

# Roadmap Based Partitioning

Jory Denny, Phillip Coleman, Lydia Tapia, and Nancy M. Amato  
Parasol Lab, Dept. of Computer Science & Engineering,  
Texas A&M University, College Station, TX 77843  
{jdenny, pcoleman, ltapia, amato}@cse.tamu.edu

August 6, 2009

In the field of robotics, one of the most challenging aspects is the perception and mapping of the environment. The robots need a good way of categorizing and organizing the regions of the environment to plan their motion around the environment. However, dividing the environment into regions is a challenging problem. These regions can be used for different purposes whether it be multi-agent systems, adaptive motion planning, or parallel graph algorithms. Roadmap Partitioning might be defined as simply splitting the roadmap into different regions for use of another algorithm. However, splitting the environment in an intelligent manner is difficult. We show how a method that we have created is better than the basic K-means Clustering Algorithm by which the user specifies the number of regions, rather than finding the optimal number of regions automatically. There are ways to find the optimal number of regions for K-means Clustering but they have weaknesses. In this paper we show the need for partitions, the uses of partitions, and a comparative analysis between two partitioning methods. Once these partitions are made from a roadmap, the motion planning of each partition can possibly be used as previously stated.

## 1 Introduction

It is common in robotics to deal with very complex robots and environments. These environments range from small to large, and may contain any combination of obstacles placed in any way around the space. In order to simplify the motion planning problem one may consider splitting the environment into subsections. This is roadmap partitioning, the splitting of the environment based upon the roadmap that conceptually lies within it. This problem has proven to be difficult because of the wide range of environments that are out there, and the subjective nature of a “good” region. A partition is some fraction of the environment, and can be considered a sub-environment that might even need to be sub-partitioned at a later time.

Partitioning the roadmap has a variety of uses. In motion planning, partitioning can be used to simplify a complex problem by splitting it into sub-problems, or for identifying regions to better sample the environment to create a more complete roadmap for the motion planning problem. Also in the area of motion planning, multi-agent systems can use partitions to bind agents to a particular region, or for creating complex interaction between the agents. An example of this might be a system trying to recreate the natural behavior of animals. By binding the agents to specific partitions, the system can effectively simulate the territorial nature of animals. In both of these cases the partitioning could be extended to creating an effective way of parallelizing motion planning. In the first case of region identification, if the partitions were created each processor could handle a different partition. In the later case of multi-agent systems, each processor could handle the planning for the agents in a specific partition which would allow for larger and more complex simulations.

Our general approach to this problem is to define a partition, create different partitioning methods, and save a hierarchy of partitions for later use by other algorithms. In this paper we compare two specific methods which both have their advantages. K-means Clustering is fast, but the user has to specify the number of partitions needed for the environment in the basic algorithm. Growable Partitions, though not optimized for this paper, is slow but generates the ideal number of partitions for an environment. Both methods have their benefits and disadvantages.

In the rest of the paper, we discuss the related work in motion planning and partitioning. Then we discuss our general framework of partitions, partitioning methods, and the partitioning hierarchy. After this, we cover an in-depth analysis of the two methods to be tested, K-means and Growable Partitions. After the analysis of our approach, we present our results then conclude the paper.

### 1.1 Problem Definition

Below defines certain terms used throughout the paper.

**Environment:** The environment is a bounding volume which contains obstacles.

**Roadmap:** The Roadmap is a graph data structure where the nodes are valid configurations of a robot and the edges are valid paths between the nodes. The roadmap is a representation of the environment.

**Partition:** A Partition is a sub-volume of the environment.

## 2 Related Work

### 2.1 PRMs

In motion planning, much work has been done to solve the general problem of moving from a start configuration to an end configuration given an environment. There are multiple methods to solve this problem, ranging from calculating the C-space[10] (an improbable solution for degrees of freedom of more than 3[3]), to probabilistic methods. We use a Probabilistic Roadmap Method (PRM)[6] in order to create a roadmap around the environment, and then partition the environment from the roadmap. After this is created we

partition this roadmap. The PRMs have been previously proven to be probabilistically complete, and are a good general solution to the motion planning problem for high degrees of freedom (more than 3). The Probabilistic Roadmap Method consists of two phases the construction phase, and the query phase.

**Roadmap Construction.**[6] The construction phase consists of two parts node sampling and node connection. There are many projects that have found and gone over sampling and connection of nodes, here we present the basic sampling and connection methods for PRMs. Sampling is simplest when dealing with a uniform distribution over the environment. One such method of connection is to connect the  $k$  closest nodes.

- **Node Sampling:** The node sampling phase starts with selecting random possible configurations of the robot. This can be done with using a uniform distribution, Gaussian distribution, etc. Then every configuration is tested for validity. Validity test is collision detection in all cases. Two tests will be computed for each configuration, a self collision test of the robot, and an obstacle collision test in the environment. The valid configurations are saved, and the invalid ones are removed in the basic PRM.
- **Connection:** During the connection phase, every node is connected to its  $k$  closest neighbors. The connection between these configurations is tested with a local motion planner, most commonly a straight line local planner. In this type of local planner configurations along this line up to a resolution  $D$  are tested for validity, if all the configurations along the line are valid then the knowledge of the connection is saved. After the  $K$  closest connections are calculated, the connected components of the graph are connected using a similar method.

After both of these phases are completed, a roadmap is constructed which has been proven to be probabilistically complete.

**Query Phase.**[6] During the Query Phase of the PRM, multiple queries of paths from start configurations to goal configurations are calculated. This is calculated using Dijkstra’s algorithm, or some other common method to find the shortest path. However, before these paths are calculated, the start and goal configurations of a query are connected to the roadmap.

## 2.2 Uses of Partitioning.

As previously stated, partitioning has a wide range of uses, whether it is in parallel motion planning, or splitting a hard motion planning problem into sub-problems. In this section we describe two previous works using partitioning, Feature Sensitive Motion Planning[11], and the Unsupervised Adaptive Strategy for motion planning[13].

**Feature Sensitive Motion Planning.** Since there are many methods of solving motion planning problems, each with their own advantages, Feature Sensitive Motion Planning tries to combine these to get the best of all worlds. In this method regions are produced using some partitioning method, then these regions are categorized for specific sampling. For example, an environment might contain one free region (without any obstacles) and one cluttered region (with a lot of obstacles). Once these regions are collected different PRMs are selected to match the specific problem. For the previous example, a basic PRM would be used on the free region, and a MAPRM (Medial-axis PRM)[8] or OBPRM (Obstacle Based PRM)[1] for the cluttered region. This work showed an advanced way of solving a wider range of problems than other specified planners could solve. This method also creates a more accurate roadmap which is more representative of the environment.

**Unsupervised Adaptive Strategies.** The Unsupervised Adaptive Strategy(UAS) is similar to the feature sensitive motion planning in the sense that it identifies regions and specifies the planner to the region. In UAS, K-means, a common partitioning method where the user selects the number of partitions for the environment, is used for partitioning. The user selects features for the K-means method to use. These might include X position, Y position, Visibility Ratio, etc. This method showed an improvement in speed

and quality in the roadmaps generated when compared to the feature sensitive motion planner and other specific planners.

**Workspace Adaptation Methods.** Many methods have been proposed to explore the impact of adaptation in response to the features of the planning workspace. A recent adaptation of the hybrid PRM method [7] uses workspace information, extracted from a cell decomposition, to define locations where samplers should be applied. Another workspace-based approach applies the watershed method (previously applied in image processing) to identify narrow passageways in the workspace[2]. After such features are identified, the planning can be adapted based on the characterization of the region. While these approaches rely heavily on spatial characteristics in order to decide where to apply a planner, they do not consider the change in the topology that is discovered as a space is explored. Their performance also degrades in more complex problems (different C-space types) where difficult (e.g., narrow) regions of C-space can no longer be identified from difficult regions of the workspace (such as with articulated linkages and other constrained robots).

## 2.3 Previous Partitioning Methods

Partitioning does not have to be map based, it does not even have to be deterministic, but these two things would be important for creating an effective method of identifying regions. Some previous methods of partitioning that are not roadmap based, which requires the partitioning method to know the obstacles, are Gap Partitioning, Random Partitioning, and Entropy Based Partitioning[12].

**Gap Partitioning.** Gap Partitioning uses the knowledge of the free sampled nodes in order to create the partitions. It specifically targets large gaps between the free nodes. This method is deterministic but bases its partitions on the obstacles, within which a node sample cannot be kept. The partitions are placed in areas of large gaps between obstacles. The partitions become based upon obstacles, which is good only in certain environments, and certain problems.

**Random Partitioning.** Random Partitioning is a elementary subdividing strategy by which a random point is used to split the data based upon one random DOF. this process can be repeated on each node if necessary. This method has the weakness of not being deterministic, and does not guarantee any quality of partitions based upon features.

**Entropy Based Partitioning.** Entropy Based Partitioning uses the concept of entropy to partition an environment. Entropy is the amount of disorder in a region. It partitions in order to lose entropy. The end result will be two or more partitions of lowest entropy based on a given feature from the environment.

**Clustering.** Clustering Techniques have been widely studied. In clustering, the goal is to find clusters, or dense regions of nodes, in a graph or other data set. By finding these clusters, they should accurately represent a partition. K-means Clustering[5] is a popular clustering technique, which finds centroids and assigns the nodes to their closest centroid. Then the algorithm averages the nodes to find a new centroid. When no change has occurred in the reassignment phase the clusters are found. In the case of the basic algorithm, the user must specify the number of clusters for K-means. In that sense the basic clustering techniques have a disadvantage.

## 3 Partition Representation

### 3.1 Partition

A Partition is a sub-region of the environment. This can be seen as a volume of the planning space. The actual information that is kept inside of a partition is as follows:

- A reference to the roadmap

- A vector of node indexes of the roadmap that are part of the partition
- A bounding volume

The roadmap that is referenced is the roadmap created by a basic PRM method, that is used as information for the partitioning method to partition the environment. The vector of node indexes or, VIDs, represent the nodes that are part of the partition. The bounding volume is an approximate representation of the partition and is calculated from the VIDs or can be used for a non roadmap based partitioning method. In Figure 1 an example of a partition is shown, this is a partition in a multi-agent system in which agents are bound to their partition.



Figure 1: Example of partitions in multi-agent systems

### 3.2 Partition Hierarchy

**Representation.** The partition hierarchy is represented as a tree data structure in which the root node is the entire environment and the leaf nodes are the lowest level of partitions. The leaf nodes are the only nodes which store information. These leaf nodes actually consist of a partition as defined above. The internal nodes can recreate their partition from the information from their children. In Figure 2 a picture of the partition hierarchy is shown. This picture shows what and where information is stored and represented.

**Construction.** The tree is created by replacing a leaf node with an internal node whose children are the new partitions that are created from a partitioning method. The new leaf nodes represent the new sub-regions, and the new internal node is the old region. The maximum number of partitions would be  $N$ , the number of Nodes. In which each of the  $N$  partitions will contain one node. This tree structure controls all access to the partitions and creating new partitions.

**Possible Uses.** The partition tree can be used for parallelizing the motion planning through an environment. Depending on the number of partitions and processors one could use the internal nodes to assign partitions to a processor. For example if a partition tree has a root node with two internal nodes as children whom each have two additional children, and there are two processors available, then each processor will be assigned one internal node to control.

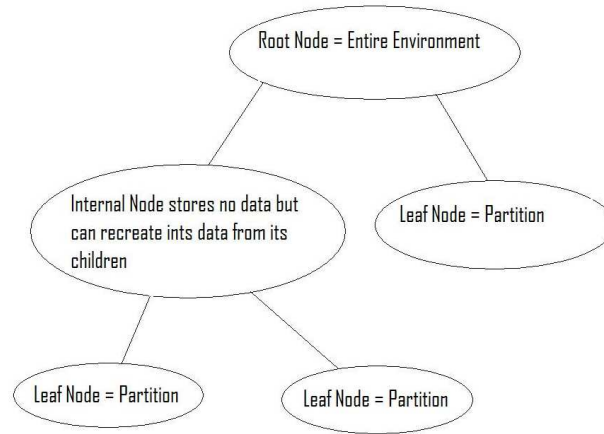


Figure 2: The partition hierarchy

## 4 Partitioning Methods

A partitioning method is completely modular to the framework. Different implementations can be swapped very easily, all that is needed is to define how to split up a partition. Basically, the method takes a partition as input then outputs a set of partitions. The partition tree controls this and creates the nodes (leaf, internal, and root) for the tree from the partitions.

---

### Algorithm 4.1 General Partitioning Method

---

*Input.* A Partition

- 1: Create a vector of Partitions
  - 2: **return** Vector of Partitions
- 

### 4.1 K-means Clustering

The basic algorithm of K-means clustering is shown in the algorithm below. Basically the algorithm takes an integer value  $k$ , and features of the environment, such as X position, visibility, or rotation. Then it starts by randomly selecting  $k$  centroids within the bounding range of each of the features. After the initial centroids are created, each node is assigned to its closest centroid. After this assignment of nodes has occurred, the average of the clusters are determined. These values become the new centroids. The reassigning phase continues until either  $n$  number of iteration have passed or no reassignment occurs in an iteration. The algorithm gives  $k$  clusters. Each cluster is then converted to a partition, and  $k$  partitions are created from this method. The advantage of this method is speed, but the user must provide the number of clusters wanted. The optimal number can be determined through analysis of the Elbow Criterion[4, 9]. In the elbow criterion, a plot of the average variance of the data vs. the varying  $k$  value is found. Then the second derivative of this plot is found. The optimal number of clusters is the  $k$  which holds the absolute maximum value on this second derivative plot. However, this takes time and the rerunning of clustering many times on the same data set.

---

**Algorithm 4.2** K-means Clustering

---

- 1: Randomly choose  $k$  random points
  - 2: Assign every node to its closest point
  - 3: Average the values of the nodes in each cluster to get a new center point
  - 4: Repeat steps 2 and 3 until convergence is reached
- 

## 4.2 Growable Partitions

The concept of Growable Partitions is the method starts with  $N$  partitions, one for each node of the roadmap. Then merges the  $N$  partitions into  $M$  final partitions. This method is outlined in the algorithm below. To start,  $N$  partitions are created, these are called the free partitions. After this the partitions are merged together if the new partition scores are better then each of the separate partitions. The merging starts with the 1st partition in the free partitions. The closest partition in the free partitions is selected and then the scores are calculated to determine if these partitions should be merged, if so then the partitions are removed from the free partitions and the new merged one is added to the free partitions. The scoring method has one part currently, but are modular and can be switched out. For this paper we use the visibility of the partitions. Visibility is calculated by determining the number of valid connections between all pairs of nodes in the partition. This divided by the total number of connections possible is how the percentage of visibility is determined. The method continues if a merge has happened, if no merge has happened then the optimal number of partitions is determined. The advantage of this method is that the optimal number of partitions is found, the only downfall is that the method is slow with an upper bound of  $O(N^4)$ .

---

**Algorithm 4.3** Growable Partitioning

---

- 1: Start with  $N$  partitions, one for each node
  - 2: Until no merging occurs
  - 3: Compare each partition to its closest neighboring partition
  - 4: If the merging of these two partitions is greater then the separate partitions
  - 5: Merge the two partitions
  - 6: Else do not merge and continue
  - 7: If a merge occurs repeat
- 

## 5 Experimental Results

### 5.1 Experimental Setup

**System Specifications.** The experiments that we performed where run on a linux machine running Cent OS 5.1, which has a Pentium 4, 3 GHz, multi-core processor. We ran tests over two environments and captured the speed of the partitioning method and the average visibility per partition. This will be discussed in more detail in a later section. We used a roadmap created by a Basic PRM. We ran the partitioning methods on the same maps to compare the results.

**Environments.**

- Maze: The maze environment, shown in Figure 3, shows three distinct partitions. One at the top, one at the bottom, and one for the middle obstacle. The challenge is detecting the middle region. In basic PRM the chance that nodes land there are slim, so in reality when using a basic PRM to decide partitions a smart partitioning method should detect two regions of high visibility, top and bottom.

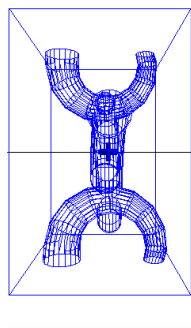


Figure 3: The Maze Environment

- Random-Box: The random-box environment, shown in Figure 4, is a challenge to detect the correct number of partitions. The elbow criterion for K-means shows that there should be three partitions for this environment.

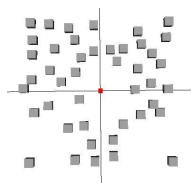


Figure 4: The Random Box Environment

**Quantified Analysis for the Quality of Partitions.** In order to quantify the quality of the partitions we took a visibility calculation. To calculate this quantity within a partition we calculated the total number of valid connection between nodes divided by the total possible connections between the nodes. This is not the only method of evaluation but for our purpose it was good enough. The scoring mechanism is interchangeable, and we could have just as easily decided to test it with expansion ratio, or some other quantity.

## 5.2 Results

### 5.2.1 Maze Environment

In the first half of the experiment, we ran the experiments on the Maze Environment. In this test we were trying to see if the partitioning methods could find the correct number of highly visible areas using a basic PRM method. The elbow criterion determined the optimal number of clusters was three for the Maze Environment. A plot of the average visibility over all partitions vs the number of nodes used in the PRM sampling, and a plot of the time took to partition the environment are shown in figure 5.

In the first plot the results of quality are shown. Generally the Growable Partitioning Method successfully identified the two areas in all but the run with 300 nodes. In the run of 200 nodes the K-means method found two pockets of high visibility and one with low visibility which explains the dip in the average visibility for this method. The cause of this was that there were more nodes then expected in the middle range of the environment. This high number causes the K-means to identify the middle region as a cluster. Generally

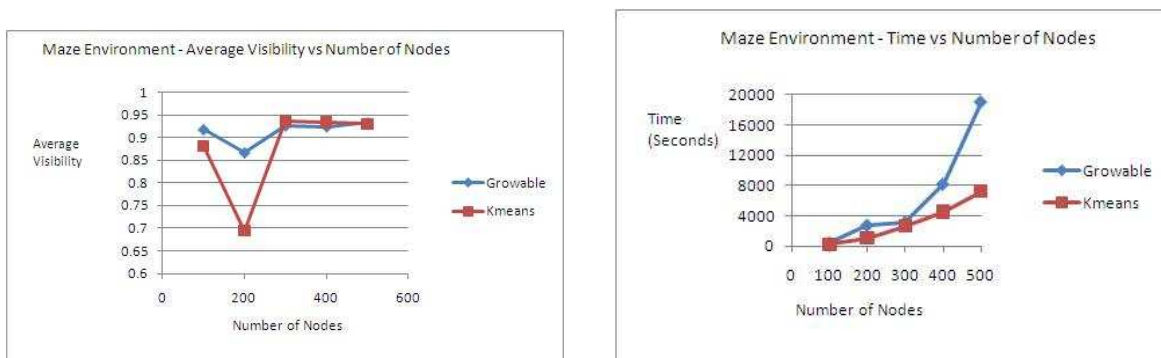


Figure 5: Comparison of Methods on the Maze Environment

speaking the K-means failed to detect the two areas, giving partitions as shown in Figure 6. There are two at the top and one at the bottom. However, in the growable partitioning method, the partitions were correctly determined to be one at the top and one at the bottom as seen in Figure 7.

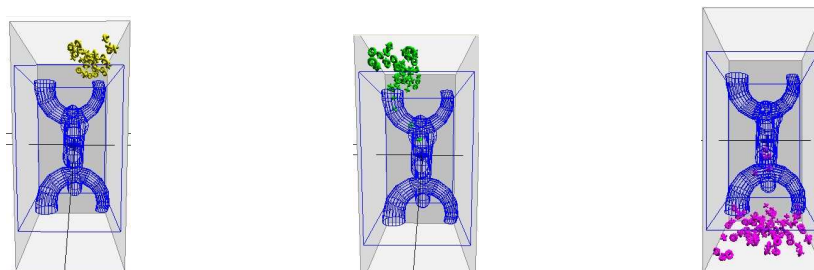


Figure 6: K-means Clustering on Maze Environment

The second plot shows the time analysis comparison between the two methods. Theoretically K-means is a faster optimized algorithm bound by  $O(N^2)$  and growable partitioning is an unoptimized method that is bound by  $O(N^4)$ . The  $N$  is the number of nodes in the PRM roadmap. This plot generally follows this scheme except for the times on the run of three hundred nodes where the plots touch. This was the one run in which the growable method determined three partitions instead of two.

### 5.2.2 Random-Box Environment

In the second part of the experiment we were testing to see if the method could find pockets of high visibility. For the random-box environment there were many pockets of high visibility which theoretically K-means should not be able to detect because the user specifies the number of partitions. For this environment, because the K-means was required to have a  $k$  value of three based upon the elbow criterion, the visibility of these clusters greatly decreased. Whereas in the growable method many small regions of high visibility were found. This is shown in the first plot of Figure 8.

In the second plot the running time comparison between the methods is shown. The time in seconds is plotted against the number of nodes. In the previous section, K-means was shown to be faster, but in this case K-means ran slower in every case. In this experiment a more complex relationship between the number of nodes, number of partitions and time was found for the time analysis of growable partitioning.



Figure 7: Growable Partitioning on Maze Environment

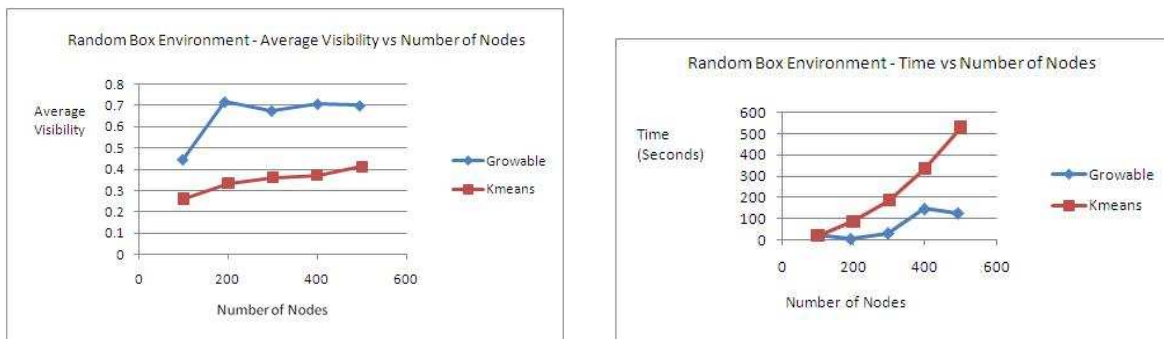


Figure 8: Comparison of Methods on the Random-Box Environment

The K-means time complexity is predictable and is bound by the theoretical approximate previously stated. The growable partitioning method varies greatly. This is explained by the number of partitions formed. The number of partitions was always greater than 8 in the case of Growable Partitioning meaning that the algorithm did not run to its full length but rather stopped at the appropriate number of regions. Note that the goal of the experiments was to find highly visible regions which is why the random box yielded many partitions. But if the accurate number of partitions is very small then the growable method is going to take longer than K-means. Note that the reason each of these methods took hours to complete was that each method required visibility calculations. To calculate this we found the number of valid connections between the nodes. This is a hefty calculation and required many collision detection calls. The time analysis for the growable partitioning method shows some areas that need improvement and others that are good.

Our results show that in general to solve this initial problem of finding high visibility areas the growable partitioning is better, but it also shows that our method needs many improvements including making the algorithm more consistent, speeding up the algorithm, and allowing for mid-range visibility and low visibility. Our results are the first step and there is much left to do, but the growable partitioning method is very promising. The results show that there is a general trade-off between speed and quality of the partitions. Generally, K-means is a faster algorithm but requires the user to provide the number of partitions; and Growable Partitioning takes a heavier calculation but the user does not have to specify the number of partitions.

## 6 Conclusion

In conclusion, we discuss the faults of the previous methods of partitioning, we discussed our framework, and we discussed our results. In our method we define partitioning, explain the partitioning hierarchy, and describe the general algorithm for partitioning. In the results section of the paper we show the first step in a series of evaluations that are needed to show a great improvement in partitioning methods with our method, Growable partitioning. The problem in the experiments was to find areas of high visibility. The growable partitioning method, although generally slower than the K-means Clustering method, generated better quality of partitions. The framework we created is modular even though our experiments show only one use of the framework.

In the future, the algorithm of growable partitioning needs to be modified to allow for three ranges of visibility - high, medium, and low in order for the partitioning method to be applied to the motion planning framework already set up in the Unsupervised Adaptive Strategy[13]. The algorithm also needs to analyze the effects of quality partitions on the Motion Planning Problem and parallel graphing algorithms. In the case of parallel algorithms, each partition could be assigned to a processing core and a study of the effects of quality partitioning on the speed of such algorithms can be done to see if any speedup or overhead exist with these partitions. There is much work to do, but our work is the first step in improving the current methods to solve all these problems.

## References

- [1] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *Robotics: The Algorithmic Perspective*, pages 155–168, Natick, MA, 1998. A.K. Peters. Proc. Third Workshop on Algorithmic Foundations of Robotics (WAFR), Houston, TX, 1998.
- [2] J. Berg and M. Overmars. Using workspace information as a guid to non-uniform sampling in probabilistic roadmap planners. *Int. J. Robot. Res.*, 24(12):1055–1072, 2005.
- [3] J. F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1988.
- [4] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.
- [5] T. Kanungo, D. M. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 24(2002):881–892, 2002.
- [6] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, 12(4):566–580, August 1996.
- [7] H. Kurniawati and D. Hsu. Workspace-based connectivity oracle - an adaptive sampling strategy for prm planning. In *Algorithmic Foundation of Robotics VII*, pages 35–51. Springer, Berlin/Heidelberg, 2008. book contains the proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR), New York City, 2006.
- [8] J.-M. Lien, S. L. Thomas, and N. M. Amato. A general framework for sampling on the medial axis of the free space. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 4439–4444, September 2003.
- [9] L. Lieu and N. Saito. Automated shapes discrimination in high dimensions. *Proc. of SPIE*, 6701 – Wavelets XII(67011W), 2007.

- [10] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, October 1979.
- [11] M. Morales, L. Tapia, R. Pearce, S. Rodriguez, and N. M. Amato. A machine learning approach for feature-sensitive motion planning. In *Algorithmic Foundations of Robotics VI*, pages 361–376. Springer, Berlin/Heidelberg, 2005. book contains the proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR), Utrecht/Zeist, The Netherlands, 2004.
- [12] M. A. Morales A., L. Tapia, R. Pearce, S. Rodriguez, and N. M. Amato. C-space subdivision and integration in feature-sensitive motion planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 3114–3119, April 2005.
- [13] L. Tapia, S. Thomas, B. Boyd, and N. M. Amato. An unsupervised adaptive strategy for constructing probabilistic roadmaps. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2009.