The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

Cell-RRT: Decomposing the Environment for Better Plan

Julien Guitton, Jean-Loup Farges and Raja Chatila

Abstract— **In order to define an architecture for task and motion planning of a mobile robot, we propose the Cell-RRT path planner that combines the advantages of planning approaches by decomposition of the environment and the advantages of probabilistic approaches. Experiments of the method for various decomposition granularities and various adjustments of the planner settings show that using a bias towards the goal while choosing a random configuration reduces the paths length but can cause failures, that the choice of the criterion for analysing the environment is important, and that the method can profit from a reuse of already made computations in a part of the environment.**

I. Introduction

The field of mobile robotics is subject to a research effort over the past years. The increasing needs of autonomy for robotic systems imply an embedded mission planning system. Within this framework, algorithms have to be more and more efficient.

New reasoning architectures for mobile robotics have been developed, among which hybrid planning architectures that interleave search phases for actions allowing to fullfill the mission and search phases for paths to accomplish these actions [1]. This kind of architectures requires a low-time response from the path planner to the task planner requests. However, the computation time is strongly dependent on the complexity of the environment in which the robot is evolving.

Path planning algorithms commonly use a structuration of the environment or configuration space under the form of a graph or a grid [2], [3], [4]. Most of these methods can be grouped under four classes: reduction approaches, cell decomposition approaches, potential field approaches and probabilistic approaches. Trying to bring together the advantages of these four classes is a research track that aims at improving path planning algorithms.

In this paper, we propose to combine a decomposition approach with a probabilistic approach. Our method is pretty similar to Discrete Search Leading continuous eXploration (DSLX) [5], [6] but instead of only guiding the search process, it restricts the search space. The algorithm splits the environment into a set of cells that allows, using a shortest-path search technique, to define a corridor in which the search for a solution will take place, and then to apply one of the current most powerful and used path planning algorithm: the Rapidly-exploring Random Trees (RRT) algorithm [7].

First, we describe the basic RRT algorithm and some of the extensions that have been proposed. Then, we present the core of Cell-RRT, the proposed planner. Finally, we compare the Cell-RRT performances with the basic RRT algorithm used without decomposing the environment and we study how Cell-RRT scales according to the settings of its configuration parameters.

J. Guitton is with ONERA, the French Aerospace Lab, Toulouse Research Center, 2 avenue Edouard Belin, F-31055 Toulouse, France and Université de Toulouse ; UPS, INSA, INP, ISAE ; LAAS ; F-31077 Toulouse, France, julien.guitton@onera.fr

J.-L. Farges is with ONERA, the French Aerospace Lab, Toulouse Research Center, 2 avenue Edouard Belin, F-31055 Toulouse, France, jean-loup.farges@onera.fr

R. Chatila is with CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse, France and Université de Toulouse ; UPS, INSA, INP, ISAE ; LAAS ; F-31077 Toulouse, France, raja.chatila@laas.fr

II. RRT

The algorithm is a probabilistic path planning algorithm. This is an incremental method that rapidly explores the environment by trying to iteratively connect random configurations until reaching the goal configuration.

The core of the RRT method is presented in Algorithm 1. Starting from an initial configuration q_{start}, the aim of this algorithm is to find a sequence of configurations allowing the mobile robot to reach the final configuration q_{goal}. The algorithm is composed of a loop that calls three main functions: chooseTarget() which selects a random configuration, nearestNeighbor() which returns the nearest configuration of the randomly chosen one that is already in the tree, and extend() which creates a new configuration.

For each iteration of the loop, a new configuration q_{target} is randomly drawn. Then, the tree node corresponding to the nearest configuration $q_{nearest}$ is selected. Finally, a new configuration q_{new} is created by extending the selected configuration towards the drawn configuration according to a beforehand fixed distance. The algorithm stops when the new configuration can be connected to the goal, in this case a solution path is found, or when the maximum number K of drawn configurations is reached. This version of the RRT algorithm is called RRT-Extend [7].

Another variant, called RRT-Connect [8], consists in trying to connect the nearest configuration directly to the drawn configuration. In this case, the nearest configuration is extended towards the random configuration until connection or until encountering an obstacle.

In order to accelerate the search for a solution path, one possible improvement of previous algorithms is to use two trees instead of a single one [7], [8]. The first tree is initialized with the initial configuration q_{start} and the second one with the final configuration q_{goal}. The search stops when both trees meet each other. When the RRT-Extend method is implemented with the two trees, the algorithm is named RRT-ExtExt. When the RRT-Connect is implemented, it is named RRT-ConCon.

Algorithm 1 RRT-EXTEND(q_{start}, q_{goal})

```
1: $T$.init($q_{start}$);
2: $i$ = 0; $q_{new}$ = $q_{start}$;
3: while ($i < K$ and canConnect($q_{new}$, $q_{goal}$) == $null$) do
4:     $q_{target}$ = chooseTarget();
5:     $q_{nearest}$ = $T$.nearestNeighbor($q_{target}$);
6:     $q_{new}$ = extend($q_{nearest}$, $q_{target}$);
7:     if ($q_{new} \neq null$) then
8:         $T$.add($q_{new}$);
9:     end if
10:    $i$ = $i$ + 1;
11: end while
12: return $T$;
```

The RRT algorithm has received a special attention and many improvements have been proposed. These improvements, for highly-constrained environments [9], aim at guiding more efficiently the construction of the exploration tree by taking into account information about the environment [10], [11] or by specifically treating narrow passages [12], [13]. Another research track is to extend RRT in order to apply it to dynamic environments [14], [15].

III. The Cell-RRT planner

One of the common critics done to RRT planners is their slowness when the search is done in complex environments such as labyrinthian environments. Indeed, in the case of the basic RRT-Extend method, the algorithm uniformly extends a network of possible configurations or towards the goal if the randomization is biased, but it does not take into account the environment topology.

In order to overcome this lack of efficiency, we propose a method allowing to restrict the configuration space to a *corridor* in which both ends contain the initial and the final configurations. This corridor is built from a set of cells that are selected using the A* search algorithm [16].

The implemented algorithm is organized into two parts. First, the environment is divided into cells which are associated to indicators of traversability. Paths are searched in the cells adjacency graph by optimizing a cost function linked to these indicators, allowing to limit the search to few parts of the environment. The result is a set of cells for which two waypoints are defined: an entry waypoint and an exit waypoint. Then, the RRT algorithm is applied to each cell providing a set of trees. A solution path between the initial and the final configuration is found by connecting trees together.

In a first step, we present the models we chose to represent the environment and the mobile robot. Then, after having presented the overall functioning of the algorithm, we detail its two main functions: the restriction of the environment space to a corridor and our implemented version of RRT allowing to obtain a solution path according to the kinematic model of the robot.

A. Modeling

1) Robot modeling: The mobile robot used to illustrate Cell-RRT is a car-like vehicle. The attitude of such robot can be described by three configuration variables (x, y, θ): x and y are the cartesian coordinates of the robot position and θ is its heading. The robot is considered to move with a constant speed and its turning angle is bounded:

$$-\phi_{max} \leq \phi \leq \phi_{max} \tag{1}$$

The robot kinematic model is defined as follows:

$$\begin{cases} \dot{x} &= v.cos(\theta) \\ \dot{y} &= v.sin(\theta) \\ \dot{\theta} &= \frac{v}{L}tan(\phi) \end{cases} \tag{2}$$

where v is the speed and L is the vehicle's axle length. Computing a robot movement corresponds to integrating this model on a time interval Δt. In order to limit the search space, changes of turning angle are modeled by a three-valued command:

$$\phi \in \{-\phi_{max}, 0, \phi_{max}\} \tag{3}$$

2) Modeling transitions between two configurations: A transition is represented by the pair $< \phi, \Delta t >$:

$$< x, y, \theta > \xrightarrow{<\phi,\Delta t>} < x', y', \theta' > \tag{4}$$

where Δt is the movement duration.

B. Planner main loop

During the first step, the planner splits up the environment into cells and computes a corridor between the initial and the final configurations. Within this corridor, subtrees are computed for every cells using the waypoints between adjacent cells of the corridor. If a cell does not contain a solution, the link between this cell and the next one is removed from the adjacency graph in order to prevent the A* algorithm from choosing this cell. The loop is iterated until obtaining a solution path or a failure. A failure occurs when it is not possible to find a corridor and so a solution path. In case of success, the solution path is built by gathering the extracted paths from the trees of each cell belonging to the corridor.

C. Environment division and corridor computation

1) Division and computation of the traversability: The environment is discretized into a set of cells. Each cell is stored in a table and will be considered thereafter as a full environment, i.e., an initial and a final configuration will be associated to it. If this cell is selected so that it belongs to the corridor, a path between these two configurations will have to be computed.

For each cell c_i of the environment, a weight corresponding to cell traversability $t(c_i)$ is computed. This weight is the ratio of the obstacles occupation compared to the free space. A weight of 0 corresponds to a cell without obstacle, i.e., a perfectly traversable cell. We consider a traversability threshold prohibiting a cell to belong to the corridor. On the other hand, according to the chosen crossing direction, the obstacles configuration can prevent a cell to be crossed, even with a low ratio.

2) Definition of waypoints: In order for a cell to belong to the solution corridor it has to contain a solution path. Nevertheless, at this stage, it is not possible to compute the path because the purpose of this part is to quickly and effectively reduce the search space. In order to compute a path, a cell must have waypoints defined on its boundaries. In addition, two adjacent cells may have the same waypoint on their common boundary.

For 2D environments, waypoints are computed as follows: for each boundary of the considered cell, free segments whose length is greater than a threshold are identified. These segments are recursively divided until obtaining a minimal length. For each subsegment seg_i, a square surface S_{seg_i} with no obstacle along the segment mediator is computed:

$$S_{seg_i} = min(sL(seg_i), mL(seg_i))^2 \tag{5}$$

where sL and mL are respectively the length of the segment and the length of its free mediator. The waypoint associated to the considered cell boundary is the center of the segment having the largest square surface.

This method allows to take into account the length of the passage between two cells in addition to its width and thus it avoids narrow passages: the larger the surface is, easier the planner will find configurations allowing to connect two adjacent cells according to the robot kinematic constraints.

3) Corridor computation: After computing the set of waypoints, a solution corridor between the cell containing the robot initial configuration and the cell containing the final configuration is computed using the A* algorithm. A possible heuristic for the choice of the next cell c_i of the path is the euclidian distance between the center of the cell and the center of the destination cell c_{goal} and, as a transition cost function, the distance $d(.,.)$ between the two cells ponderated by the traversability $t(.)$ of the cell:

$$\begin{cases} g(c_i) = g(c_{i-1}) + (d(c_{i-1}, c_i) * (1 + \gamma * t(c_i))) \\ h(c_i) = d(c_i, c_{goal}) \end{cases} \tag{6}$$

where γ is a positive tuning parameter. If this cell does not contain an entry waypoint on its boundary with the previous cell c_{i-1}, then it is abandoned and the next cell with lower cost is selected. The solution corridor is obtained when the algorithm reaches the destination cell, i.e., the cell containing the final configuration.

D. Trees construction: RRT-ExtExt

Once the search space has been restricted to a corridor, a path respecting the kinematic constraints of the robot is computed. The algorithm is divided into three main functions: selection of a random configuration q_{target}, choice of the nearest configuration $q_{nearest}$ which belongs to the tree, and extension of $q_{nearest}$ towards q_{target} which allows to obtain a new configuration q_{new}.

Selection of random configurations In order to guide the tree expansion towards the final configuration, the random draw can be non-uniform. According to a probability $probabilityGoal$, the algorithm will choose configurations that are less than $radius$ units of the final configuration. Otherwise, according to a probability $(1 - probabilityGoal)$, this draw is uniform. Algorithm 2 describes this method.

Algorithm 2 chooseTarget()

```
1: if (random([0,1]) < $probabilityGoal$) then
2:     $q_{target}$ = randomNodeAroundGoal($q_{goal}$, $radius$);
3: else
4:     $q_{target}$ = randomNode();
5: end if
6: return $q_{target}$;
```

Choice of the nearest node. The aim of this algorithm is to choose a node whose configurations will be extended towards the randomly drawn configuration. For the selection of this node, the heuristic we use is the euclidian distance between two configurations. Thus, the nearest tree node of q_{target} is selected. However, it has to be farest than a minimal distance of the q_{target} configuration so that the extension respects the kinematic constraints.

Adding a new configuration. This addition is done by extending the $q_{nearest}$ configuration towards the q_{target} configuration. Algorithm 3 presents this extension. The first step is to compute ϕ, i.e., to compute whether the robot should go ahead in a straight line to reach the target configuration or if it has to turn on the right or left. In order to choose the command to apply, we checked if the target is in a restricted visibility cone $minCone$ of the robot. If the target is in this cone, the robot has to go ahead, otherwise it has to turn in order to get closer to the target. Then the new configuration is generated. It corresponds to the configuration where the robot will be after a time interval Δt. This interval is the incremental step of the algorithm. Finally, the algorithm checks the feasibility of the motion between the $q_{nearest}$ configuration and the new configuration, i.e., the new configuration should not be located within an obstacle and a path must exist between them.

Algorithm 3 extend($q_{nearest}$, q_{target}, Δt)

```
1: $\alpha$ = (-$\theta(q_{nearest})$ + direction($q_{nearest}$, $q_{target}$)) % $2\pi$;
2: if ($|\alpha| > minCone$) then
3:     $\phi$ = $\phi_{max}$ × signum($\alpha$);
4: else
5:     $\phi$ = 0;
6: end if
7: $q$ = generateConfiguration($q_{nearest}$, $\phi$, $\Delta t$);
8: if (notInObs($q$) and isConnectable($q_{nearest}$, $q$)) then
9:     $q_{new}$ = $q$;
10: else
11:     $q_{new}$ = null;
12: end if
13: return $q_{new}$;
```

Generation of new configurations. generateConfiguration (line 7) integrates the kinematic model described by equation (2) over time intervals Δt.

Trees connection: The CSC method. The developed RRT-ExtExt algorithm generates simultaneously two trees and stops when the configurations associated with two leaf nodes of these trees can be connected. To connect these two trees, we use the CSC (Circle-Segment-Circle) algorithm.

The modeled robot corresponds to a *Dubins Car* robot. Indeed, its speed is always positive and constant, and its turning radius is fixed to the minimal radius. Then, the robot trajectory can be modeled by Dubins curves. It has been shown in [17] that the shortest path between two configurations can always be expressed by a combination of at most three movements. Possible movements are "go straight" (S), "turn right" (R) or "turn left" (L).

The CSC method consists in using only sequences {LSL, LSR, RSL, RSR} to connect two configurations, i.e., the robot must first perform a turn, then a travel in straight line, and finally a second turn to reach the target configuration.

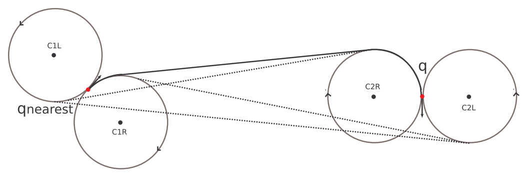


Fig. 1. 4 possible tangents for the connection of 2 directed circles

The process is as follows: two circles with radius equal to the minimum turning radius are computed for both configurations to connect. The bitangents allowing to connect the circles together are defined. Only four tangents are possible (see Fig. 1) because the circles are directed according to the robot heading value of the configurations. Then, the four trajectories {LSL, LSR, RSL, RSR} are computed. Intermediate configurations which link together the $q_{nearest}$ and q configurations are generated from the movements sequence of minimal length.

IV. Experiments and Results

In order to present the performances of our Cell-RRT path planner and to compare the different possible parameter settings, we led experiments on a set of 20 randomly generated environments. Presented results are averages of values for 1000 runs on each environment and for each environment decomposition.

For each experiment, the number of drawn configurations is limited to 12000 for the whole environment. Thus, the maximum number of drawn configurations per cell is:
$K = 12000/number\_of\_cells$.

In a first step, we show the advantages of using environment decomposition on the computation time and the path length by varying the size of cells. Indeed, computing the optimal decomposition granularity is not easy. We use as an upper bound the environment size. Lower bound must be enough to permit the connection of two waypoints according to the robot kinematic model, i.e., squares with a side length of 2 minimal radius. Then, we evaluate the planner performances in terms of replanning rate, computation time and traveled distance when the environment decomposition is combined with various random bias, cost functions for the A* algorithm and with the reuse of trees and path segments already computed.

The aim of these experiments is to analyse the impact of these parameter settings on computation time, path length and necessary replannings while avoiding failures.

A. Advantages of the environment decomposition

First experiments aim at demonstrating the advantages of dividing the environment into a set of cells then reducing the search space to a corridor. We study the impact of our approach on the computation time of a solution path as well as on the traveled distance.

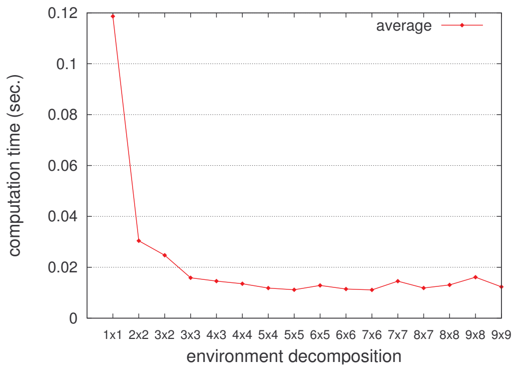
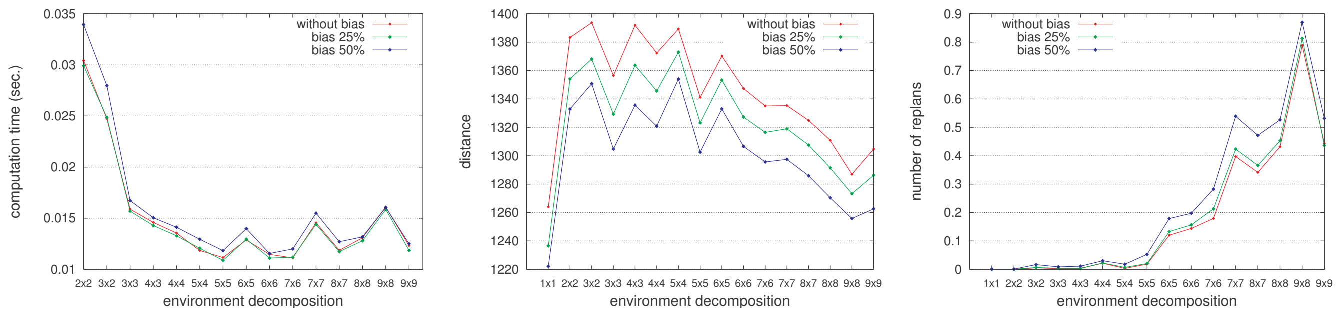


Fig. 2. Influence of decomposition granularity on computation time.

On Fig. 2, we see that the decomposition into 2x2 cells leads to a decrease of computation time of 75% and a decomposition into 5x5 cells reduces this time of 91% compared to the time without decomposition. This optimal decomposition is related to the used environments and seems not to be general. Indeed, For a smaller size of cell, the computation time is subject to slight variations suggesting that some cell sizes for some environments do not allow to obtain directly a solution path and require several replanning loops. This is reinforced by the evaluation of the number of replannings presented on Fig. 3c (red curve). For a size of cells greater than the one giving the minimal time, we can make the assumption that the time decreases because the explored surface decreases with the cell size. However, such decomposition of the environment induces an augmentation of the traveled distance compared to the path length for an environment without decomposition (Fig. 3b, red curve). This degradation decreases with the cell size because the corridor between the initial configuration and the final configuration is more direct. Nevertheless, this degradation persists for a minimum cell size. The analysis of the graph representing the distance indicates that it varies between 1,5% (9x8 cells) and a little more than 10% (3x2 cells).

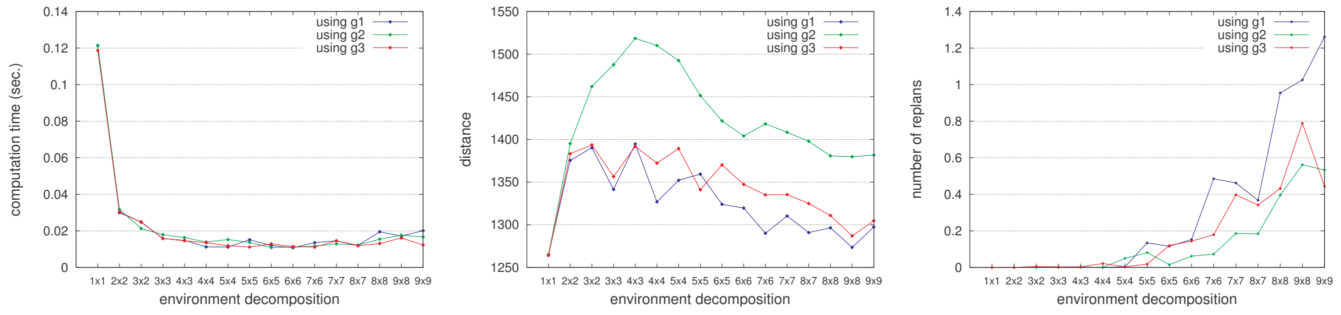
B. Influence of the drawing method

The aim of using a bias is to encourage expansions of RRT trees towards final configurations. In these new experiments, we compare a uniform random draw with methods in which



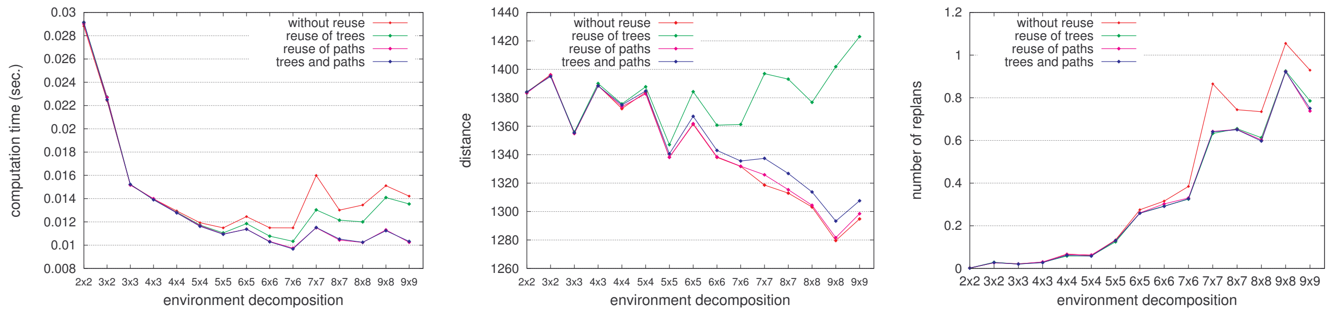
(a) computation time (b) distance (c) replanning

Fig. 3. Influence of drawing methods on computation time, distance and replanning.



(a) computation time (b) distance (c) replanning

Fig. 4. Influence of cost functions on computation time, distance and replanning.



(a) computation time (b) distance (c) replanning

Fig. 5. Influence of trees and segments reuse on computation time, distance and replanning.

probabilities of selecting new configurations in a 10 units perimeter around the target are 25% and 50%.

According to graphs on Fig. 3c, the more the bias increases, the more the number of necessary replannings increases and, globally, the probability of finding a solution decreases. This is particularly true for the division granularities 8x7 and 8x8 which do not seem best suited to some of the used environments and induce more replannings.

Fig. 3a shows that using a biased random draw increases the computation time. This increase is around 10 to 15%. On the other hand, this bias allows to improve the quality of the solution without division, and to limit the loss of quality when the environment is decomposed. It is interesting to note that the performance gain in terms of traveled distance is constant whatever the cell size and it is about 3% for a bias of 50%. For a division into 9x8 cells, the length of solution paths is lower than the one obtained without division and without bias.

C. Influence of the A* cost function

Another adjustable parameter of the planner is the cost function used by the A* method for the corridor search. Indeed, two kinds of information can be taken into account: the distance between the new cell and the target cell as well as the traversability of this new cell. We experimented three different cost functions:

- g1 : only the euclidian distance is used ($\gamma = 0$);
- g2 : only the traversability is used (γ = high value);
- g3 : distance is added to the traversability ratio ($\gamma = 1$).

This third function is used in experiments presented in previous figures and therefore is used as a reference to g1 and g2 in next figures.

Fig. 4c depicts the number of replannings when using each of the three cost functions. The function g2 (traversability) avoids more replannings than g1 and g3 when the number of cells is increasing. This is because the algorithm searches in priority for a path through cells with few obstacles.

On graphics of Fig. 4, we note that the influence of the cost function on the computation time is hard to quantify. The use of function g1 seems to increase the time when the number of cells increases due to higher replanning requirements. In the other hand, the choice of this function strongly influences the traveled distance. Whatever the granularity of the environment division, function g2 shows the worst performances because, in order to avoid obstacles, the corridor between the initial configuration and the final configuration is less direct. This loss of performance is greater than 20% in the worst case, unlike 10% of degradation obtained when using function g3. The traversability seems to be more useful to eliminate cells from the graph when it is used as a threshold than as a criterion for the corridor search.

D. Reuse of trees and path segments

Reducing the environment to a corridor allows to reduce the search space of the RRT algorithm and thus to decrease the computation time. However, there is not necessarily a path connecting the initial configuration to the final configuration. Replannings can be necessary in order to define a new corridor before searching for a new path. This new corridor may contain cells that belonged to the old corridor. In this case, if the entry and exit waypoints of a cell are the same and if it exists a path to cross it, then the cell does not need to be recomputed. In addition, previously computed trees can be reused in order to limit the solution search process. We compared results for the basis version with the values obtained when the planner reuses previously computed trees, previously defined path segments, and trees and path segments together.

Graphics on Fig. 5 indicate that such a reuse decreases the computation time when the decomposition granularity increases. This effect is mainly due to the reuse of path segments. However, we note that reuse offers less optimal solutions in terms of path length. This is particularly true when trees are reused. Indeed, the more the reused tree is developed, the lower is the probability that the nearest node is on a "direct" branch. Moreover, this reuse improves the replanning rate.

V. Conclusion and future works

In this paper, we have presented the Cell-RRT algorithm that allows to plan efficient travels for a mobile robot. This planner combines a decomposition method with a probabilistic algorithm. The objective of the decomposition step is to reduce the search space to a corridor in which the RRT algorithm is used to find a solution path.

Experimental studies of our algorithm highlight the advantages of this decomposition step by allowing a reduction of the computation time by around 75%, which is consistent with DSLX results. However, this decomposition can lead to a degradation of the solution quality up to 15%. Nevertheless, a judicious choice of the planner parameter settings can reduce the loss of quality and even improve the computation time.

We showed, on the one hand, that using a bias is beneficial to the path length but harmful to the computation time and, while on the other hand, that using traversability as a cost criterion for the A* algorithm is harmful to the path length. It allows however to avoid replannings with a reduced influence on the computation time. Finally, reusing trees and paths improves the computation time but it leads to a slight degradation of the solution length.

One extension of this work is the integration of this algorithm into a hybrid planning architecture. Another interesting research track we did not explore is how to automatically choose the best decomposition granularity according to the environment topology.

References

[1] J. Guitton, J.-L. Farges, and R. Chatila, "A planning architecture for mobile robotics," in *1st Mediterranean Conference on Intelligent Systems and Automation*, 2008, pp. 162–167.

[2] J. Latombe, *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[3] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[4] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[5] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Discrete search leading continuous exploration for kinodynamic motion planning," in *Proceedings of Robotics: Science and Systems*, 2007.

[6] ——, "Impact of workspace decompositions on discrete search leading continuous exploration (DSLX) motion planning," in *IEEE International Conference on Robotics and Automation*, 2008, pp. 3751–3756.

[7] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," in *IEEE International Conference on Robotics and Automation*, vol. 1, 1999, pp. 473–479.

[8] J. Kuffner and S. LaValle, "RRT-Connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation*, 2000, pp. 995–1001.

[9] M. Kalisiak and M. von de Panne, "RRT-blossom: RRT with a local flood-fill behavior," in *IEEE International Conference on Robotics and Automation*, 2006.

[10] S. Rodriguez, X. Tang, J.-M. Lien, and N. Amato, "An obstacle-based rapidly-exploring random tree," in *IEEE Int. Conference on Robotics and Automation*, 2006, pp. 895–900.

[11] B. Burns and O. Brock, "Single-query motion planning with utility-guided random trees," in *IEEE International Conference on Robotics and Automation*, 2007.

[12] M. Strandberg, "Augmenting RRT-planners with local trees," in *IEEE International Conference on Robotics and Automation*, 2004, pp. 3258–3262.

[13] L. Zhang and D. Manocha, "An efficient retractation-based RRT planner," in *IEEE International Conference on Robotics and Automation*, 2008, pp. 3743–3750.

[14] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *IEEE International Conference on Robots and Systems*, 2002.

[15] D. Ferguson, N. Kalra, and A. Stenz, "Replanning with RRTs," in *IEEE International Conference on Robotics and Automation*, 2006, pp. 1243–1248.

[16] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," in *IEEE Trans. on Systems Science and Cybernetics*, 1968, pp. 100–107.

[17] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, pp. 497–516, 1957.