2011 IEEE/RSJ International Conference on
Intelligent Robots and Systems
September 25-30, 2011. San Francisco, CA, USA

Sampling Heuristics for Optimal Motion Planning in High Dimensions

Baris Akgun and Mike Stilman

Abstract— **We present a sampling-based motion planner that improves the performance of the probabilistically optimal RRT* planning algorithm. Experiments demonstrate that our planner finds a fast initial path and decreases the cost of this path iteratively. We identify and address the limitations of RRT* in high-dimensional configuration spaces. We introduce a sampling bias to facilitate and accelerate cost decrease in these spaces and a simple node-rejection criteria to increase efficiency. Finally, we incorporate an existing bi-directional approach to search which decreases the time to find an initial path. We analyze our planner on a simple 2D navigation problem in detail to show its properties and test it on a difficult 7D manipulation problem to show its effectiveness. Our results consistently demonstrate improved performance over RRT*.**

I. INTRODUCTION

Motion planning for a robot must consider both optimality of the plan and efficiency of the algorithm. The two goals are particularly important and challenging in high-dimensional configuration spaces such as those of robot arms and mobile manipulators. A common approach to this challenge is to find a suboptimal plan and refine in the remaining time that is allocated to the planning process. This paper presents a novel method that improves the performance of optimal motion planning techniques. Both the process of finding an initial plan and plan optimization are performed faster in comparison to existing methods.

Global collision-free planning for robot manipulators was computationally infeasible prior to the advent of sampling based planners. The critical advances made by [1–3] made it possible to find feasible paths in high dimensions. Recently, an optimal sampling based planner, RRT* [4], was introduced by Karaman. It takes advantage of the rapid expansion of RRT [2] while introducing path cost and optimality. RRT* iteratively refines its path and converges to an optimal one given coverage of the configuration space. We propose two sampling heuristics that improve RRT*, increase the rate of cost reduction and implement a bi-directional version of the algorithm to improve efficiency. The improvements can be qualitatively seen in Figure 1.

II. RELATED WORK

Sampling-based randomized algorithms are the current state-of-the-art to cope with high-dimensional motion planning problems. For multi-query planning, most planners are based on Probabilistic Roadmaps(PRMs) [1]. For single-query planning, Rapidly-exploring Random Tree (RRT) [2, 3] based planners have become the *de facto* standard in high-dimensional spaces. Our focus on changing environments leads us to concentrate on single-query planning.

Baris Akgun and Mike Stilman are with the Center for Robotics and Intelligent Machines and The School of Interactive Computing, Georgia Institute of Technology, Atlanta, GA. (barisakgun@gmail.com, mstilman@cc.gatech.edu)

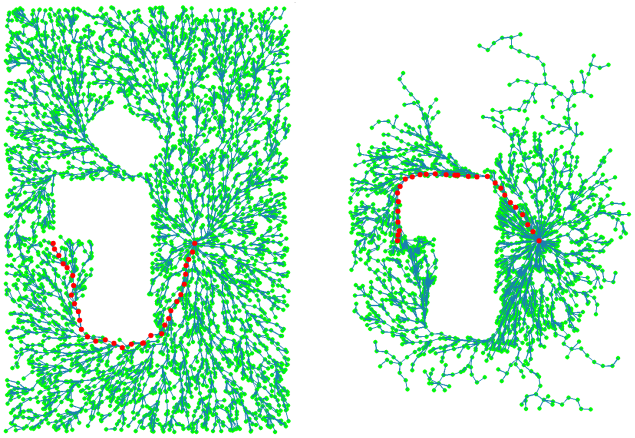


Fig. 1. Two search trees produced in the same amount of time. Left: Baseline RRT*. Right: A tree produced with proposed heuristics. The proposed tree finds the optimal solution in less time.

In general, RRT-like algorithms are only used to find a feasible path efficiently without the notion of cost. The skeleton of this algorithm is given in Algorithm 1. A tree in the configuration space of the robot is grown by sampling points and adding them to the tree if there exists a collision free path between the sampled point and the tree. The inherit nature (bias towards unexplored regions) of RRTs make them good at randomly exploring a space but not necessarily towards finding a path. Simple heuristics like sampling towards a goal (goal biasing) is often employed to speed up planning. The algorithm presented in [3] takes a double tree approach, by growing one tree from the start configuration and the other from the goal configuration, and includes a simple heuristic to guide the two trees towards each other. However, this does not take any notion of cost into account and is only concerned with finding a path.

Initial attempts to incorporate cost and bridge the gap between exploration and exploitation issue in RRT-like planners include the work in [5], which proposes to use a heuristic quality function to guide the search. Instead of selecting the nearest neighbor, they choose k-nearest neighbors and extend the neighbor with the best quality. An anytime approach to RRTs is developed in [6], where each subsequent call to RRT improves the solution by utilizing the information from previous calls. Samples which could not decrease the cost of the path are rejected. However, the solution does not necessarily converge to an optimal one. The algorithm presented in [7], biases the search towards the low cost

Algorithm 1: RRT

1: $V \leftarrow q_{init}$
2: $E \leftarrow \emptyset$
3: **while** $i < N$ **do**
4: $\quad q_s \leftarrow SAMPLE(i)$
5: $\quad EXTEND(V, E, q_s)$
6: **end while**

regions of a cost-map by rejecting high cost states and moving towards lower cost states via slopes of the cost function. In [8], exploration is done in the configuration space and exploitation (via potential fields) is done in the workspace of the robot but optimality is not considered. *Probabilistically optimal* RRTs, which the authors call RRT*, are recently introduced in [4]. Probabilistic optimality is analogous to probabilistic completeness. Under the assumptions given in [4], the solution converges to the optimal path as the number of samples approaches infinity. This approach includes the notion of cost within the tree. The details of RRT* will be given in the subsequent section. The work in [9] describes Rapidly-Exploring Roadmaps which is another recent optimal single-query planner. The algorithm maintains a graph which is updated by RRT type expansions for exploration and PRM type connections along the found paths for exploitation. This algorithm does not consider any sampling strategies.

III. APPROACH

First, we describe the RRT* algorithm. We then present the challenges involved in applying it to high-dimensional spaces and introduce our sampling heuristics.

A. Baseline RRT*

The baseline RRT algorithm is given in Algorithm 1. The *SAMPLE* routine selects a point in the robot's configuration space. *EXTEND* routine tries to grow the tree towards that selected point. RRT* algorithm has its own *EXTEND* routine, detailed in Algorithm 2. A concept introduced in RRT* is near neighbors, the neighbors within a certain radius of a node. RRT* connects newly added nodes to its neighbor node which yields the minimum cost. For the details on how to choose this radius, the reader is referred to [4]. This is the first loop in in Algorithm 2 and depicted in Figure 3(a). Another concept is rewiring the tree. If the path from the newly created node to a near neighbor node yields a lower cost for that near neighbor, then the parent of the near node is changed to the newly created node. This is the second loop in in Algorithm 2 and depicted in Figure 3(b).

In Algorithm 2 *EXTEND*, generates a new point that is ν away from the nearest node towards the direction of the sampled point as depicted in figure 2, but does not add this node to the tree. *NoCollision* routine checks for collision on the line between two points in the configuration space. The *NEAR* routine finds the *near* neighbors of a node in the tree. The *Cost* calculates the cumulative (integrated) cost from the initial node along the tree and *CostIm* calculates immediate

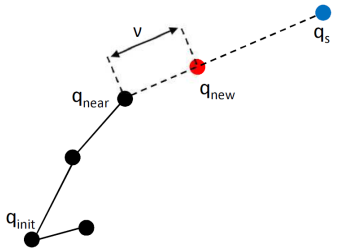


Fig. 2. The *EXTEND* routine. The nearest node to the sampled point on the tree is selected and a new node is added ν distance from the nearest node in the direction of the sampled point.

cost of going from one node to the other. The step size ν is decreased as the number of samples increase. as in [4].

B. Challenges in High Dimensions

In a time constraint scenario, where the path needs to be improved in the remaining planning time, it is desirable to have an algorithm that quickly decreases the cost of its plan.

For high dimensional spaces, the use of RRT* is limited. Random sampling is not able to refine the paths and/or find the optimal one in a reasonable amount of time. RRTs, and consequently RRT*, have an exploration heavy nature. This also adversely affects the rate of cost reduction in low-dimensional spaces.

The reason behind this behavior can be explained by utilizing the analysis in [10]. In a RRT, a path is defined by the list of nodes that connect the start node to the goal node. The nodes on the path are called the *path nodes*. Let's define two types of nodes as *inner nodes* and *frontier nodes*. Inner nodes are the ones that are surrounded by other nodes and frontier nodes are the ones that are neighboring the unexplored parts of the configuration space. Inner nodes have small Voronoi regions and frontier nodes have large Voronoi regions. Voronoi region of a node is directly related with its probability of being chosen for expansion. The issue becomes more severe in high-dimensional spaces where the frontier nodes are expected to have larger Voronoi regions.

As iterations go by, path nodes tend to become inner nodes and their probability of being selected for expansion decreases and becomes much lower than that probability of the frontier nodes, due to corresponding sizes of their Voronoi regions. This in turn hinders the path refinement.

C. Local Biasing

Local biasing is a sampling heuristic that facilitates sampling around the path nodes to accentuate the effect of RRT*'s cost reduction. The local sampling algorithm is given in Algorithm 3 and shown in figure 4. This heuristic attempts to circumvent the decreasing probability of path nodes being selected for expansion by explicitly sampling near them and helps refining the current path towards a locally optimal one. The aim is at least to reach a locally optimal solution iteratively within a reasonable amount of time, while preserving global optimality. In a time constraint scenario, decreasing the cost is more desirable than hunting for the optimal path

Algorithm 2: EXTEND_RRTS(V,E,q)

$q_{nearest} \leftarrow NEAREST(V,q)$
$q_{new} \leftarrow EXTEND(q_{nearest},q,v)$
if $NoCollision(q_{nearest},q_{new})$ **then**
  $V \leftarrow V \cup q_{new}$
  $q_{min} \leftarrow q_{nearest}$
  $Q_{near} \leftarrow NEAR(V,q_{new})$
  for all $q_{near} \in Q_{near}$ **do**
    if $NoCollision(q_{near},q_{new})$ **then**
      if $Cost(q_{near})+CostIm(q_{near},q_{new}) < Cost(q_{new})$ **then**
        $q_{min} \leftarrow q_{near}$
      end if
    end if
  end for
  $E \leftarrow E \cup (q_{min},q_{new})$
  for all $q_{near} \in Q_{near} \backslash q_{min}$ **do**
    if $NoCollision(q_{near},q_{new})$ AND $Cost(q_{new}) + CostIm(q_{near},q_{new}) < Cost(q_{near})$ **then**
      $q_{par} \leftarrow Parent(q_{near})$
      $E \leftarrow E \backslash (q_{par},q_{near})$
      $E \leftarrow E \cup (q_{new},q_{near})$
    end if
  end for
end if

Algorithm 3: LOCAL_BIAS(path)

$q \leftarrow RANDNODE(path)$
$q_1 \leftarrow path(q-1)$
$q_2 \leftarrow path(q+1)$
$q_{tmp} \leftarrow (q_1+q_2)/2 - q$
$q_{rand} \leftarrow q + \frac{q_{tmp}}{\|q_{tmp}\|} RAND(r_{min},r_{max})$
return q_{rand}

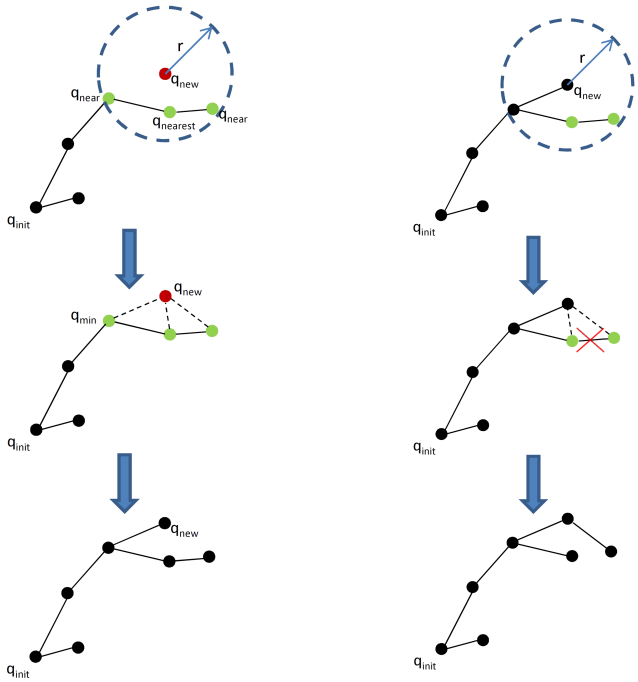
which is unlikely to be found. Local biasing heuristic is activated after an initial path is found.

In Algorithm 3, *RANDNODE* returns a random node within the given path. The rest of the operations correspond to selecting a random node, q_{rand}, along the direction of q_{tmp} which is the vector between q and the middle of its two neighbors, q_1 and q_2. If appropriate r_{min} and r_{max} are chosen, the procedure tends to *straighten out* the path. We keep the former an order of magnitude smaller and the latter on the same order as the step size to stay local to the path.

D. Node Rejection

Having a lower number of nodes in the tree and performing a lower number of *EXTEND* steps are beneficial towards efficiency. We can reject inclusion of nodes if they are guaranteed not to decrease the cost of the path. This rejection step is activated once a path is found. Node rejection is applied after *SAMPLE* routine selects a node and before *EXTEND* routine. A node is rejected if the following inequality holds;

$$\|q - q_{start}\| + \|q_{goal} - q\| > c_{best} \quad (1)$$



(a) First Loop (b) Second Loop

Fig. 3. Depiction of the two loops in the algorithm 2.

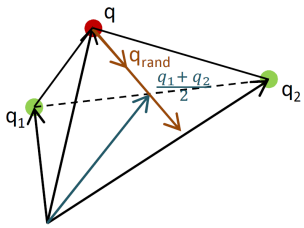


Fig. 4. Random node generation procedure for local sampling. Nodes are shown as vectors for visualization purposes.

Where q_{start} and q_{goal} correspond to the start and the goal configurations, q corresponds to the candidate node and c_{best} corresponds to the cost of the current lowest path cost.

A node is rejected if the current best cost is lower than the total cost of directly going from the initial node to this node and from this node to the goal node. Note that this approach requires the cost measure to obey the triangular inequality.

E. Bi-RRT*

The bi-directional version of the RRTs (Bi-RRT), introduced in [3], is more efficient than the single tree method. Bi-RRT is presented in Algorithm 4. In this algorithm, one of the trees is initialized with the start node and the other with the goal node. One of the trees takes a step towards a point (*EXTEND* part) and the other tree tries to connect to the newly added node, denoted by q_a, of the first tree. The presented algorithm assumes that the *EXTEND_RRTS* routine reports back whether it added a new node or not. In the *CONNECT* routine, the second tree takes multiple steps (in this case multiple *EXTEND_RRTS* calls with q_a as input) towards the last added node of the other tree until

Algorithm 4: Bi-RRT*

$V_a \leftarrow q_{init}, \quad V_b \leftarrow q_{goal}$
$E_a \leftarrow \emptyset, \quad E_b \leftarrow \emptyset$
while $i < N$ **do**
  $q_s \leftarrow SAMPLE(i)$
  if $EXTEND\_RRTS(V_a, E_a, q_s)$ **then**
    $CONNECT(V_b, E_b, q_a)$
  end if
  $SWAP(V_a, V_b), \ SWAP(E_a, E_b)$
end while

Algorithm 5: SAMPLE(k)

$p \leftarrow RAND(0.0, 1.0)$
if $\theta < p$ *AND* $\neg PathFound$ **then**
  return q_{goal}
else if $\beta < p$ *AND* $PathFound$ **then**
  return $LOCAL\_BIAS(path)$
else
  return $RANDQ$
end if

the node is reached or a collision occurs. The N defines the maximum number iterations. The reader is referred to [3] for other details of the Bi-RRT approach.

F. Resulting Algorithm

We combine the presented algorithms and heuristics to make a complete planner. We have a single tree and a double tree version. For brevity we will only present the former and mention the differences with the latter in the text.

The resulting *SAMPLE* routine is given in Algorithm 5. Before a path is found, this routine either returns a random configuration, or the goal configuration. The latter is done to bias the tree growth towards the goal so that the path is found faster. The parameter $\theta \in [0,1]$ sets the frequency of goal-biasing. After a path is found, the routine either returns a random configuration or a configuration obtained from local biasing. The parameter $\beta \in [0,1]$ sets the frequency of local-biasing. We do local biasing for explotiation and random sampling for exploration. In the bi-directional version, goal biasing is omitted. This is already facilitated by the Bi-RRT* algorithm itself. Note that the presented algorithm is terminated with the maximum number of iterations but in implementation, it is terminated with allocated time.

The overall algorithm is given in Algorithm 6. There a few details that are left out for brevity. Node rejection needs the lowest current cost and local-biasing needs the corresponding path. In the bi-directional version, when local biasing is done, both trees take a single step, i.e., *CONNECT* is not used.

Whenever a node gets in the vicinity of the goal (the other tree in the bi-directional case), that node (node pair in the bi-directional case) is stored. All the paths from these nodes can be traced, their costs can be calculated and the one with the lowest cost can be chosen. In practice, calculating the best path does not have significant effect on performance amidst near-neighbor calculations and collision checks.

It is important to note once again that the node rejection and local biasing heuristics are only activated after a path is found. The aim of this combined algorithm is to find a solution almost as fast as the baseline algorithm (RRT*) and iteratively improve the path quickly in the remaining computation time. To summarize, we use RRT* updates for optimality, local biasing to ensure that the nodes in the path get expanded, bidirectional implementation to find paths fast and node rejection to increase efficiency.

Algorithm 6: PLANNER

$V \leftarrow q_{init}$
$E \leftarrow \emptyset$
while $i < N$ **do**
  $q_s \leftarrow SAMPLE(i)$
  if $\neg NODEREJECT(q_s)$ *OR* $\neg PathFound$ **then**
    $EXTEND\_RRTS(V, E, q_s)$
  end if
end while

IV. ANALYSIS

A. Probabilistic Completeness

The probabilistic completeness and exponential convergence of RRTs have been proven in [11]. It has also been shown that RRT* shares the same properties, as shown in [4]. Since we keep random sampling, and do not reject any nodes until a solution is found, our algorithm does not lose this property.

B. Probabilistic Optimality

Given the same assumptions, our algorithm shares the probabilistic optimality of RRT* since we keep random sampling and sample rejection only rejects samples if they are guaranteed not to produce lower cost solutions if used in the path. It is important to note that local biasing is only a sampling heuristic which does not affect the types of paths produced.

C. Relation to Path Shortening and Smoothing

The paths returned by sampling-based planners are usually jagged and need shortening and/or smoothing. A typical example is the *corner-cut* approach presented in [12]. In this approach, every other node on the solution path is removed and the resulting path is checked for collisions. Variants, such as selecting two random nodes on the path and checking whether the robot could move from one to the other without collisions, are fairly common.

The effect of local biasing heuristic is similar to these post-processing steps in effect; both help to shorten and smooth the path but there are a few key differences. Local biasing is a sampling heuristic and does not delete any node from the tree, thus it can be active during tree construction. Corner-cut has an inherent lower limit on the cost, depending on the locations of the path nodes. It only explores their possible combinations whereas local biasing explores more space.

It is important to note that thw post-processing steps and the local biasing heuristic are not orthogonal and these steps can be applied on the path that our planner returns.

TABLE I

2D PERFORMANCE FOR A TIME BUDGET OF 1*sec*. RESULTS AVERAGED OVER 10 RUNS. PARENTHESES INDICATE STANDARD DEVIATION.

	c_1	c_{end}	Iterations
PL	8.07 (1.42)	5.84 (0.45)	5063 (166)
NR	8.07 (1.42)	5.16 (0.37)	9699 (1099)
LB	8.07 (1.42)	5.58 (0.68)	4732 (215)
CO	8.07 (1.42)	5.39 (0.71)	5593 (380)

V. SIMULATIONS AND RESULTS

We selected two test cases for simulation. First one is a simple 2D navigation problem and the second one is a relatively hard 7D manipulation problem. We use Euclidean distance in the configuration space of the robot as our cost.

A. 2D Navigation Case

This case was selected to easily visualize the trees and show what each improvement does. We analyze each heuristic separately and together against the baseline RRT* algorithm. The environment is shown in figure 6(a). There are three possible routes from start to goal. The one in the middle is the globally optimal one and poses a narrow passage. This environment is deliberately chosen to be very simple, to better understand the properties of the algorithm.

We set our parameters as $\nu = 0.2$ (step size), $\theta = 0.2$ (goal-bias) and $\beta = 0.2$ (local-bias). We set $r_{min} = 0.1\nu$ and $r_{max} = 2\nu$ for the local bias algorithm. We have four conditions; plain (PL), node rejection (NR), local bias (LB) and combined (CO). The combined condition is the combined planner. We gave each condition a time budget of 1*sec*. We pre-specified 10 random seeds to minimize the effect of chance. Note that the conditions are exactly the same until a path is found, so the path that they try to improve is the same for all conditions, given the same random seed.

The average results of these runs are presented in Table I. The columns are as follows; c_1 is the average initial path cost, c_{end} final path cost and the other is number of iterations. From Table I, it can be seen that the most significant improvement is node rejection. Node rejection increases the number of iterations by more than 90% on top of plain RRT* algorithm. Inclusion of local biasing decreases the final cost, but the positive effect is not as strong as in the node rejection case. This is partly because the environment is small and all the trees more or less explore the necessary part of the space to converge on a low cost.

The effect of local-biasing is more prominent during the initial iterations. This is explored by analyzing a typical run in detail. Figure 5 shows that local biasing has an impact on initial cost decrease. This is beneficial for our scenario in which remaining time is used for path improvements. Figure 6 shows the initial tree (same for all conditions) and the final trees for different conditions. Note that these final trees are not exhaustive and there are other cases. It can be seen that the local bias smoothes and shortens path and node rejection keeps the tree less spread. Note that the PL condition was not able to converge to the optimal path (see figure 6(c)), but it would have given more time. The time evolution of each method is present in the accompanied video.

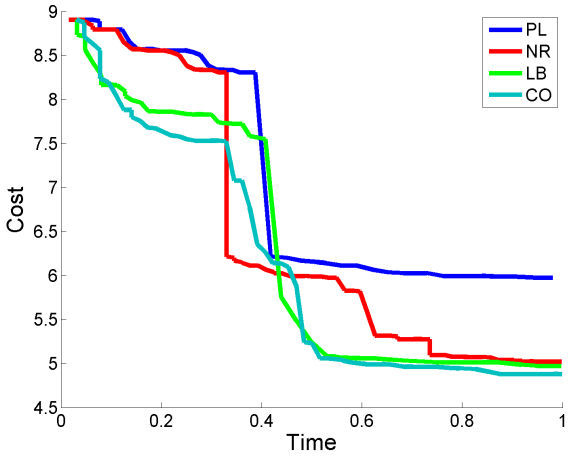
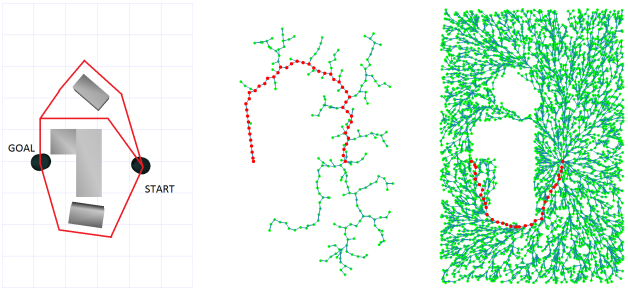
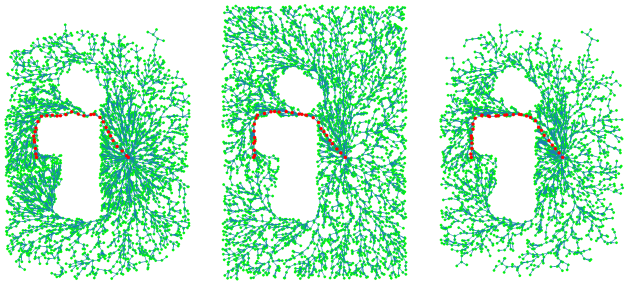


Fig. 5. Path cost versus time taken for a single run in the 2D case.



(a) The navigation environment (b) Tree when the first path is found (c) Final Tree for PL



(d) Final Tree for NR (e) Final Tree for LB (f) Final Tree for CO

Fig. 6. The shape of typical trees for different conditions at 1 *sec*.

B. 7D Manipulation Case

This case is relatively hard to solve, similar to the one in [8]. We compare the bi-directional version of our algorithm with the bi-directional version of RRT* since single tree versions are not sufficient for this problem. The environment is shown in figure 7. We used the same parameters as before except we set $\beta = 0.4$ to compensate for the larger space. We again take the 10 pre-specified random seed approach to test our algorithm. We gave each condition a time budget of 300*sec*. We denote the conditions with a *bi* prefix (e.g. biPL).

The average results of these runs are presented in Table II. It can be seen that node rejection does not contribute by

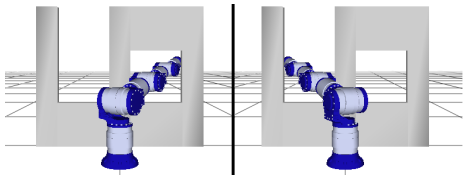


Fig. 7. The manipulation environment, start (left) and goal (right) configurations.

TABLE II

7D PERFORMANCE FOR A TIME BUDGET OF 300*sec*. RESULTS AVERAGED OVER 10 RUNS. PARENTHESES INDICATE STANDARD DEVIATION.

	c_1	c_{end}	Iteration
biPL	22.48 (1.86)	21.01 (1.39)	61766 (8869)
biNR	22.48 (1.86)	21.01 (1.39)	60043 (8942)
biLB	22.48 (1.86)	17.25 (1.66)	49991 (12359)
biCO	22.48 (1.86)	17.21 (1.66)	50209 (12384)

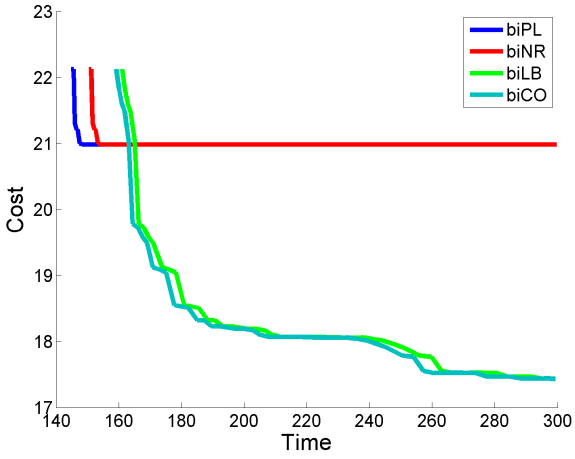


Fig. 8. Path cost versus time taken for a single run in the 7D case.

itself. We argue that the reason is the restrictive nature of the space. Node rejection helps when the search space is wide. On the other hand, local-biasing decreased the cost by 23.4% on the average whereas RRT* decreased it by 6.5%. Figure 8 shows a typical cost evolution versus time plot. It can be seen that initial cost decrease is prominent which is the desired result in our scenario. For some random seeds, biPL and biNR did not decrease the cost at all and for others biLB and biCO performed significantly better. The halting of cost decrease in biPL and biNR supports the hypotheses of path nodes eventually becoming inner nodes. Moreover, the resulting motion of the arm is smoother for the biCO, as can be seen in the accompanying video.

Plain RRT* algorithm finds the initial path fastest whereas our algorithm finds it later. We argue that streamlining the implementation of our algorithm would help close the gap. Note that biPL and biNR have almost the same performance which suggests that best path calculation (for our algorithm) does not significantly effect the performance.

The results show that local-biasing is a promising sampling heuristic to accelerate cost decrease in sampling-based planners. Node rejection, with its current implementation, is not very useful when the space is restrictive and may be omitted. However, its impact on computation time is negligible, thus it is recommended to be kept.

VI. CONCLUSION

RRT* algorithm is a big step towards incorporating the notion of cost and introducing probabilistic optimality into the domain of sampling-based planners. However, its direct implementation is not practical for high dimensional spaces. RRT* needs simple heuristics to make it more practical as RRTs needed simple heuristics to make them efficient.

Towards this end, we have developed a sampling based planner that adds two improvements over RRT* and implements a bi-directional version. This new planner can address the challenges of RRT* in high dimensional spaces and make it more efficient in general. The planner lets the user have control over exploration and exploitation by providing parameters. The heuristics are simple and can readily be incorporated into existing sampling based planners.

More informed local biasing, tighter node rejection criteria and deeper analysis of the effects of heuristic parameters (e.g. r_{min}&r_{max}) are the immediate extensions of this work. Adapting the biasiness parameter according to the cost decrease performance is also an interesting topic. There are also other potential heuristics that can further improve performance. Incorporating cost both into sampling and into tree growing is an attractive future research direction.

ACKNOWLEDGEMENTS

We would like to thank Neil Dantam, Jon Scholz, Ana Huaman and our reviewers for their helpful comments. This work is partially supported by NSF grant IIS-1017076.

REFERENCES

[1] L. Kavraki, P. Svestka, J. claude Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," in *IEEE ICRA*, 1996, pp. 566–580.
[2] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Iowa State University, Tech. Rep., 1998.
[3] J. Kuffner and S. M. Lavalle, "Rrt-connect: An efficient approach to single-query path planning," in *IEEE ICRA*, 2000, pp. 995–1001.
[4] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Rob. Res.*, vol. 30, pp. 846–894, June 2011.
[5] C. Urmson and R. Simmons, "Approaches for heuristically biasing rrt growth," in *IEEE/RSJ IROS*, October 2003.
[6] D. Ferguson and A. T. Stentz, "Anytime rrts," in *IEEE/RSJ IROS*, October 2006, pp. 5369 – 5375.
[7] L. Jaillet, J. Cortes, and T. Simeon, *Transition-based RRT for path planning in continuous cost spaces*. IEEE/RSJ IROS, Sep. 2008.
[8] M. Rickert, O. Brock, and A. Knoll, "Balancing exploration and exploitation in motion planning," in *IEEE ICRA*, 2008, pp. 2812–2817.
[9] R. Alterovitz, S. Patil, and A. Derbakova, "Rapidly-exploring roadmaps: Weighing exploration vs. refinement in optimal motion planning," in *IEEE ICRA*, 2011.
[10] A. Yershova, L. Jaillet, T. Simeon, and S. M. LaValle, "Dynamic-domain rrts: Efficient exploration by controlling the sampling domain," in *IEEE ICRA*, 2005, pp. 3867–3872.
[11] S. M. Lavalle and J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics*, 2001.
[12] P. C. Chen and Y. K. Hwang, "SANDROS: A dynamic search graph algorithm for motion planning," *IEEE Transactions on Robotics & Automation*, vol. 14, no. 3, pp. 390–403, 1998.