Cell-based Probabilistic Roadmaps (CPRM)
for Efficient Path Planning in Large Environments

Klaas Klasing   Dirk Wollherr   Martin Buss
Institute of Automatic Control Engineering
Technische Universität München
80290 Munich, Germany
{kk,dw,mb}@tum.de

Abstract— This paper presents a novel sampling-based path planning method called *Cell-based Probabilistic Roadmaps* (CPRM). The algorithm has both anytime and replanning characteristics and is superior to classical approaches in that it incrementally builds and reuses the roadmap. Because it maintains a graph structure that is tailored to the queries, CPRM is very fast in answering repeated queries, e.g. in replanning scenarios. Different design parameters allow to control and fine-tune the behavior of the algorithm.

I. INTRODUCTION

Over the last decade *Probabilistic Roadmaps* (PRMs) [1] have become the favored method for solving multiple query path planning problems in high-dimensional configuration spaces. Several extensions to the method have been developed to foster the use of PRMs in different planning environments.

This paper introduces a novel PRM variant called *Cell-based Probabilistic Roadmaps* (CPRM) that tailors the PRM construction process to a set of specific queries. Essentially, the method uses a global grid cell structure to grow the PRM only in individual cells relevant to a certain query. This approach is motivated by scenarios in which a series of queries is known before planning, e.g. assembly tasks, surveying, tracking etc., and in which the number of nodes in N can grow very large.
CPRMs offer several advantages over the classic PRM approach:

- Given a planning query, the algorithm starts planning right away. It does not need to wait for the construction of a roadmap for the entire configuration space.
- The algorithm reuses previously built parts of the roadmap. This makes it efficient for replanning towards the same goal.
- If a solution does not satisfy a quality criterion, the algorithm keeps searching for better paths.
- The cell structure allows for a memory-bounded version of the algorithm.

CPRMs can be used both as a replanning and an anytime method. If patches of the roadmap are invalidated because obstacles have changed place, the corresponding cells can be quickly identified and the PRM can be regrown in these cells. This allows for replanning in dynamically changing environments. CPRMs are an anytime path planning method

in the sense that – once a graph exists – the method instantly returns the suboptimal path on the existing graph structure. As more time is made available to the algorithm, it continues to find better solution paths until some path quality criterion is satisfied or a new query is issued.

This paper is structured as follows: Section II gives a non-exhaustive overview of the PRM method and outlines drawbacks of existing algorithms. Section III explains the ingredients of the CPRM algorithm and gives a detailed description in pseudo-code. In Section IV simulation results are presented that verify the efficiency of the method. Possible future work and extensions to the method are outlined in Section V. Section VI concludes this paper.

II. METHOD AND MOTIVATION

A. The Probabilistic Roadmap Method

The PRM method captures the connectivity of C_{free}, the unobstructed part of the robot configuration space C, by repeatedly and randomly sampling configurations in C and testing them for collision. The free configurations are connected by means of a *local planner* to form a graph structure $G = (N, E)$, with nodes N and edges E, that is called the *roadmap*. Planning queries consist of an initial configuration c_{init} and a goal configuration c_{goal}. To answer a query, these configurations merely need to be connected to the existing roadmap and a suitable graph search algorithm such as A* is executed to find the shortest path on G.

In its original form [1] the PRM method is holonomic, deterministic, static and the roadmap creation is a pure preprocessing step. To cope with these limitations, several extensions of classic PRMs have been developed. Non-holonomic planning problems can be handled by using more powerful local controllers, such as *Rapidly-exploring Random Trees* (RRTs), see [2], as demonstrated in [3] and [4]. In [5] PRMs are adapted to handle uncertain representations of the environment. To use PRMs in dynamic environments, researchers have developed methods such as FADPRM [6] or the approach presented in [7]. Both of these methods rely on dynamic anytime graph search algorithms, such as AD*, see [8], and dynamically update edges of the graph that have been invalidated by moving obstacles. The former method also solves the problem of having to create the entire PRM before planning is commenced.

B. Motivation

The general drawback of PRMs is that collision checking is computationally costly. Thus, pre-computing the entire PRM before planning can be prohibitive, especially if parts of the roadmap need to be rebuilt due to changes in the environment. This has led to the development of lazy collision checking variants of PRM [9], [10], which, however, are single-shot methods and delay collision checking to the time of planning. Pre-computing the PRM for the entire configuration space has other drawbacks as well: Despite efficient sampling techniques, the resulting graph structure may grow very large, which can result in significant slowdown of graph searching algorithms. For demonstration purposes, Figure 1 shows the performance of the A* algorithm in a large 2D environment with 400 obstacles and roadmap containing 10,115 nodes and 24,813 edges. A* expands a total of 5,164 nodes, which is more than 50% of the graph. If the search graph is tailored to the actual query, this number can be reduced significantly, as is shown in this paper. To the authors' knowledge there exist few techniques to combine sampling-based path planning algorithms with cell-based approaches as a meta-structure. In [11] an interesting method is presented, that combines RRTs as local controllers with the parti-game cell decomposition technique (PDRRTs). Another method called *Probabilistic Cell Decomposition* (PCD) is presented in [12]. Both approaches are similar in that they start out with a coarse structure of rectangular cells, which is then iteratively refined by splitting cells that contain "difficult" regions surrounding obstacles. Cell splitting is performed during query execution, so both methods operate as single-query variants[1]. There is no mentioning of anytime extensions, although at least PCD seems to be extensible in this direction. PDDRTs seem to require extra post-processing effort as they yield suboptimal path lengths when compared to RRTs. Another drawback is that for both methods graph search is performed on the cell structure itself, meaning that for difficult scenarios there is a considerable overhead until the needed cell resolution has been reached.

III. THE CPRM ALGORITHM

The CPRM algorithm has a number of core ingredients. The following subsections describe these ingredients and outline how different parameters influence the algorithm behavior.

A. Cell Structure

As the name indicates, the Cell-based PRM method starts by slicing the configuration space into cells. Square cells were chosen for CPRM, because they nicely generalize to n dimensions and are computationally convenient, however, it is possible to use other cell shapes (e.g. *hexagonal* for 2D problems). Each cell has the following properties:

- `cell.origin` - the origin of the cell.
- `cell.size` - the cell size (width of a square cell).

---

[1]Furthermore, PCD is lazy in that it assumes a cell to be completely unoccupied until disproven.
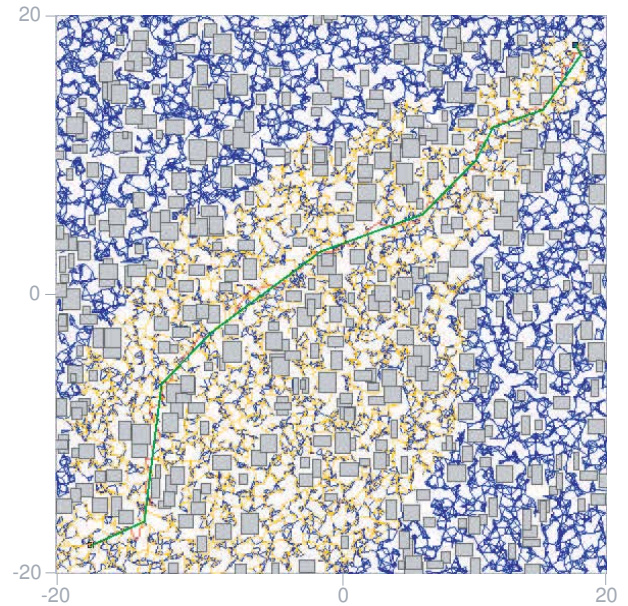

Fig. 1. A 2D environment with 400 obstacles and a regular PRM with 10,115 nodes and 24,813 edges. The initial and goal configuration are shown as squares, the solution path found by A* is depicted by a thick line and the edges traversed by A* are shown as the shaded area around the solution path. A* expands roughly 50% of all nodes.

- `cell.numNodes` - the number of free configurations contained in this cell.
- `cell.numTrials` - the number of times the algorithm has tried to sample in this cell.
- `cell.numComponents` - the number of components of nodes in this cell.
- `cell.connectedness` - a number that reflects how connected the nodes in this cell are.
- `cell.dist` - the distance measure of the cell.
- `cell.occupancy` - the occupancy measure of the cell.
- `cell.value` - the value of the cell used in the priority queue.

The algorithm operates with two cell lists: The $OPEN$ list contains the cells in which the PRM needs to be grown and the $CLOSED$ list those that are sufficiently connected or contain the maximum number of nodes allowed.

B. Construction Parameters

In addition to the standard parameters for PRM creation such as `maxDist`, `maxNeighbors` etc., see [1], CPRM has the following construction parameters:

- `numCellsPerAxis` - the grid resolution.
- `maxNodesPerCell` - the maximum number of nodes per cell.
- `nodeIncrementPerCell` - the number of nodes added to a cell each time the PRM is grown.
- `occupancyThresh` - the threshold at which a cell is considered to be unoccupied.
- `w1` - weight of the distance measure.
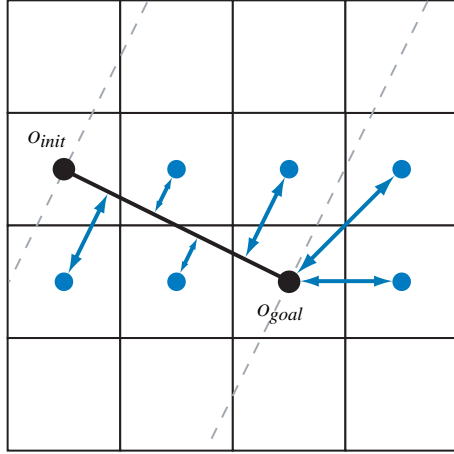- `w2` - weight of the connectedness measure.

Fig. 2. Distance between some neighboring cells and the line (o_{init}, o_{goal}) in a 2D environment.

These parameters are used to build and control the grid cell structure which in turn controls the growth of the roadmap in individual cells.

C. $OPEN$ List Criteria

The key idea behind CPRM is to grow the roadmap only in cells which are expected to contain the final solution path. To achieve this, the $OPEN$ list is prioritized according to two criteria.

1) Distance Measure: The first criterion is the distance of each cell from the direct line connecting o_{init}, the origin of the cell containing c_{init}, and o_{goal}, the origin of the cell containing c_{goal}. Figure 2 illustrates how this distance is computed for cells in a 2D environment. If the origin of a cell falls within the region bounded by the two dashed lines, its distance measure is the orthogonal distance to the line, otherwise its distance measure is the distance to the closest endpoint of the line[2]. Although the actual solution path may not lead along the direct line (o_{init}, o_{goal}), this criterion helps the algorithm to focus its search in a fashion similar to A*. Another advantage of this measure is that for every query it only needs to be computed once for every cell, in other words it can be calculated during initialization.

2) Connectedness Measure: The second criterion for prioritization is the connectedness of a cell. If a cell contains many separate components, it is preferable to keep sampling in this cell until the number of components has been reduced or until `maxNodesPerCell` has been reached. Connectedness is defined as

$$\texttt{cell.connectedness} = \frac{\texttt{cell.numNodes}}{\texttt{cell.numComponents}}. \quad (1)$$

The overall value of a cell is computed as

$$\texttt{value} = \texttt{w1} \cdot \texttt{dist} + \texttt{w2} \cdot \texttt{connectedness} \quad (2)$$

The $OPEN$ list is sorted by value in ascending order every time the PRM has been grown in a cell. The weights

---

[2]In this paper a Euclidean distance metric is used. Depending on the C-space properties different metrics may be chosen, see [13].

`w1` and `w2` greatly influence the behavior of the algorithm. For large `w1`, the CPRM will grow in a very focused manner around the direct line between o_{init} and o_{goal}. For large `w2` the algorithm explores more and preferably picks cells in which the PRM has not been grown yet or which contain many separate components.

D. $CLOSED$ List Criteria

Cells that have been filled with `maxNodesPerCell` nodes are put on the $CLOSED$ list. Furthermore, the degree of occupancy of a cell is taken as a criterion for stopping growth in the cell. Occupancy[3] is defined as

$$\texttt{cell.occupancy} = \frac{\texttt{cell.numNodes}}{\texttt{cell.numTrials}}. \quad (3)$$

When `cell.occupancy > occupancyThresh`, the cell is put on the $CLOSED$ list. The rationale behind this is that the PRM does not need to be grown much in cells that are almost unoccupied. This raises the non-trivial question of how many samples per cell are needed until one can be sure that the occupancy measure reflects the true occupancy of a cell. An analysis using Chernoff bounds is possible but is not part of this paper. The reader is advised that `nodeIncrementPerCell` must be chosen sufficiently large to guarantee that the occupancy estimate is accurate enough after the first round of sampling. The occupancy measure of a cell must be updated every time the PRM is grown in the cell.

E. Algorithm Description

Like any other sampling-based path planning algorithm, CPRM takes as a problem description a pair of two configurations c_{init} and c_{goal} (or a list of such pairs) as well as an environment description containing the obstacles, the robot, a collision test and a local planner. First of all, the algorithm creates the cell structure, initializes the lists and puts the parent cell of n_{init} on the $OPEN$ list. It then repeats the following steps until either n_{init} and n_{goal} belong to the same component, or the $OPEN$ list is empty, in which case failure is reported:

1) Take the first cell (having the lowest value) from the $OPEN$ list.
2) Grow the PRM in this cell.
3) Perform random walks in the cell to reduce the number of components.
4) Update the cell statistics.
5) If the cell appears unoccupied or contains the maximum number of nodes put it on the $CLOSED$ list, else put it back on the $OPEN$ list.
6) Add the neighbors of the cell to the $OPEN$ list, unless they are on the $CLOSED$ list.

Once n_{init} and n_{goal} are in the same component, the algorithm finds the shortest path between the two nodes and publishes it as a solution. If this path does not satisfy a specified path quality condition, steps 1) - 6) are repeated and a new shortest path search is started every k iterations, where k is specified by the user. As soon as a newly found path satisfies the quality condition it is published as the

---

[3]Note that the definition is counter-intuitive, i.e. an occupancy value of 0 corresponds to a completely occupied cell and vice versa.
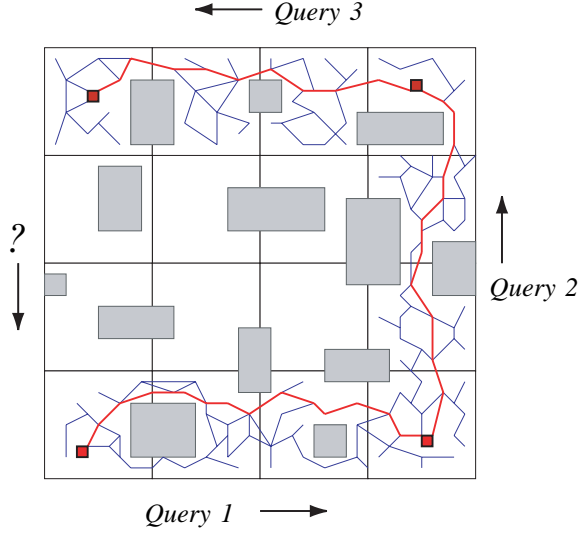
Fig. 3. A planning scenario in which the existing graph does not yield the optimal solution for the query (from the top left cell to the bottom left cell). CPRM will instantly publish the suboptimal solution and the continue to grow the roadmap until the direct solution is found.

final solution and the algorithm terminates. This anytime characteristic of CPRM is essential for the planning scenario shown in Figure 3.

A detailed pseudo-code description of CPRM is shown in Algorithm 1.

IV. SIMULATION AND RESULTS

A. Implementation Details

The CPRM algorithm was implemented in C++ on a Pentium machine with 2.1 GHz and 2GB of RAM. The simulation environment is a dimensionless 40x40 2D configuration space filled with 400 rectangular obstacles of random widths and heights $\in [0.4, 1.4]$. The obstacles are placed at random across the configuration space and may overlap. In every corner of the configuration space a 3.3x3.3 square is left unoccupied to guarantee that queries can be started from there.

Random walks were used in both the PRM implementation and the CPRM implementation. The pseudo-code in algorithm 1 denotes this on line 33. It is a known fact that cycles in the roadmap are an essential factor for finding good solution paths [14]. However, the method for adding useful cycles presented in [14] proved be rather slow. Instead of first constructing a forest graph structure and then manually adding cycles, the following method proved to be faster and more suitable for our purposes: When connecting a new node n_{new} to its neighbors, connect it to as many as 3 neighbors, even if they are within the same component already. This approach yielded good speed and connectedness of the final graph. As a spatial data structure *quadtrees* were chosen instead of *kd-trees*, because they do not require tree balancing when a node is inserted and are easy to implement for a 2D configuration space. It is assumed that *kd-trees* will be

---

Algorithm 1 The CPRM Algorithm

```
1:  INITIALIZE()
2:  for all (cell ∈ cells) do
3:      cell.dist = distPointQueryLine(cell.origin);
4:  end for
5:  OPEN = {parentCell(n_init)};
6:  CLOSED = ∅;

7:  GROWPRMINCELL(cell)
8:  numSamples = 0;
9:  while (numSamples < nodeIncrementPerCell) do
10:     sample random configuration c_new in cell;
11:     cell.numTrials++;
12:     if (isFreeConfig(c_new)) then
13:         add node n_new to N;
14:         connect n_new to neighbors, add edges to E;
15:         update cell.numComponents of cell;
16:         numSamples++;
17:     end if
18: end while
19: cell.numSamples += numSamples;
20: updateOccupancy(cell);
21: updateConnectedness(cell);
22: updateValue(cell);

23: SOLVEQUERY(n_init,n_goal)
24: Initialize();
25: add n_init, n_goal to N;
26: connect n_init, n_goal to neighbors, add edges to E;
27: loop
28:     if (OPEN = ∅) then
29:         report failure;
30:     end if
31:     cell = takeFirst(OPEN);
32:     GrowPRMInCell(cell);
33:     PerformRandomWalksInCell(cell);
34:     if (cell.occupancy > occupancyThresh or
            cell.numSamples ≥ maxNodesPerCell) then
35:         CLOSED ← cell;
36:     else
37:         OPEN ← cell;
38:     end if
39:     for all (neighbor ∈ neighbors(cell)) do
40:         if (neighbor ∉ CLOSED) then
41:             OPEN ← neighbor;
42:         end if
43:     end for
44:     sortAscendingByValue(OPEN);
45:     if (parentComp(n_init) = parentComp(n_goal)) then
46:         path = findShortestPath(G);
47:         if (path satisfies quality condition) then
48:             return path;
49:         else
50:             publish path;
51:         end if
52:     end if
53: end loop

54: MAIN()
55: create array cells;
56: N = ∅;
57: E = ∅;
58: G = (N, E);
59: for all (query ∈ queries) do
60:     solutionPath = SolveQuery(query.n_init,query.n_goal);
61:     doSomethingWith(solutionPath);
62: end for
```

TABLE I
SIMULATION SCENARIOS

Scenario	Description
Single Query	Execute a single query from c_{init} to c_{goal} in a 2D environment.
Single Query Replanning	Issue repeated queries from different c_{init} to the same c_{goal} in a 2D environment.
Round Trip	Traverse the four corners of a 2D environment.

more efficient for higher-dimensional C-spaces. For post-processing of the solution path a very simple method was used: starting with n_{goal} the post-processor checks whether a shortcut connection is possible to any of the previous nodes. It continues backwards until it reaches n_{init}.

B. Parameter Values

CPRM requires tuning of some of the parameters to achieve the desired behavior. For the following simulations the parameters were chosen as `numCellsPerAxis` $= 8$, `maxNodesPerCell` $= 1000$, `nodeIncrementPerCell` $= 150$, `occupancyThresh` $= 0.95$, `w1` $= 4$, and `w2` $= 1$.

C. Scenarios and Results

Table I shows the scenarios that were simulated. The first scenario consists of a single query $c_{init} = [-18.35, -18.35]^T$, $c_{goal} = [18.35, 18.35]^T$, i.e. getting from the bottom left to the top right corner of C. The roadmap that is built by the CPRM method can be seen in Figure 4. The simulation results averaged over 20 runs are shown in Table II. Apparently, the overhead of maintaining the cell structure has no effect on the algorithm speed: For the given query, CPRM creates roughly a third of the nodes and edges created by PRM and needs roughly a third of the time taken by PRM. The merit of searching on a smaller graph structure becomes clear when looking at the query execution times: Both methods yield solution paths with approximately the same length and the same number of nodes, however, answering the query with CPRM takes about a fifth of the time taken by PRM. This is because CPRM operates on a smaller graph and thus only expands half of the nodes expanded by PRM.

The second scenario contains a single query with constant c_{goal} and changing c_{init}. This is motivated by the necessity for replanning in real-world problems: a robot that starts to traverse the solution path may – after a while – discover that it has ended up in a different location. It then needs to reissue the planning query with an updated c_{init}. For this scenario it is assumed that the PRM has been built beforehand. The repeated single query scenario contains four queries, starting with $c_{init} = [-18.35, -18.35]^T$, $c_{goal} = [18.35, 18.35]^T$, identical to the first scenario. Let the bottom left cell be denoted $(1,1)$ and the top right cell $(8,8)$, then, for the remaining three queries, c_{init} is generated randomly within the cells $(2,2)$, $(3,3)$ and $(4,4)$ respectively. Table III shows the results averaged over 20 runs. Note that for the CPRM
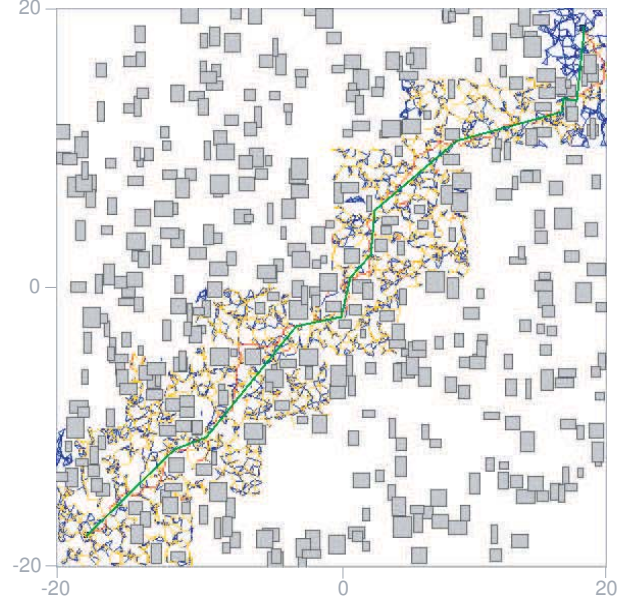

Fig. 4. A single query answered by CPRM in a 2D environment with 400 obstacles. The roadmap is tailored to the query and consists of 3,257 nodes and 7,905 edges.

TABLE II
SINGLE QUERY SCENARIO

Algorithm	n_{nodes}	n_{edges}	n_{comps}	$t_{creation}$
PRM	10,000	24,836	36.85	3,364 ms
CPRM	3,486	8,745	11.90	1,731 ms
	l_{path}	n_{path}	$n_{expanded}$	t_{query}
PRM	55.22	14.40	5,378	874.10 ms
CPRM	57.16	16.95	2,704	159.85 ms

t_{query} now denotes the total time needed to build the roadmap and find the solution path. For the PRM t_{query} denotes only the time needed to find the solution path, since it is assumed that the complete roadmap has been built beforehand. Not surprisingly, the PRM is faster for the first query, where the CRPM needs to build a roadmap from scratch. Remarkably, on the second and third query the CPRM clearly beats the PRM, because it reuses the roadmap built in the first query and searches a smaller number of nodes. On the fourth query PRM takes a slight lead because the remaining graph portion to be searched up to c_{goal} is now small enough.

The third scenario is a benchmark scenario only for CPRM. Four planning queries are issued that traverse the four corners of the C-space. After three queries, the situation depicted in Figure 3 arises. Again all c_{init} and c_{goal} are located at $[\pm18.35, \pm18.35]^T$ and the results are averaged over 20 runs. As can be seen in Table IV, CPRM almost instantly publishes the solution found along the existing roadmap. With a length of 106.31 this solution is highly suboptimal, so CPRM keeps growing the roadmap until after roughly three seconds the final solution is found.

TABLE III

PRM	l_{path}	n_{path}	$n_{expanded}$	t_{query}
Query 1	55.43	14.50	5,338	854.25 ms
Query 2	47.59	13.40	4,081	636.75 ms
Query 3	39.45	12.40	2,867	368.1 ms
Query 4	32.92	10.90	2,166	245.8 ms
CPRM	l_{path}	n_{path}	$n_{expanded}$	t_{query}
Query 1	56.16	17.45	2,419	1,668.00 ms
Query 2	48.20	15.70	2,397	383.00 ms
Query 3	39.48	12.50	1,940	327.40 ms
Query 4	32.01	10.70	1,440	268.40 ms

TABLE IV

ROUND TRIP SCENARIO

CPRM	l_{path}	n_{path}	$n_{expanded}$	t_{query}
Query 1	40.44	10.40	1,391	922.44 ms
Query 2	40.04	13.00	1,469	790.00 ms
Query 3	39.52	10.00	1,578	888.00 ms
Query 4 (subopt.)	106.31	27.00	4,280	226.00 ms
Query 4 (final)	39.31	10.00	2,433	3,245.00 ms

V. EXTENSIONS AND FUTURE WORK

Probabilistic Roadmaps have been studied extensively over last decade. Many of the investigated extensions and improvements can be directly implemented as part of the CPRM algorithm. Different sampling and node adding techniques such as gaussian, halton, obstacle-based, visibility-based etc. – see [15] and [16] for an overview – can all be implemented as part of the GROWPRMINCELL() routine.

In its current form CPRM has a number of parameters which need to be hand-tuned to a specific problem. Future work will aim to automate some of this parameter tuning. Furthermore, a validation of the algorithm in high-dimensional planning problems will be conducted. Another interesting possible extension to the basic CPRM algorithm is a memory-bounded variant that "'forgets'" cells that have not been visited for a defined time.

VI. CONCLUSION

In this paper the Cell-based Probabilistic Roadmap, a novel sampling-based path planning algorithm, has been presented. The method has anytime as well as replanning capabilities and can be used as an incremental multiple-query path planning method. Due to its cell structure and the adaptation of the roadmap to a set of queries, CPRM is especially efficient for path planning in large environments, where regular roadmaps cause a slowdown of graph search algorithms. The efficiency of the CPRM method has been verified by a number of simulations in a basic 2D environment.

ACKNOWLEDGMENTS

REFERENCES

[1] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566–580, 1996.

[2] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning." TR 98-11, Computer Science Dept., Iowa State University, Oct. 1998.

[3] K. E. Bekris, B. Y. Chen, A. M. Ladd, E. Plaku, and L. E. Kavraki, "Multiple query probabilistic roadmap planning using single query planning primitives," in *2003 IEEE/RJS International Conference on Intelligent Robots and Systems (IROS)*, (Las Vegas, NV), pp. 656–661, October 2003.

[4] M. Akinc, K. E. Bekris, B. Y. Chen, A. M. Ladd, E. Plaku, and L. E. Kavraki, "Probabilistic roadmaps of trees for parallel computation of multiple query roadmaps," in *Eleventh International Symposium of Robotics Research (ISRR)* (D. Paolo and R. Chatila, eds.), vol. 15 of *Springer Tracts in Advanced Robotics (STAR)*, pp. 80–89, Siena, Italy: Springer Verlag, 2003.

[5] P. E. Missiuro and N. Roy, "Adapting probabilistic roadmaps to handle uncertain maps," in *Proceedings IEEE Conference on Robotics and Automation (ICRA)*, 2006.

[6] K. Belgith, F. Kabanza, L. Hartman, and R. Nikambou, "Anytime dynamic path-planning with flexible probabilistic roadmaps," in *Proc. of the IEEE International Conference on Robotics and Automation*, pp. 2372 – 2377, 2006.

[7] J. van den Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," in *Proc. of the IEEE International Conference on Robotics and Automation*, May 2006.

[8] M. Likhachev, D. Ferguson, G. Gordon, A. T. Stentz, and S. Thrun, "Anytime dynamic A*: An anytime, replanning algorithm," in *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, June 2005.

[9] R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (San Fransisco, CA), pp. 521–528, IEEE Press, April 2000.

[10] G. Sanchez and J. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," 2001.

[11] A. Ranganathan and S. Koenig, "PDRRTs: Integrating graph-based and cell-based planning," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

[12] F. Lingelbach, "Path planning using probabilistic cell decomposition," in *IEEE International Conference on Robotics and Automation (ICRA), New Orleans, LA, USA*, Apr. 2004.

[13] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006.

[14] D. Nieuwenhuisen and M. Overmars, "Useful cycles in probabilistic roadmap graphs," in *IEEE International Conference on Robotics and Automation*, pp. 446–452, 2004.

[15] R. Geraerts and M. Overmars, "Sampling techniques for probabilistic roadmap planners," 2004.

[16] R. Geraerts and M. Overmars, "A comparative study of probabilistic roadmap planners," *Algorithmic Foundations of Robotics V, Springer Tracts in Advanced Robotics*, vol. 7, pp. 43–57, 2004.