Comparative Study of Probabilistic Cell Decomposition And Probabilistic Roadmap

Fahim Mannan
School of Computer Science
McGill University

Abstract

. This report looks at a new approach to motion planning known as Probabilistic Cell Decomposition (PCD). This approach combines ideas from Approximate Cell Decomposition (ACD) and Sampling-based motion planning to create a planner that can work in high-dimensional static configuration space. This report first gives an overview of PCD and then describes the implementation of the planner along with implementation of Probabilistic Roadmap planner (PRM). In addition to that a basic comparison between the two has been done by running the planners on a set of problems. Finally, some of the implementation issues and possible ways of improving the planner are discussed.

I. Introduction

Over the years several approaches for solving motion planning problems under various conditions have been proposed. But in general all of these can be categorized into either a) Combinatorial approach or b) Sampling based approach [1]. One of the most applied approach in combinatorial motion planning had been cell decomposition based motion planning[2]. But its field of application was restricted by "the curse of dimensionality". Due to the amount of computation required in cell decomposition based approaches, they are not feasible for planning in high dimensional space. In this arena, sampling-based motion planning approaches perform much better. Recently a new method that combines cell decomposition with sampling-based motion planning was proposed[3,4]. This is known as Probabilistic Cell Decomposition (PCD). The objective of this project is to study this new approach by implementing it and comparing it with other sampling-based motion planning approaches. The scope of the project is restricted to comparing PCD and Probabilistic Roadmaps (PRM) for simple environments. The report will mainly focus on introducing PCD, its implementation and its comparison with a standard implementation of probabilistic roadmap. More emphasis is given on the implementation issues that were faced. Also rather than restricting discussion only to comparison, some ideas for possible extensions are given along with their justification.

The report is organized as follows – section II gives an overview of PCD as well as related methods. Section III elaborates on the implementation of PRM and PCD. In section IV experimental results are presented. Section V has discussion on the results, issues faced during implementation, possible shortcomings of PCD and some ideas for improvement. Finally, section VI concludes with possible directions for future work.

In the rest of this report, C_{free} and C_{obs} will denote free space and obstacle space respectively and κ_i will be used to represent the i^{th} cell in approximate or probabilistic cell decomposition.

II. Background

In this section, overview of the related methods are given. The discussion will mainly be restricted to cell decomposition, sampling-based motion planning, and probabilistic roadmap.

A. Cell Decomposition

Cell decomposition is a combinatorial approach to motion planning. The objective of this approach is to divide the robot's free space Cfree into non-overlapping regions which are known as cells. The adjacency relations between the cells are then constructed using a connectivity graph. Motion planning is then reduced to the problem of finding a sequence of cells (known as channel) that connects the cell containing the initial configuration to the cell containing the goal configuration. The final path is then extracted from this channel.

There are basically two classes of cell decomposition methods. They are – a) Exact Cell Decomposition and b) Approximate Cell Decomposition. The main difference between the two is that, in exact cell decomposition the set of cells exactly covers Cfree whereas in approximate cell decomposition the cells just approximately cover Cfree. Besides this approximate cell decomposition uses simpler cells (rectangloids) and therefore decomposing the configuration space into cells and determining adjacency is easier than the exact method. However, there is a tradeoff. The exact cell decomposition can always find a path if it exists. But approximate cell decomposition is resolution complete.

Probabilistic cell decomposition method is mostly related to approximate cell decomposition approach. Therefore, a more thorough overview is going to be provided in the following. In approximate cell decomposition a cell is categorized as – a) Empty – if and only if its interior does not intersect the C-obstacle region, b) Full – if and only if the cell is entirely contained in C-obstacle region and c) Mixed – when none of the above conditions are satisfied. The connectivity graph in approximate cell decomposition contains nodes that represent the mixed and free cells and edge between two nodes specify adjacency between cells. After the cells have been classified the objective is to find a sequence of cells - either mixed or free – from the cell containing initial configuration to the cell containing goal configuration. If all the cells in a channel are free cells then it is an E-Channel otherwise it is called an M-channel. Path planning in this setting is usually done in a hierarchical way. A simple approach is described below –
1. Compute a rectangloid decomposition
2. Search the connectivity graph to find a sequence of cells connecting initial cell to the goal cell. If an E-channel was found then return success.
3. Otherwise for every mixed cell in an M-channel, compute a rectangloid decomposition and go to step 2.

One major problem with this approach is that it requires building the configuration space which is computationally expensive. Also one common approach used for decomposing a cell is 2m-tree decomposition, in which a cell is divided up into 2m subcells. As a result, the number of cells grows very rapidly. Therefore this approach is only applicable for low dimensional (≤ 4) motion planning problems.

B. Sampling-based Motion Planning and PRM

Combinatorial motion planning approaches try to build an exact representation of the configuration space. Sampling-based motion planning avoids that by generating samples and using collision detector to determine whether a sample is in Cfree or Cobs. Since sampling-based motion planners do not have to build the complete configuration space they are the method of choice for high dimensional motion planning problems. In sampling-based motion planning the general approach is to generate samples and check whether the samples are in Cfree using a collision detector. The samples that are in Cobs are discarded. Finally, the planner tries to build a graph using the configurations that are in free space. Solution to the motion planning problem is then the path from the initial configuration to the goal configuration in this graph. Sampling-based motion planning approaches mainly have two categories – a) Single query and b) Multiple-query.

Probabilistic Roadmap is a multi-query path planner where a roadmap is first built by sampling the configuration space in the learning phase and then in the query stage the planner finds a path from the initial configuration to the goal configuration. There are several variations of PRM. Some focus on achieving speed by delaying collision checking (eg. Lazy-PRM [5]) while some other focuses on finding path through narrow passages (eg. Obstacle-based PRM [6]). But the basic idea of sampling, collision checking and building a roadmap is the same.

There are also other variations of sampling-based motion planners. For instance expansive space planners like Rapidly Exploring Random Trees [7]. They are basically used mostly for kinodynamic motion planning. But in this report, the concentration will be mainly on PRM type planners.

C. Probabilistic Cell Decomposition (PCD)

A novel approach of combining sampling-based motion planning and cell decomposition method was proposed by Frank Lingelbach [3,4]. This is known as Probabilistic Cell Decomposition. Like cell decomposition its objective is to decompose the configuration space into cells. However unlike cell decomposition (and more like sampling-based motion planning approaches) it tries to do so without building the configuration space explicitly. This allows it to be applicable for high-dimensional motion planning algorithms.

The cells in PCD are axis-aligned rectangloids similar to approximate cell decomposition method. Unlike approximate cell decomposition where a cell is classified as – Full, Empty, or Mixed – in PCD a cell is classified as – 1) Possibly Occupied $P(\kappa_i \subset C_{obs}) > 0$, 2) Possibly Free $P(\kappa_i \subset C_{free}) > 0$ or 3) Known to be Mixed $\kappa_i \not\subset C_{free} \wedge \kappa_i \not\subset C_{obs}$. This is because PCD does not have complete representation of the C-space to classify a cell with complete certainty. As the algorithm progresses these labels are modified and mixed cells are split to form possibly occupied or free cells. The connectivity graph G, keeps the possibly free cells as its node. There is an edge between two nodes if the corresponding free cells are adjacent. The basic PCD algorithm [3,4] is as follows –

```
while ( !success )
    if ( path ← findCellPath(G) )
        if( checkPath(path))
            success ← true
        else
            splitMixedCells
    else
        q ← randomState(pOccCells)
        if ( !collision(q))
            splitMixedCells
```

In every iteration of the algorithm, the graph G is searched for a path using ‘findCellPath’. The function performs A* search for finding a sequence of cells connecting the initial cell to the goal cell. If such a sequence is found then those cells are checked for collision free path using ‘checkPath’. The ‘checkPath’ function implements local path planning which basically finds a path between adjacent cells. This path is a straight line connecting the center of the shared area which is an (n-1)dimensional hyperplane. Therefore if κ_{i-1}, κ_i and κ_{i+1} are adjacent then the path connecting κ_{i-1} and κ_{i+1} through κ_i would be a straight line connecting the center of the shared area between κ_{i-1}, κ_i to the center of the shared area between κ_i, κ_{i+1}. This straight line path has to be in C_{free}. This path is also checked for collision either using an incremental approach or binary collision checking. If some colliding configuration is found during this step then it is known that the cell in which it was found is a mixed cell. Therefore the cell is split up into possibly free and possibly occupied cells. If a path was not found by ‘findCellPath’ then the possibly occupied cells are sampled and if any one of them has a free configuration in it, the cell is split into possibly free and possibly occupied cells as before. Several strategies can be taken in this step to determine the possibly occupied cell to sample. One sample can be generated for all the possibly occupied cells, a number of samples can be distributed over the accumulated volume of the possibly occupied cells, or even sample a subset of the cells that are more important.

Due to the shape of the cells (axis-aligned rectangloid) splitting and adjacency checking is relatively easy in n-dimensional space. A cell is split by taking the nearest existing sample and cut orthogonally in the middle in the dimension of the largest distance. This ensures enough separation between the configuration in the cell and its boundary. If a cell contains m+1 samples where m samples are similar and the other is different then for an n-dimensional problem the number of children would be at most min(2m, n) + 1. This number is significantly less than the 2^m-tree decomposition approach.

III. Implementation

For this project, both the Probabilistic Cell Decomposition and Probabilistic Roadmap planners were implemented. A framework was created for implementing planning algorithms and testing them in different environments with different queries. The implementation was done in C++. Samples were generated using random number generators that are available in the Boost library. Collision detection was done using the Proximity Query Package (PQP). The details of the framework are given in Appendix A. In this section, the algorithmic details of both Probabilistic Cell Decomposition and Probabilistic Roadmap are presented.

The most basic PRM was implemented. In each iteration it generates a number of samples and checks whether each of these samples are in free space using the aforementioned collision detector. Only the samples that are in C_{free} are kept and the rest are discarded. After that K-nearest neighbor algorithm is used for finding the nearest nodes of each vertex. For each pair of such nodes a binary path checker is used to see if the straight line path connecting the nodes is in C_{free}. The binary path collision checker uses an approach similar to binary search to find whether any configuration on the path is in C_{obs}. If the path is collision free then an edge between the nodes is added to the graph. Finally, a graph search is done to see if there is a path between the initial configuration and the goal configuration. If the graph search does not return any path then the planner does the same thing in next iteration and this continues until it has tried for a certain number of times.

The PCD algorithm was implemented following the description in section II. First there is only one cell containing the initial and goal configuration. A simple straight line path is constructed between these two configurations and the path is checked using the binary path checker that was mentioned previously. If the path checker finds a colliding configuration then that configuration is returned to the main PCD algorithm which then uses that to split up the initial possibly free cell. When there are more than one cells then the A* search algorithm is used to find the shortest path between the initial cell and the goal cell. The implementation uses Dijkstra’s algorithm for this purpose. The heuristic cost is simply the sum of current cost (i.e. cost to reach the cell) and the cost-to-go approximated using the Euclidean distance.

If the A* algorithm fails to find a path then the algorithm needs to sample the possibly occupied cells. For faster performance the implementation keeps track of the current possibly occupied cells in a list which is updated whenever a cell’s label changes to or from possibly occupied. The implementation traverses this list of possibly occupied cells and generates one sample in each of these cells. One point to be noted here is that if the cell is too small i.e. all the dimensions have almost collapsed to a point, then no sample is generated in that cell. Finally, for all these cell and sample pair the implementation checks whether any of the samples are in free space. If there is such a case then the corresponding cell is split up into possibly occupied and possibly free cells. Otherwise the generated samples are added to the cell. Mixed cells are not explicitly represented in the implementation. Whenever a cell is known to be mixed it is splitted up into possibly free and occupied cells.

A cell is splitted up by first finding the nearest neighbor of the configuration that is used to split the cell. Then the maximum dimension is determined. The corresponding maximum vertex of the current cell is then set to half way between the two configurations along that dimension. A new cell is also created with the corresponding dimension of the minimum vertex set appropriately. Finally, all the configurations that were in the previous cell are checked to see whether they are in the new cell and updates are made accordingly. This process stops when the colliding configuration/free configuration is isolated in a cell. During the above process the cells status is also updated (ie. possibly free and possibly occupied).

The adjacency relation between cells is also updated during the cell splitting process. An adjacency list data structure is maintained. Whenever a new cell is created its adjacent cells are determined and the adjacency list is updated. Two cells are adjacent if and only if exactly one of their dimensions is equal and they have a common shared area. Shared area is determined by checking whether the region formed by the minimum and the maximum vertices on the (n-1)D hyperplane of both cells overlap.

IV. Experimental Results

Experiments were carried out for three environments with 2DoF and 3DoF robots. The setups are shown in figure 1. The environment in 1(c) has a narrow passage in it. None of the planners were able to solve the motion planning for this case.

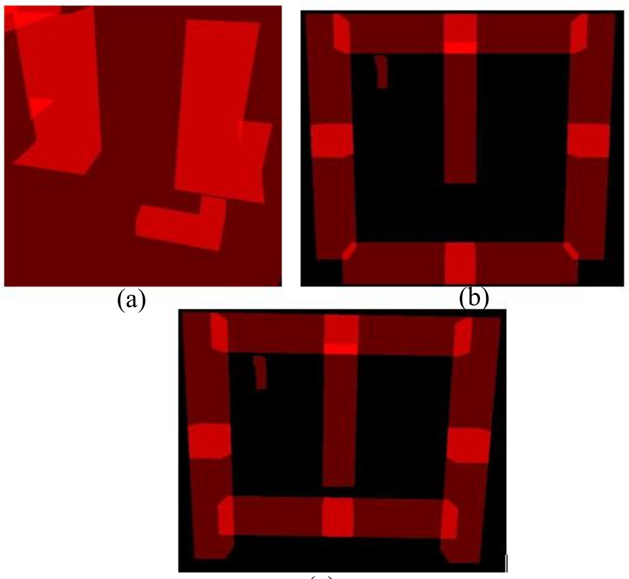

Figure 1:Workspace with (a) pathway in the middle (b)pathway on one end, and (c) narrow passage

Some sample solutions that were found by both planners are illustrated in the following figures. Figures 2(a)-(d) are for the first environment with a 2DoF robot. Both planners were able to solve this problem fairly easily. On an average 20 nodes were generated by the PRM planner and 13 cells were generated by PCD.

Figures 2 (e) and (f) show plans for the same environment but with 3DoF robot. The average number of nodes and cells that were generated were almost the same. The final plan tended to be much shorter because of allowing rotation.

The final two solutions shown in Figures 2(g) and (h) are for the simple maze shown in 1(b). The planners took more time in this case on average. PCD sometimes performed cell division very efficiently and came up with very good solutions.

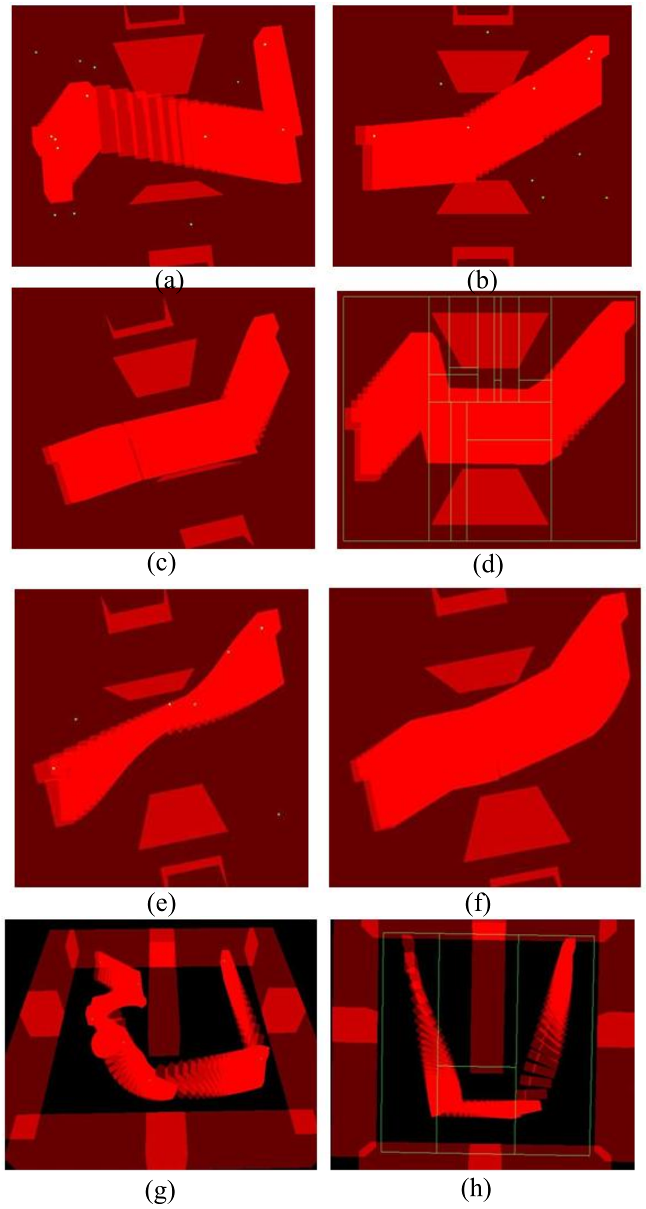

Figure 2: (a) and (b) PRM planner for 2DoF robot (c) and (d) PCD planner for 2DoF robot (e) PRM for 3DoF robot (f) PCD for 3DoF robot (g) PRM for 3DoF robot in a simple maze (h) PCD for 3DoF robot in a simple maze

V. Discussion

From the previous section it can be seen that both planners worked well for the first two environments and allowed degree-of-freedom. However, for the narrow passage problem both failed. This is not completely unexpected since none of the planners aim to solve narrow passage problems. From implementation point of view PRM was easier to implement. The most problematic part with PCD was cell splitting. The difficulty mainly arises when the samples are on the boundary. This may occur even when checking whether a path is free. Because the center of the shared (n-1)D hyperplane may intersect with C_{obs}. In [3,4], the author mentions that the samples that do not lie on the boundary are stored. This implies that if a sample is found on the boundary then it is discarded and in case if it is in collision state and on the boundary of a possibly free cell then that possibly free cell is not splitted. But in this case the algorithm would fail because only splitting the cell would allow it to find a new path. An example of this is shown in Figure 4.

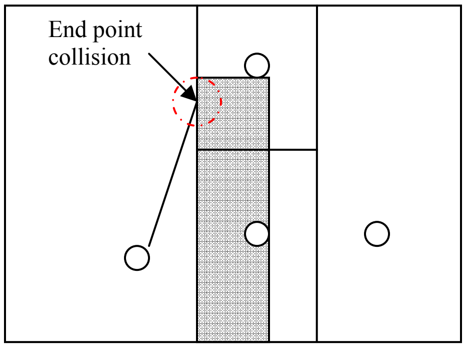

Figure 4. Center of shared region in C_{obs}

In the current implementation, configurations on the boundaries were considered for splitting the cell. Also it is not clearly specified whether samples will be generated for any possibly occupied cells. Because if a cell becomes too small, then generating samples and splitting that cell is not very useful. In the current implementation samples are generated only when the size of a cell is above a certain threshold value.

In the following some possible ideas for improvement are discussed. As was mentioned before the path checking step simply checks whether a path connecting the center of the shared region between adjacent cells is free. There is no explicitly consideration of the case when the center of the shared area might be in colliding state. One approach that can be taken here is to search for a state on the shared boundary that is in C_{free} and consider that for the straight line path. Also the path within a cell might not be a simple straight line path. And considering a straight line path might increase the number of cells. For the case shown in Figure 5, possibly a better approach would be to take a strategy similar to Lazy-PRM [5] where a node enhancement step is performed by generating samples around the collision point to find a collision free path.

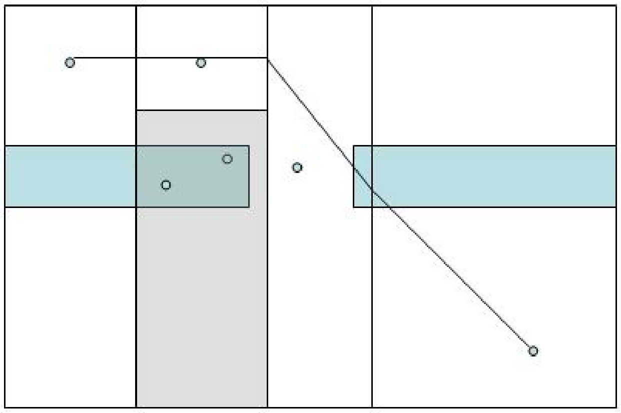

Figure 5. Case for which local path planner needs to be improved

VI. Conclusion and Future Work

This report describes the implementation of a novel motion planning approach known as Probabilistic Cell Decomposition and compares it with basic Probabilistic Roadmap planner. The approach is interesting because it fuses classical cell decomposition with more recent sampling-based approach. During the implementation of PCD a number of implementation specific issues came up. These issues and some scenarios where PCD might benefit by using a different approach were also discussed. For simple problems the performance of PCD and PRM were almost similar. However in some cases PCD seemed to perform better in terms of finding optimal solution to a problem.

Current work does not do any comprehensive comparison of the two planners or comparison of PCD with other planners. Future work will focus on improving the current implementation of the planners and using more complex benchmarks for evaluation. It would also be interesting to see how the performance of PCD is affected by incorporating the ideas for improvement.

Appendix A. Motion Planning Framework

In this appendix, a brief overview of the framework that was implemented for evaluating motion planning algorithms is given. It is a minimal framework that allows implementing probabilistic planners like PRM and PCD. The framework allows loading environment files, models and motion planning queries. The basic architecture of the framework is shown below in UML.

The Environment class manages all the models in an environment including the robot. It also interacts directly

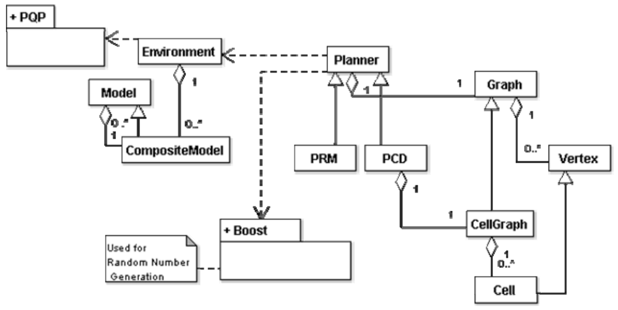

Figure A.1: UML Diagram of Motion Planning Framework

with PQP to determine obstacle collision. The planner also uses it to access information like degree-of-freedom and testing samples.

To use the framework for implementing a new planner, users first need to inherit their planner from the planner class. The planner class will also give access to the Graph library. If a specialized graph data structure is required (eg. CellGraph), then the Graph class can be extended for that purpose.

For sampling, the current planners use the random number generators provided by the Boost library. For this particular project uniform sampling was done using the Mersenne Twister implementation provided by the Boost library.

In addition to these, the planner also provides support for visualizing and tracing plans generated by a planner. Future work on the planner will mainly be on integrating CGAL for using dD Spatial Searching and other computational geometric algorithms for motion planning.

References

[1] S. M. Lavalle, *Planning Algorithms*. Cambridge University Press. 2006.
[2] J.-C. Latombe, *Robot Motion Planning*. Kluwer Academic Publishers, 1991.
[3] F. Lingelbach, "Path Planning using Probabilistic Cell Decomposition," Licentiate Thesis, Royal Institute of Technology (KTH), February 2005.
[4] F. Lingelbach, "Path Planning using Probabilistic Cell Decomposition,", *In Proceedings of International Conference on Robotics and Automation,* 2004.
[5] R.Bohlin and L. Kavraki, "Path planning using lazy PRM," *In Proceedings of International Conference on Robotics and Automation,* 2000.
[6] N. Amato, O. Bayazit, L.Dale, C.Jones, and D. Vallejo, "OBPRM: An obstacle-based PRM for 3D workspaces," *In Proc. International Workshop on Algorithmic Foundations of Robotics (WAFR),* 1998.
[7] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Computer Science Dept., Iowa State University, Tech. Rep. 98-11, 1998.