Rapidly-Exploring Random Tree Based Memory Efficient Motion Planning

Olzhas Adiyatov and Huseyin Atakan Varol
Department of Robotics and Mechatronics
Nazarbayev University
Astana, Kazakhstan
ahvarol@nu.edu.kz, oadiyatov@nu.edu

***Abstract* - This paper presents a modified version of the RRT* motion planning algorithm, which limits the memory required for storing the tree. We run the RRT* algorithm until the tree has grown to a predefined number of nodes and afterwards we remove a weak node whenever a high performance node is added. A simple two-dimensional navigation problem is used to show the operation of the algorithm. The algorithm was also applied to a high-dimensional redundant robot manipulation problem to show the efficacy. The results show that our algorithm outperforms RRT and comes close to RRT* with respect to the optimality of returned path, while needing much less number of nodes stored in the tree.**

Index Terms – Rapidly-Exploring Random Trees, Path Planning, Motion Planning, Redundant Manipulators.

I. Introduction

Robots are being used in many different domains and motion planning is essential in most if not all robotic applications. Robot motion planning problem encompasses finding a sequence of control actions to steer a robot from the initial state to the final state while satisfying the constraints imposed by the environment and the robot's own dynamics. The robot motion planning problem becomes a harder challenge in higher dimensional configuration spaces, which is prevalent in robotics. Initial approaches to the motion planning problem involved cell decomposition based methods [1], potential fields [2] and visibility graphs [3]. Due to the need for explicit representation of the obstacles in the configuration space, these planning algorithms are not suitable for high dimensional spaces with high number of obstacles. One frequently employed approach for solving the motion problem is to find a computationally inexpensive suboptimal plan quickly and improve this solution iteratively.

Recently, sampling-based planners have become the defacto standard for higher dimensional motion planning problems. Sampling-based methods offer significant computational savings over complete motion planning algorithms since explicit representation of the environmental constraints (obstacles) is not necessary. Instead sampling-based planners employ collision checking in order to verify the feasibility of a candidate trajectory, and connect a set of collision-free points in order to create a graph of feasible trajectories. This graph is then used to find a collision-free motion plan between the initial and the goal states.

Most popular sampling-based planners are Probabilistic Roadmaps (PRMs) [4, 5] and Rapidly-Exploring Random Trees (RRTs) [6, 7]. Both RRT and PRM are probabilistically complete, i.e. they converge to the optimal solution as the number of solutions approaches infinity. RRT and PRM randomly sample points from the state space and connect them to create a graph. Their primary difference is the way they create the graph connecting the sampled points. Multiple query-based PRM uses local planners to find snippets of collision-free trajectories in the state space connecting the randomly sampled points. Then a graph search is employed to determine the shortest path motion plan between the initial and goal states using the fragments of collision-free trajectories. Even though the multiple query methods such as PRM are valuable tools for motion planning in structured environments, they are not suitable for real-time implementations with changing environments and/or unknown environments.

Single query approaches such as RRT, which employs incremental sampling, are more advantageous for scenarios with dynamic environments and changing constraints. A pseudocode of RRT is given in Algorithm 1. Basic RRT algorithm grows a tree of feasible trajectories rooted at the initial state incrementally and returns a solution as one of the collision-free trajectories reaches the goal region. RRT does not use a metric to measure the optimality of the trajectory between the initial state and the other nodes. It tries to find a feasible solution as quickly as possible. Usually, the algorithm is not terminated after finding the first feasible solution. The remaining computation time is used to optimize the trajectory with respect to a cost function. Early work on the optimality of the RRT based motion planning algorithms is generally based on heuristics [8-11]. RRT* algorithm is introduced, which

Algorithm 1 $\tau = (V, E) \leftarrow \text{RRT}(z_{init})$

```
1   τ ← InitializeTree();
2   τ ← InsertNode(Ø, z_init, τ);
3   for i = 1 to i = N do
4       z_rand ← Sample;
5       z_nearest ← Nearest( , z_rand);
6       (z_new, u_new) ← Steer(z_nearest, z_rand);
7       if ObstacleFree(z_new, z_nearest), then
8        τ ← InsertNode(z_min, z_new, τ);
9       end if
10  end for
11  return τ
```

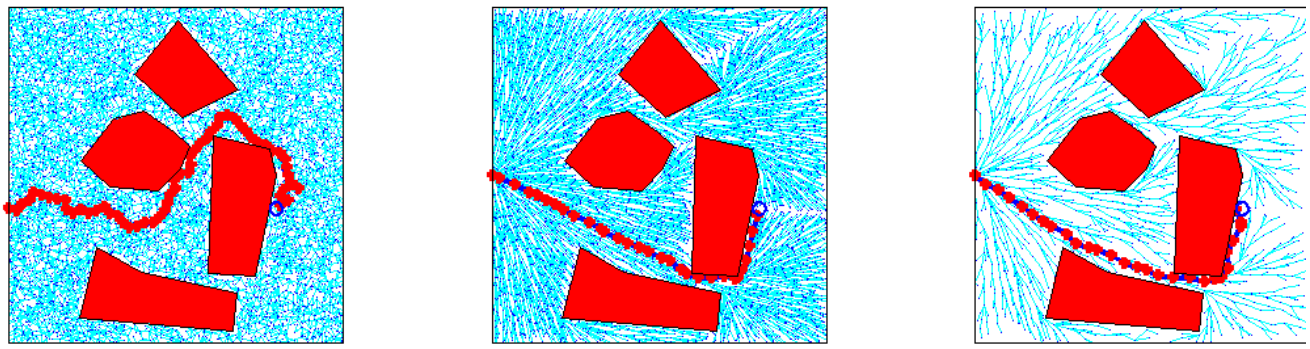

Fig. 1. Three search trees with the same number of iterations for a simple 2D mobile robot navigation task: RRT (left), RRT* (middle), RRT*FN (right). RRT finds a feasible but not optimal solution. Trees generated by RRT* and RRT*FN both converge to the optimal solution. RRT*FN tree resembles to a sparse version of the RRT* tree.

addresses the optimality problem of the RRT-based single-query planners with theoretical guarantees [12]. Under certain assumptions, this algorithm converges to the optimal solution as the number of samples reaches infinity. RRT* achieves this by incrementally rewiring the tree as lower cost trajectories become available with the addition of new nodes to the tree. RRT* does not guarantee upper bounds for the computation time of the optimal solution. Sampling heuristics are presented in [13] to combine the exploration-oriented nature of the RRT* with exploitation capabilities such that the convergence to the optimal solution is faster.

To the best of author's knowledge, there is no prior work, which limits the number of nodes in tree generated by the RRT* algorithm. The implementation of the RRT* algorithm in embedded systems with limited memory can become problematic, since the number of nodes in the tree grow indefinitely as the solution gets optimized. The number of nodes can be fixed to alleviate this problem, although this will mean that the convergence of the solution to the optimal solution is also hindered. In this work, we are presenting a new RRT based algorithm, which optimizes the path similar to RRT*. However, our algorithm limits the number of maximum nodes by employing a node removal procedure. We call this algorithm RRT* Fixed Nodes (RRT*FN). The nature of our approach can be qualitatively observed in Figure 1.

The rest of the paper is organized as follows: In Section II, the path planning problem is formulated and RRT* algorithm is described in detail. Section III presents our RRT* like algorithm, which limits the number of nodes in the trees. After describing the algorithm in detail, we present a series of simulation results for the algorithm in Section IV. Specifically, we compare the solution quality, computational requirements and convergence properties of our algorithm.

II. Background

A. Problem Formulation

In this part, we formulate the feasible and optimal path planning problems and provide definitions and notations, which will be relevant for the rest of the paper. Let Z^d be the state space of the problem, where $d \in \mathbb{N}, d \geq 2. Z_{obs} \subset Z^d$ is the obstacle region. The obstacle-free region is denoted as $Z_{free} = Z \backslash Z_{obs}$. The initial state z_{init} and the goal state z_{goal} are both elements of the Z_{free}.

Let the continuous function $\sigma: [0,1] \rightarrow \mathbb{R}^d$ be a path. A path is collision-free, if $\sigma(\alpha) \in Z_{free}$ for all $\alpha \in [0,1]$. A collision-free path is a feasible path, if $\sigma(0) = z_{init}$ and $\sigma(1) = z_{goal}$. The triplet $(z_{init}, z_{final}, Z_{free})$ contains the parameters, which define the path planning problem. A feasible path planning problem tries to find a feasible path with bounded variation such that $\sigma(0) = z_{init}$ and $\sigma(1) = z_{goal}$, if a feasible path exist. RRT solves this problem.

Let $c: \Psi \rightarrow R_{\geq 0}$ be the cost function, which assigns a strictly positive cost to all the collision free paths. The cost function is monotonic and bounded. The cost function is zero, if and only if $\sigma(\alpha) = \sigma(0), \forall\, \alpha \in [0,1]$ (the system does not move from the initial state). Given a problem with the triplet of parameters $(z_{init}, z_{final}, Z_{free})$ and a corresponding cost function $c: \Psi \rightarrow R_{\geq 0}$, optimal path planning problem seeks to find a feasible path σ_{best} between the initial and goal states such that $c(\sigma_{best}) = \min\{ c(\sigma): c\ is\ feasible\}$. RRT* and our algorithm solve this problem.

In the context of this paper, all path planning algorithms return a tree $\tau = (V, E)$, which is a specialized graph consisting of vertices $V \subset Z_{free}$ and edges $E \in V \times V$. The solution path is computed from the tree by finding the shortest path on this tree.

B. Probabilistically Optimal RRT (RRT*)

Algorithm 1 describes the operation of the RRT algorithm. It starts with a tree rooted at the initial state z_{init}. At each iteration, a random sample z_{rand} is drawn for the free region Z_{free}. "Nearest" routine finds the nearest node $z_{nearest}$ in the tree to the new randomly drawn sample. Then, the "Steer" function, finds the single step reachable state z_{new} from $z_{nearest}$, which is closest to the z_{rand} and satisfies the internal motion constraints. "Steer" function also outputs the necessary control action u_{new} to reach the z_{new} from the $z_{nearest}$. If the path from the $z_{nearest}$ to z_{new} is collision free, z_{new} is added to the tree τ as a child of $z_{nearest}$.

Algorithm 2 $\tau = (V, E) \leftarrow RRT^*(z_{init})$

```
τ ← InitializeTree();
τ ← InsertNode(Ø, z_init, τ);
for i = 1 to i = N do
    z_rand ← Sample(i);
    z_nearest ← Nearest(τ, z_rand);
    (z_new, u_new) ← Steer(z_nearest, z_rand);
    if ObstacleFree(z_new) then
        Z_near ← Near(τ, z_new, r);
        z_min ← ChooseParent(Z_near, z_nearest, z_new);
        τ ← InsertNode(z_min, z_new, τ);
        τ ← ReWire(τ, Z_near, z_min, z_new);
    end if
end for
return τ
```

Algorithm 3 $z_{min} \leftarrow \text{ChooseParent}(Z_{near}, z_{nearest}, z_{new})$

```
z_min ← z_nearest;
c_min ← Cost(z_nearest) + c(z_new);
for z_near ∈ Z_near do
    (x', u', T') ← Steer(z_nearest, z_new);
    if ObstacleFree(x') and x'(T') = z_new then
     c' = Cost(z_nearest) + c(x_new);
     if c' < Cost(z_new) and c' < c_min then
          z_min ← z_near;
          c_min ← c';
     end if
    end if
end for
return z_min
```

Algorithm 4 $\tau \leftarrow \text{Rewire}(\tau, Z_{near}, z_{min}, z_{new})$

```
for z_near ∈ Z_near \ z_min do
    (x', u', T') ← Steer(z_nearest, z_new);
    if ObstacleFree(x') and x'(T') = z_near and
    Cost(z_new) + c(x') < Cost(z_near) then
        τ ← ReConnect(z_new, z_near, τ);
    end if
end for
return τ
```

RRT* extends RRT with three new concepts. The first one is the local neighborhood of a newly added node. The second one is the cost function, which provides the total cost from the initial state to a node in the tree. The third one is the rewiring procedure, which rewires the local neighborhood of the newly added node such that the cost to the initial state decreased. Now we will describe the operation of the RRT* algorithm in detail.

RRT* algorithm shown in Algorithm 2 is similar to the RRT. It tries to connect the new sample to the nearest node at first. However instead of setting the nearest node as the parent of z_{new}, it finds the nodes Z_{near} in the local neighborhood of z_{new}. Z_{near} can be found by selecting the nodes inside a region around z_{new} with a radius r. Then *ChooseParent* routine selects the best parent for z_{new}. The selection of the parent is detailed in Algorithm 3. Specifically, z_{new} is connected to the parent, which minimizes the total cost from z_{init} to the z_{new}. This is one of the two instances, where RRT* optimizes the complete trajectory. Afterwards z_{new} is inserted to the tree as a new node. The local neighborhood of z_{new} is optimized again with the *Rewire* routine (Algorithm 4). In this case, the total cumulative cost of the path from z_{init} to all nodes in Z_{near} is calculated assuming that z_{new} is the parent node of the nodes in Z_{near}. If the assignment of z_{new} as the parent decreases the total cost from z_{init} to $z_{near} \in Z_{near}$, then the *Reconnect* removes the edge between z_{near} and it's parent node and creates an edge between z_{near} and z_{new}, the new parent node. This way, RRT* also utilizes the newly inserted node to shorten the total path to the nodes in the vicinity.

III. RRT* Fixed Nodes

RRT* is proven to be probabilistically complete in [12]. The probability of converging to the optimal path goes to one as the number of nodes approaches infinity. Additional memory is needed to store each added node to the tree. Even though the benefits of motion planning for robotic and mechatronic systems is obvious, their embedded systems usually have low amount of memory hindering the prospects of using RRT* and similar motion planners. In order to implement RRT* on these types of systems, one can terminate the RRT* algorithm once a predefined number of nodes in the tree is reached. Nonetheless, in this case there will be no guarantees on the quality of the solution provided. In this work, we are presenting RRT*FN, which is an extension of the RRT* algorithm. It continues optimization of the tree after a fixed number of nodes is reached. Specifically, RRT*FN uses the skeleton of the RRT* and enriches it with a node removal procedure. Initially, the tree is grown identical to RRT* until a maximum number of nodes M is reached. If a

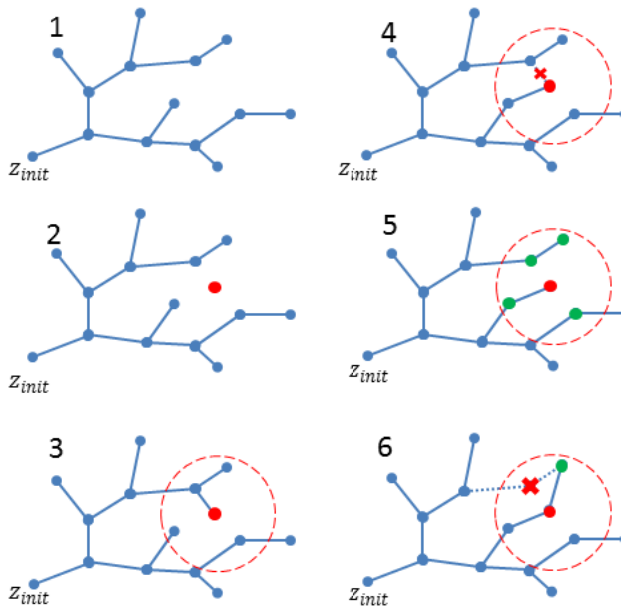

Fig. 2. Visualization of the RRT*FN algorithm for a simple 2D case depiction node insertion, rewiring and removal. Steering is omitted to make the illustration more understandable.

Algorithm 5 $\tau = (V, E) \leftarrow$ RRT*FN(z_{init}, M)

$\tau \leftarrow$ **InitializeTree**();
$\tau \leftarrow$ **InsertNode**(Ø, z_{init}, τ);
for i = 1 to i = N **do**
  if M < **NodesAdded**(τ) **then**
    $\tau_{old} \leftarrow \tau$;
  end if
  $z_{rand} \leftarrow$ **Sample**(i);
  $z_{nearest} \leftarrow$ **Nearest**(τ, z_{rand});
  ($x_{new.}$ u_{new}, T_{new}) $\leftarrow$ **Steer**($z_{nearest}$, z_{rand});
  if ObstacleFree(x_{new}) **then**
    $Z_{near} \leftarrow$ **Neighbors**(τ, z_{new}, |V|);
    $z_{min} \leftarrow$ **ChooseParent**(Z_{near}, $z_{nearest.}$ $z_{new.}$ x_{new});
    $\tau \leftarrow$ **InsertNode**($z_{min.}$ $z_{new.}$ τ);
    $\tau \leftarrow$ **ReWire**(τ, Z_{near}, z_{min}, z_{new});
    $\tau \leftarrow$ **ForcedRemoval**(τ, z_{goal});
  end if
  if NoRemovalPerformed() **then**
    $\tau \leftarrow$ **RestoreTree**();
  end if
end for
return τ

Algorithm 6 $z_{min} \leftarrow$ ChooseParent(Z_{near}, $z_{nearest}$ z_{new} x_{new})

$z_{min} \leftarrow z_{nearest}$;
$c_{min} \leftarrow$ **Cost**($z_{nearest}$) + c(x_{new});
for $z_{near} \in Z_{near}$ **do**
  (x', u', T') $\leftarrow$ **Steer**($z_{nearest}$, z_{new});
  if ObstacleFree(x') and x'(T') = z_{new} **then**
    c' = **Cost**($z_{nearest}$) + c(x_{new});
    if c' < **Cost**(z_{new}) and c' < c_{min} **then**
      $z_{min} \leftarrow z_{near}$;
      $c_{min} \leftarrow$ c';
    end if
  end if
end for
return z_{min}

Algorithm 7 $\tau \leftarrow$ Rewire(τ , Z_{near}, z_{min} z_{new})

for $z_{near} \in Z_{near} \setminus z_{min}$ **do**
  (x', u', T') $\leftarrow$ **Steer**($z_{nearest}$, z_{new});
  if ObstacleFree(x') and x'(T') = z_{near} **and**
    Cost(z_{new}) + c(x') < Cost(z_{near}) **then**
    if onlyChild(**Parent**(z_{near})) **and**
      M < **NodesAdded**(τ) **then**
      RemoveNode(**Parent**(z_{near}));
    end if
    $\tau \leftarrow$ **ReConnect**($z_{new.}$ $z_{near.}$ τ);
  end if
end for
return τ

feasible path to the goal state is not reached at this point, the algorithm must be restarted. Then RRT*FN removes a node whenever a new node is added.

Figure 2 provides a depiction of new node addition in RRT*FN for a simple 2D case. The details of RRT*FN are given in Algorithm 5-7. A new node is added, if and only if the cost to the present state from the initial state has decreased and there exists at least one node in the tree with one or no child. To add a new node, one node with one or no child is deleted from the tree. If there are more than one node present without children, one of them is selected randomly. The reason for selecting this node removal procedure is the following. If a node does not have children, it also means that it is not on a path reaching the goal state.

The first attempt to do the node removal procedure is during *Rewire.* If a node in Z_{near} is the only child of another node in Z_{near} and the cumulative path cost to this child node is lower through z_{new}, then the child node is reconnected as a child to z_{new} and the parent is deleted. We observed that there are cases, where a node cannot be added since there is no nodes with only child to remove in Z_{near} . Presumably, this would decrease the convergence rate of RRT*FN. In order to alleviate this situation, a global node removal procedure (*ForcedRemoval*) procedure is employed. *ForcedRemoval* searches the whole tree for nodes without children and removes one randomly. In case, no nodes can be found for removal in *Rewire* and *ForcedRemoval*, z_{new} is removed from the tree.

IV. Experiments and Results

Two benchmark problems were used to evaluate the efficacy of the RRT*FN algorithm compared to RRT and RRT*. These were a 2D mobile robot navigation problem and a 6D planar redundant robot manipulator motion planning problem. 2D example was selected to qualitatively compare RRT* and RRT*FN algorithms. The search space is much larger for the 6D case and it constitutes a much harder path planning problem. Cumulative Euclidean distance between the nodes on the path is used as the cost metric in both cases. For RRT*FN, the maximum number of nodes in the tree was set to 1750 and 10000 for the 2D mobile navigation and 6D redundant robot problems, respectively. The algorithms (RRT, RRT* and RRT*FN) are run ten times for ten different maps using pre-specified random seeds for both problems.

The path planned for one run of the 6D redundant manipulator problem using RRT*FN is shown in Fig. 3. As it can be observed from the figure, RRT*FN is able to generate a trajectory through a narrow passage between obstacles and reach the goal point. Figure 4 shows the average minimum path cost versus the iteration of the algorithms for the two test cases. Due to the lack of node removal procedures, RRT* algorithm finds the initial feasible path and converges to the optimal path fastest in most of the trials. RRT is the worst performing algorithm with respect to optimality of the path and path cost convergence properties. RRT*FN also converges towards the optimal path in both problems; however, the convergence is slower. Even though it appears that RRT* is the best performing algorithm, we should note that RRT*FN eventually converges to a near optimal solution as RRT* and does this with much less memory (Memory for RRT* increases linearly as iterations increase. Memory used for RRT*FN is fixed.)

The characteristics of the RRT* and RRT*FN is shown in Fig. 4. Until the maximum number of nodes for RRT*FN is reached, RRT* and RRT*FN behave similar to each other.

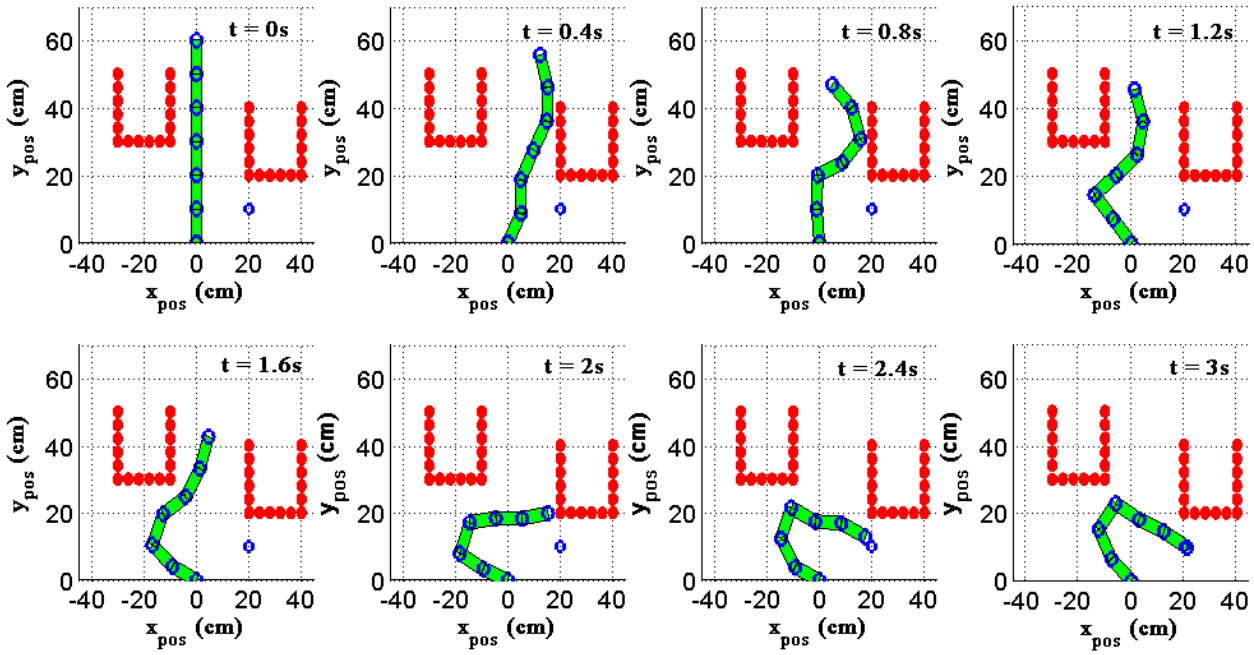

Fig. 3. Motion plan generated by the RRT*FN algorithm for the 6D planar redundant manipulator in an environment with multiple obstacles.

After this point, RRT*FN continues optimization of the path by adding better nodes and removing the ones with no children or one child. RRT* adds additional nodes to improve the path without removing any. Therefore, the tree starts looking denser in RRT* compared to RRT*FN, which is an indication of the additional memory requirements. One of the advantages of RRT* is the theoretical guarantee on the convergence to the optimal solution. Even though simulation experiments strongly suggest that this is the case for RRT*FN as well, probabilistic optimality of RRT*FN algorithms remains yet to be proven.

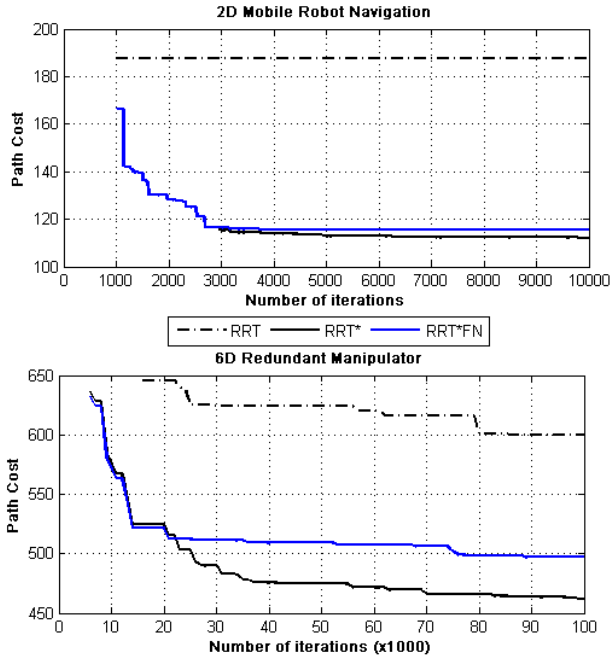

Fig. 4. Minimum path cost versus number of iterations for RRT, RRT* and RRT*FN for 2D mobile robot navigation (upper) and 6D redundant manipulator motion planning (lower) problems.

IV. Conclusion

RRT* offers a simple, practical and elegant solution to the optimal path planning problem. The implementation of this algorithm on systems with limited memory resources remains problematic since the memory required to store the tree grows as the number of the nodes increases. In this work, we presented a new single-query sampling-based planner similar to RRT*, which fixes the maximum number of nodes in the tree and yet continues to optimize the path. Simulation experiments showed that optimal path can be found with our algorithm, even though the rate of convergence to the optimal solution is slower than baseline RRT*.

Future work includes use of different node removal heuristics to improve convergence rate of our algorithm and use of RRT*FN in different robotic motion planning problems. Another area of future research is addition of dynamic rewiring methods to RRT*FN such that it can be more suitable for environments with moving or unknown obstacles.

References

[1] T. Lozano-Perez, "Spatial Planning - a Configuration Space Approach," *IEEE Transactions on Computers,* vol. 32, pp. 108-120, 1983.

[2] O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *International Journal of Robotics Research,* vol. 5, pp. 90-98, Spr 1986.

[3] T. Lozano-Perez and M. A. Wesley, "Algorithm for Planning Collision-Free Paths among Polyhedral Obstacles," *Communications of the ACM,* vol. 22, pp. 560-570, 1979.

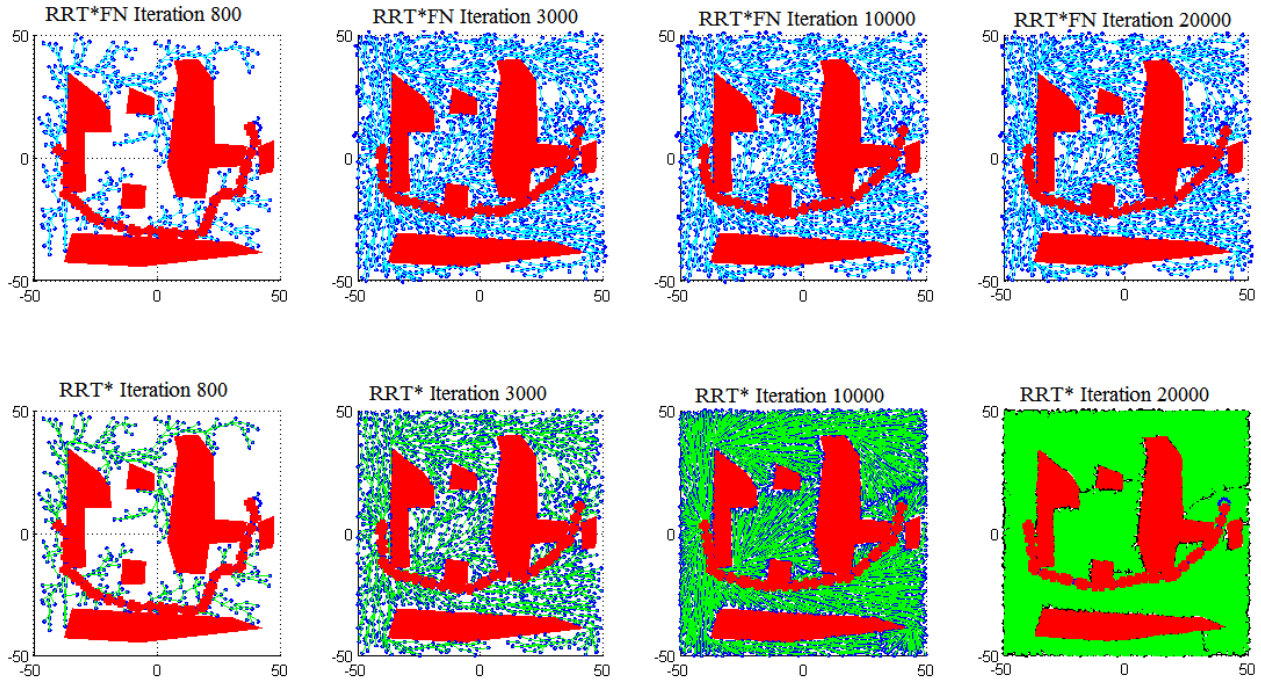

Fig. 5. Evolution of the trees and the minimum cost paths at different iterations using RRT* (upper row) and RRT*FN (bottom row).

[4] L. E. Kavraki, M. N. Kolountzakis, and J. C. Latombe, "Analysis of probabilistic roadmaps for path planning," *IEEE Transactions on Robotics and Automation,* vol. 14, pp. 166-171, Feb 1998.

[5] L. E. Kavraki, P. Svestka, J. C. Latombe*, et al.*, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation,* vol. 12, pp. 566-580, Aug 1996.

[6] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research,* vol. 20, pp. 378-400, May 2001.

[7] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring Random Trees: Progress and prospects," *Algorithmic and Computational Robotics: New Directions,* pp. 293-308, 2001.

[8] D. Ferguson and A. Stentz, "Anytime RRTs," *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vols 1-12,* pp. 5369-5375, 2006.

[9] L. Jaillet, J. Cortes, and T. Simeon, "Sampling-Based Path Planning on Configuration-Space Costmaps," *IEEE Transactions on Robotics,* vol. 26, pp. 635-646, Aug 2010.

[10] Y. Kuwata, J. Teo, G. Fiore*, et al.*, "Real-Time Motion Planning With Applications to Autonomous Urban Driving," *IEEE Transactions on Control Systems Technology,* vol. 17, pp. 1105-1118, Sep 2009.

[11] M. Rickert, O. Brock, and A. Knoll, "Balancing exploration and exploitation in motion planning," in *2008 IEEE International Conference on Robotics and Automation, Vols 1-9*, 2008, pp. 2812-2817.

[12] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *International Journal of Robotics Research,* vol. 30, pp. 846-894, Jun 2011.

[13] B. Akgun and M. Stilman, "Sampling Heuristics for Optimal Motion Planning in High Dimensions," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011.