains at least one w-suboptimal path uses the fact that the number of paths satisfying the conditions (a) and (b) is $N_{l,w}$ and then uses theorem 5 to analyze the probability that the following tree Γ' contains such path. Γ' of depth H is defined as follows: the root of the tree is s_{start}; for each non-leaf state s in Γ', its successors are the same as in Γ (generated at random on line 24), if s was expanded by R*, and are a new set of successors generated according to line 24, if s was not expanded by R*. The process of generating Γ' is a K random walk on the tree Γ^M. Theorem 5 gives us a lower bound on the probability that Γ' contains a w-suboptimal path. Similar to how weighted A* gives w-suboptimality guarantee, R* can also guarantee that the cost of the path returned by it is at most w times the cost of that w-suboptimal path. This gives the desired probabilistic bound on R* returning an w^2-suboptimal solution.



(a) motion generated by RRT (cost=91)



(b) motion generated by R* (cost=49)

Figure 5: Motions generated for a simulated 6 DOF robot arm after 30 secs of planning.

Experimental Analysis

We evaluated the performance of R* on simulated 6 (figure 5) and 20 degree of freedom (DOF) robotic arms against ARA* (Likhachev, Gordon, and Thrun 2003), which is an efficient execution of a series of weighted A* searches with gradually decreasing w, and RRT-based motion planner (Kuffner and LaValle 2000). In these experiments, the base of the arm is fixed, and the task is to move its end-effector to the goal (small circle on the left) while navigating around obstacles (indicated by grey rectangles). The arm is allowed to intersect itself. Initially, the arm is in the right-most configuration. An action is defined as a change of a global angle of any particular joint The cost of each action is 1. We discretize the workspace into 50 by 50 cells. Our heuristic function is a distance from each cell to the cell containing the goal while taking into account that some cells are occupied by obstacles. In order for the heuristic not to over-estimate true costs, joint angles are discretized so as to never move the end-effector by more than one cell in a single action. The resulting state-space is over 3 billion states for a 6 DOF robot arm and over 10^{26} states for a 20 DOF robot arm, and memory for states is allocated on demand.

If all obstacles were adjacent to walls, our heuristic would nearly always direct the robot arm in a more or less correct direction. As a result, the local minima for weighted A* search would have been small and it would have been able to compute solutions fast (see (Likhachev, Gordon, and Thrun 2003)). In our more general setting, however, obstacles can appear anywhere in the environment. This can cause large local minima for the heuristic function since the robot arm may often not be able to reach the desired goal location following the heuristic function. For example, the heuristic function in figure 5 advocates going above the free-floating obstacle, whereas the robot arm can only go underneath it. This large local minima in the heuristic function make the weighted A* search (and consequently ARA*) quickly run out of memory trying to find a way out of these minima by carefully examining all states in them. In contrast, the algorithms that perform sparse exploration of state-space such as RRT and R* can handle these scenarios well. This is expressed in the results (Table 1) that show that ARA* solves much fewer environments than either RRT or R*.

Figure 5 compares the motion generated by R* to the motion generated by RRT for a 6 DOF robot arm. The cost of the motion generated by RRT is 91, while the cost of the motion found by R* is 49.[1] We have also ran 65 runs for the environments with 15 randomly placed obstacles of size 2 by 2. The robot arm had 20 DOFs. The results are reported in table 1 (with 95% confidence intervals when appropriate). In all the experiments, ARA*, RRT and R* were all run for 30 secs.[2] (Each state expansion was slow due to forward kinematics and collision checking for every generated successor configuration of a 20DOF arm.) ARA* was executed with initial $w = 10$ and the decrease of 0.2. RRT was executed in anytime fashion, meaning that after it found its first solution, it continued to grow the tree and whenever possible to improve the solution. In addition, following the suggestion in (Ferguson and Stentz 2006), we controlled the growth of the tree and pruned away the states that were clearly irrelevant to improving the solution based on their heuristic values and the cost of the current solution. 30 seconds were enough to execute R* several times. R* was always executed with $w = 10.0$. In fact, each execution was limited to 10 seconds, even if it did not find the solution within this time period. R* returned the best solution it found across its multiple runs.

The metric used by R* to compare the distance in between states against Δ was set to be the maximum difference in x, y coordinates of the end-effector positions of two states in question. The actual value of the parameter Δ was set to 20. Thus, a goal state (a single state representing *any* configuration with the desired end-effector) was generated and added to the set $SUCCS(s)$ whenever its end-effector was within Δ distance from the end-effector of s, the state that was being expanded (line 26 in figure 3).

While both R* and RRT were able to solve about the same number of environments (we are not sure whether the ones that were not solved were solvable at all), ARA* solved only half of the environments solved by RRT and R*. On the other hand, the solutions returned by ARA* were better than the ones returned by RRT. The solutions by R* were of the same quality as the ones returned by ARA* (Table 1(b)) and significantly better than the solutions returned by RRT (Table 1(c)). The amount of work done by all algorithms was about the same in terms of the number of expansions (the columns labeled exp) and exactly the same in terms of runtime - 30 secs per environment.

Typically, planning for mobile and articulated robots such as the ones in our experiments gets harder as non-holonomic constraints become more pronounced. One of the intents of our experiments was to show that R* can handle these constraints efficiently. It does so by postponing very difficult

[1]While sometimes paths generated by RRT can be smoothed by post-processing, smoothing is limited to homotopic paths. It also relies on Jacobians which can be hard to derive for non-trivial domains and impossible to derive for domains modeled by weighted graphs.

[2]The motivation for our work was planning for autonomous robots, where planning is typically done in real-time and therefore the available planning time is in seconds. Hence, was our choice of 30 secs as opposed to minutes.

	solved
ARA*	17
RRT	33
R*	38

(a) number of solved

	exp	cost	overhead in cost
ARA*	29,785	61.9	2.8% (± 8.6%)
R*	30,625	62.1	0.0% (± 0.0%)

(b) ARA* vs. R* (average over solved by both)

	exp	cost	overhead in cost
RRT	31,382	71.5	18.3% (± 10.9%)
R*	32,935	61.9	0.0% (± 0.0%)

(c) RRT vs. R* (average over solved by both)

Table 1: Experimental Results. The numbers in parentheses are 95% confidence intervals.

cases and trying first to find a solution for which it is easy to connect subgoals. In particular, R* postpones searches that need to connect hard-to-connect subgoals and tries to solve the whole problem by stitching together solutions to easier cases, where heuristic function has shallow minima. Only when absolutely necessary (to satisfy the probabilistic bound) does R* goes to evaluating hard cases (AVOID nodes). The priority function of R* makes it always favor working on easy cases first.

Acknowledgements

This work was sponsored by the U.S. Army Research Laboratory, under contract Robotics Collaborative Technology Alliance (contract number DAAD19-01-2-0012). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government.

References

Burfoot, D.; Pineau, J.; and Dudek, G. 2006. RRT-plan: a randomized algorithm for strips planning. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.

Diankov, R., and Kuffner, J. 2007. Randomized statistical path planning. In *Proceedings of IEEE/RSJ 2007 International Conference on Robots and Systems (IROS)*.

Ferguson, D., and Stentz, A. 2006. Anytime RRTs. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*.

Furcy, D. 2004. *Chapter 5 of Speeding Up the Convergence of Online Heuristic Search and Scaling Up Offline Heuristic Search*. Ph.D. Dissertation, Georgia Institute of Technology.

Gaschnig, J. 1979. Performance measurement and analysis of certain search algorithms. Tech. Rep. CMU-CS-79-124, Carnegie Mellon University.

Kavraki, L.; Svestka, P.; Latombe, J.-C.; and Overmars, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12(4):566–580.

Kuffner, J., and LaValle, S. 2000. RRT-connect: An efficient approach to single-query path planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Likhachev, M., and Ferguson, D. 2008. Planning long dynamically-feasible maneuvers for autonomous vehicles. In *Proceedings of Robotics: Science and Systems (RSS)*.

Likhachev, M., and Stentz, A. 2008. R* search: The proofs. Tech. Rep., University of Pennsylvania, Philadelphia, PA.

Likhachev, M.; Gordon, G.; and Thrun, S. 2003. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems (NIPS) 16*. Cambridge, MA: MIT Press.

Morgan, S. 2004. *Sampling-based planning for discrete spaces*. Ph.D. Dissertation, Case Western Reserve University.

Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

Pohl, I. 1977. Practical and theoretical considerations in heuristic search algorithms. In Elcock, E. W., and Michie, D., eds., *Machine Intelligence 8*, 55–72. New York: Wiley.

Rabin, S. 2000. A* speed optimizations. In DeLoura, M., ed., *Game Programming Gems*, 272–287. Rockland, MA: Charles River Media.

Zhou, R., and Hansen, E. A. 2002. Multiple sequence alignment using A*. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. Student abstract.

Zhou, R., and Hansen, E. A. 2005. Beam-stack search: Integrating backtracking with beam search. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 90–98.