National Conference with International Participation
ENGINEERING MECHANICS 2006
Svratka, Czech Republic, May 15 – 18, 2006
paper no. **177**

IMPROVING RAPIDLY EXPLORING RANDOM TREES METHOD USING TWO TREES

J. Krejsa[*], S. Věchet[+]

Summary: *The paper is focused on increasing the speed of rapidly exploring random trees method, used for path planning tasks. Original method uses single tree running from the initial node. Modified version of the method uses two trees; second one running from the goal node and nodes of both trees are occasionally connected to form the obstacle free path from initial to goal node. Description of the modification together with detailed comparison of both original and modified versions is included in the paper. Modified version significantly reduces the search time; however there are certain drawbacks mainly regarding the memory requirements.*

1. Introduction

Rapidly exploring random trees (RRT) method is a suitable method for solving the path planning problem. The method was successfully applied on mobile robot (both walking and wheeled) path planning task including the extended set of restrictions corresponding to the failure state of the robot. The method proved to be reliable and fast, however, for the complex maps it is desirable to further reduce the search time. Further text explains in detail the modification of the method, made in order to speed up the search process, including the comparison with the original method and discussion over its advantages and drawbacks.

2. Modified algorithm

The original method, introduced by La Valle, 1998 uses randomized data structure sequentially expanded by creating new nodes of a tree structure in the direction of randomly selected points. RRT starts with initial state x_{init} searching for the goal state x_{goal} and the vertices of RRT must be constructed so all the nodes are in obstacle free space. Sequential tree expansion first generates the random state x_{rand} and finds the closest node $x_{closest}$ in existing structure and new node is generated in Δx distance from $x_{closest}$ in the direction towards the x_{rand}. This step is denoted as "tree expansion" and shown in Fig. 1. When new node meets all restrictions it is added to RRT structure and the process continues until the

[*] Ing. Jiří Krejsa, PhD. Institute of Thermomechanics – Brno branch, Czech Academy of Sciences, Technická 2, 616 69, Brno, Czech Republic, tel: +420 541142885, email: jkrejsa@umt.fme.vutbr.cz

[+] Ing. Stanislav Věchet, PhD. Institute of Automation and Computer Science, Brno University of Technology, Technická 2, 616 69, Brno, Czech Republic

node sufficiently close to x_{goal} is found forming the obstacle free path. The original algorithm is described in detail in Krejsa & Věchet, 2005, including the application on mobile robot path planning and use of further node generation restrictions (corresponding to failure states of the robot). When used for mobile robot path planning, the node description simply consists of x and y coordinates of the robot.

```
1.  x_rand = random state
2.  x_closest = GetClosestNode( x_rand )
3.  x_new = GenerateNewNode( x_closest , x_rand )
4.  x_new = ApplyRestrictions( x_closest , x_new )
5.  if ( x_new is OK)
6.          RRT.AddNewNode( x_closest , x_new )
7.  else
8.          RRT.Trapped
9.  end if
```

Fig. 1. Tree expansion pseudocode

Modified algorithm uses the same expansion procedure, but it incorporate two trees – first starting from the node corresponding to initial state x_{init}, second starting from the node corresponding to the goal state x_{goal}. Trees expansion is sequential, supplemented with the routine connecting the nodes of both trees in order to find the obstacle free connection between the trees (see Fig. 2).

```
1.  repeat
2.      for i = 1 to CONNECT_INTERVAL
3.          RRT_init expansion
4.          RRT_goal expansion
5.      end for
6.      PathFound = Connect( RRT_init , RRT_goal )
7.  until PathFound
```

Fig. 2. Modified algorithm pseudocode

The *Connect* procedure sequentially tries to connect the nodes of RRT_{init} with nodes of RRT_{goal} while keeping the connecting line obstacle free and meeting all other defined restrictions. The procedure is called every *n*-th run of the expansion procedures, where *n* corresponds to CONNECT_INTERVAL variable. The simplest version of *Connect* runs all the combinations of nodes in both trees and checks whether the possible connection is obstacle free. To avoid repeated checking of previously tried node combinations the already tested combinations can be stored in lookup table.

3. Algorithm comparison

Modified algorithm was compared with the original on the mobile robot path planning tasks, using two types of maps – simple and complex, as it's behavior might depend on the complexity of the map. Figure 3. shows the maps (both of size 500 x 500) with described initial and goal positions. No further restrictions apart from obstacle free path from initial to goal position were defined for the task.

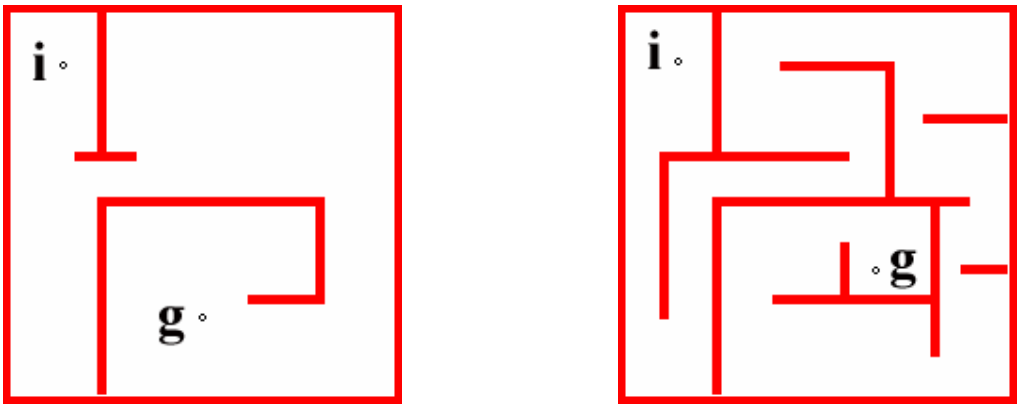

Figure 3. Sample maps with init and goal positions – simple and complex representatives

The comparison of the tree expansion for both original (top row) and modified version (bottom row) of the algorithm for simple map is shown in Fig. 4. Total number of nodes in RRT_{init} in the original version and the sum of RRT_{init} + RRT_{goal} nodes is kept the same to clearly view the difference in tree expansion. One can see that two trees modification faster covers the unsearched portions of the map.

The parameter on which the search speed depends the most is the length of the step Δx. In order to compare both versions of the algorithm the dependency graphs including the number of nodes in the tree (or both trees for the modified version) and number of nodes in the resulting path were generated for both types of the map. As tree generation is the random process the experiment was repeated twenty times for each case and mean values and standard deviations are included. Further the reduction of the number of nodes both in full trees and in the path is shown, related to the higher value. Generated graphs are shown in figures 5-8. Figures 5 and 6 show the results for the simple map, latter for the complex one. First figure shows the total number of nodes in the tree and nodes reduction, second the number of nodes in the path and again path nodes reduction.

As one can see in both cases there is a significant reduction of number of tree nodes (roughly corresponding to the search time – will be discussed in more detail in discussion section). For the simple map the reduction is oscillating in the range of 25 – 40 %, in the complex map case in the range of 30 - 50 %, with no clear relation to the step length as expected. Curves corresponding to the number of tree nodes versus step length show expected U shape for the complex map, simple map does not contain enough "hard to solve" portions for the longer steps to limit the search.

When looking to the number of nodes in the resulting paths, there is significant reduction in the simple map case (20-30 %). This is clearly caused by the shape of the map – there is vast open space in the top right part of the map which two trees version can easily cross in the

connect phase, while single tree has to use several steps to cross the section. For the complex map there is no significant change in number of path nodes, as expected.

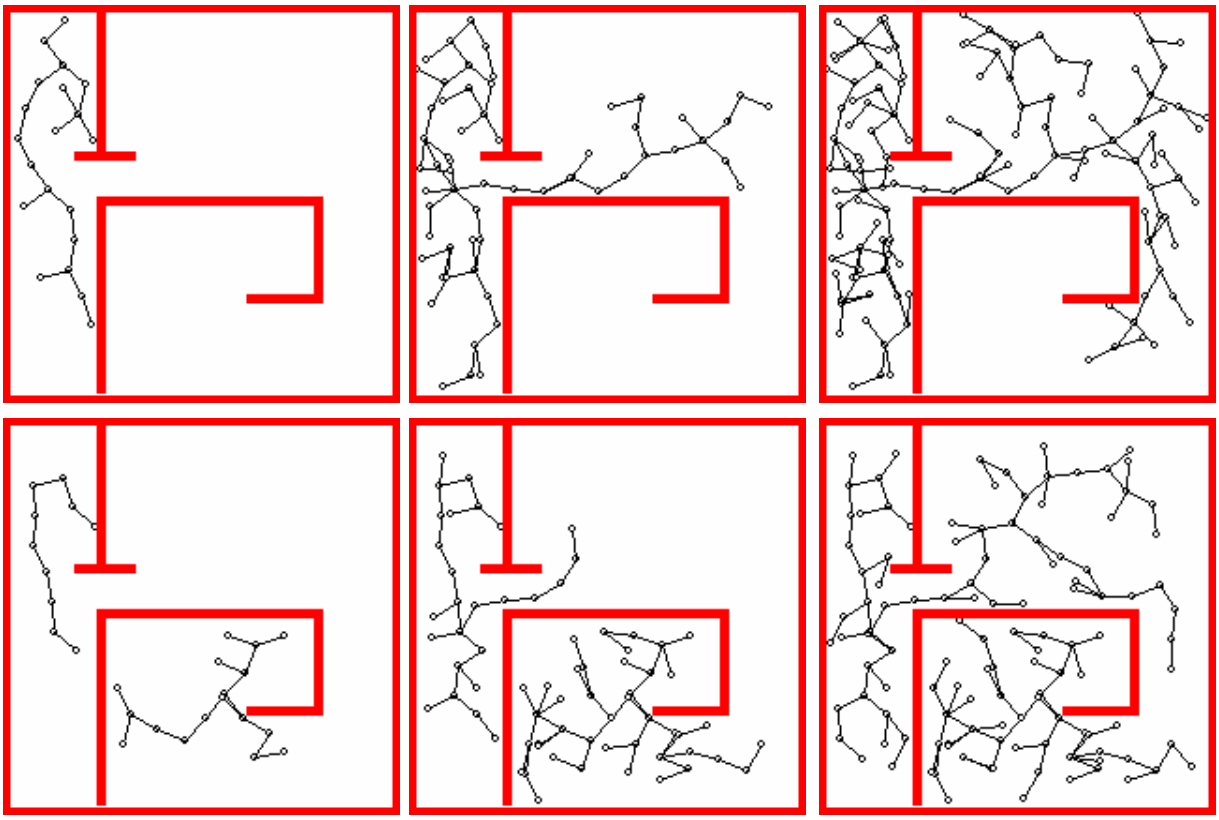

Figure 4. Tree development in simple map – single/double tree comparison. Respective number of nodes in the trees is 20, 60 and 130. Top row shows single tree expansion, bottom row shows two trees expansion. Step length $\Delta x = 80$.

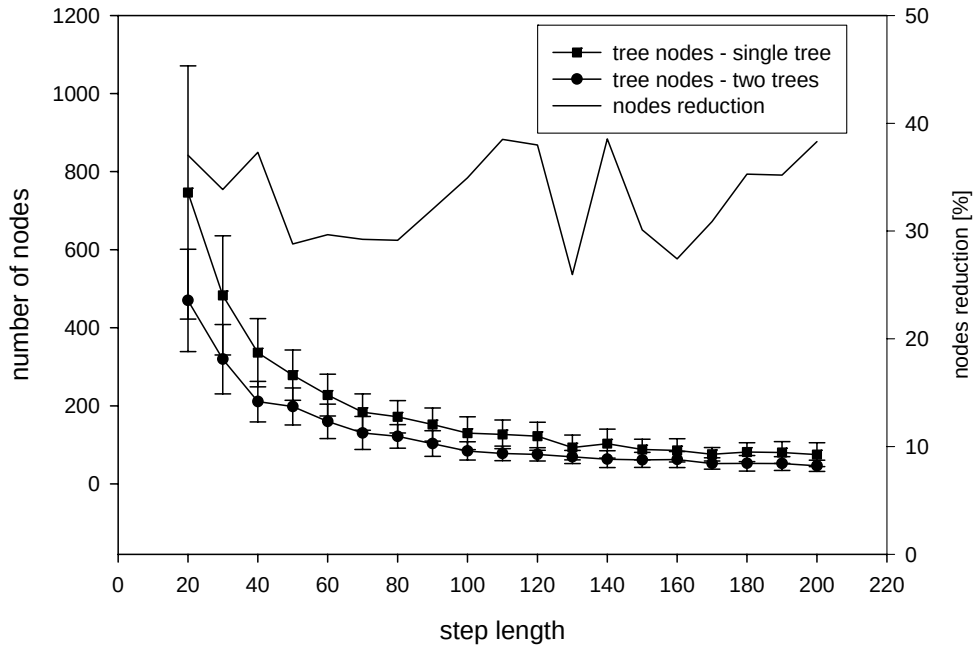

Figure 5. Total number of nodes in the tree depending on step length – simple map.

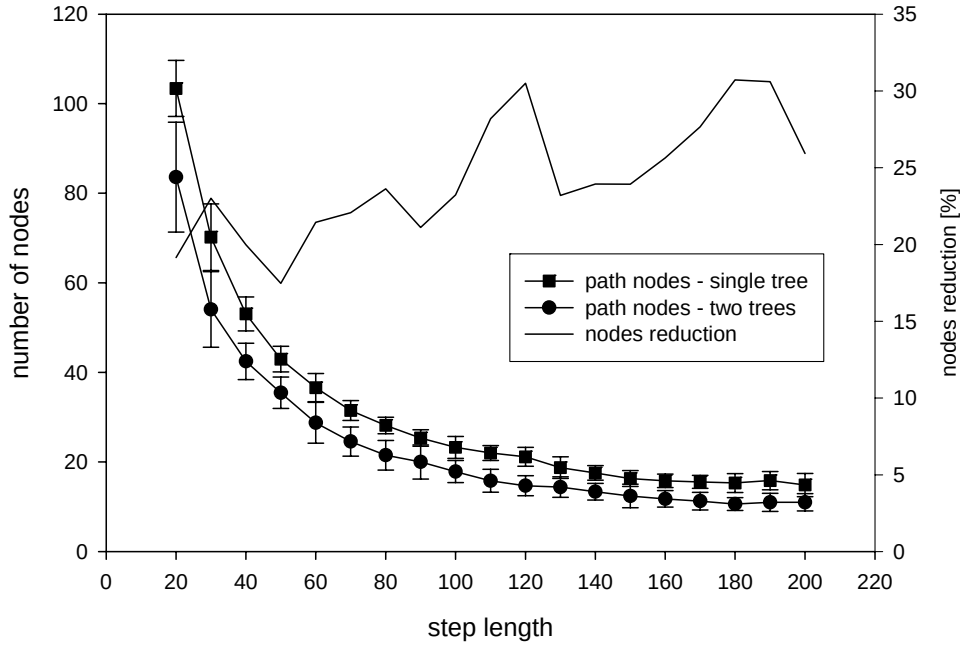

Figure 6. Number of nodes in the path depending on step length – simple map.

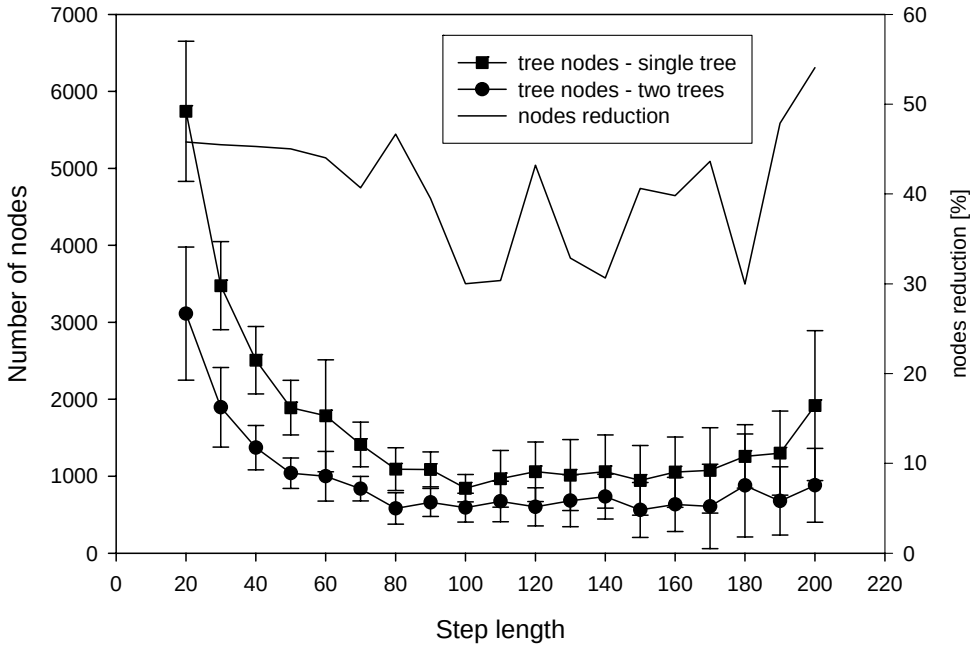

Figure 7. Total number of nodes in the tree depending on step length.

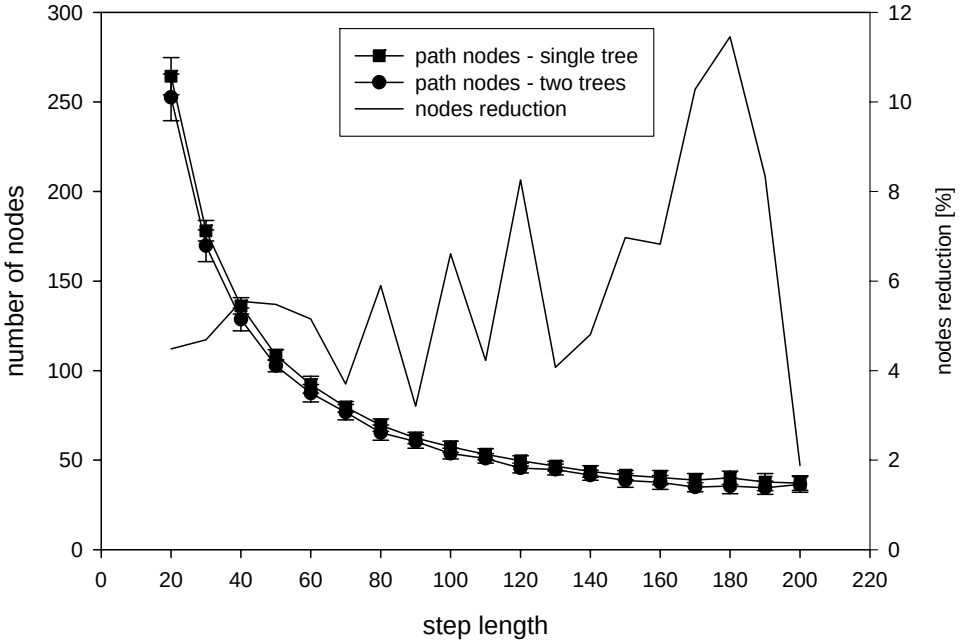

Figure 8. Number of nodes in the path depending on step length – complex map.

4. Discussion

Advantages

Significant reduction of number of nodes in resulting tree is clear in full range of step lengths used in experiments both for simple and complex maps. Depending on particular map complexity the reduction can be expected in range of 25 – 50 %, which represents significant improvement of search speed. The modification of the algorithm is simple and modified version is easy to implement.

Drawbacks

There are two major drawbacks of the modified version of the method, both related to the *connect* phase. First drawback is related to the memory usage. The number of nodes in both trees is related to the total time of the search when *connect* phase is fractional of total search time. This can be achieved by remembering what nodes were already tried to connect. In the single tree version such remembering is simply a matter of adding boolean variable to each node and setting it to false when the attempt of connecting the node with the goal is unsuccessful in order to avoid further tests of given node. However, for the two trees structure the combination of attempted nodes connections must be remembered, thus creating a memory demand exceeding the original demand of the tree structure (considering the common node definition, such as x and y coordinates in tested case). The "failed nodes combination" requires approximately $\frac{1}{4}n^2$ values to remember for total number of nodes in both trees equal to n. One can run the *connect* tests independently each time it is performed, the reduction of the search time is then smaller than *connect* requirements and overall performance of the algorithm is worse than single tree version. Therefore the increase of search speed does not come for free, it requires additional memory, as in many other cases.

Second drawback is again related to the *connect* phase. When further restrictions are applied into new node generation procedure (e.g. limited angle range), the search in the single tree case stops when the node is “sufficiently close” to the goal specifications (as the exact path is not guaranteed to exist). Therefore the exact match of nodes in two trees connecting operation is highly unlikely to appear and “close” nodes must be used instead. The definition of “close” node then depends on the particular application and for nonholonomic constraints the whole advantage of speeding up the process is likely to be neglected.

5. Conclusions

The modified version of RRT algorithm brings substantial reduction of number of nodes in the resulting tree structure thus reducing the search time, when extended memory requirements are permitted. Depending on particular application such search time reduction can bring substantial improvement of whole application, however, the limits of the method, mainly connected with the nonholonomic constraints must be taken into account prior to the application.

6. Acknowledgement

This work was supported by Czech Ministry of Education under project MSM 0021630518 "Simulation modelling of mechatronic systems".

7. References

LaValle S.M. (1998) Rapidly-exploring random trees: A new tool for path planning, *technical report*, Computer Science Dept., Iowa State University

Krejsa J., Věchet S. (2005) Rapidly Exploring Random Trees Used for Mobile Robots Path Planning. *Egineering Mechanics* 12 (4) pp. 231-238