A Machine Learning Approach for Feature-Sensitive Motion Planning

Marco Morales, Lydia Tapia, Roger Pearce, Samuel Rodriguez, and Nancy M. Amato
{marcom,ltapia,sor8786,rap2317,amato}@cs.tamu.edu

Technical Report TR04-001
Parasol Lab
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112

February 22, 2004

Abstract

Although there are many motion planning techniques, there is no single one that performs optimally in every environment for every movable object. Rather, each technique has different strengths and weaknesses which makes it best-suited for particular types of situations. Also, since a given environment can consist of vastly different regions, there may not even be a single planner that is well suited for the problem. Ideally, one would use a suite of planners in concert to solve the problem by applying the best-suited planner in each region.

In this paper, we propose an automated framework for feature-sensitive motion planning. We use a machine learning approach to characterize and partition C-space into (possibly overlapping) regions that are well suited to one of the planners in our library of roadmap-based motion planning methods. After the best-suited method is applied in each region, their resulting roadmaps are combined to form a roadmap of the entire planning space. We demonstrate on a range of problems that our proposed feature-sensitive approach achieves results superior to those obtainable by any of the individual planners on their own.

1 Introduction

The motion planning (MP) problem consists of finding a sequence of valid states to move a robot from an initial configuration to a goal configuration. In robotics path planning, valid states are typically collision-free robot configurations. In addition to robotics, instances of the MP problem can be found in a range of applications from robotics and CAD to computational biology.

The MP problem is thought to be intractable except for robots with few degrees of freedom (DOF) as there is strong evidence that any complete planner will require exponential time in the number of DOFs of the robot [22, 23, 9]. The complexity of the MP problem has led researchers to investigate heuristic planners with reasonable success, first based on cell decomposition [7] and potential fields [15]. After the introduction of the Randomized Path Planner (RPP) [2], randomized methods have been the focus of MP research and a number of techniques have been developed that were successful in solving several previously unsolved MP problems. Notable examples include the graph-based probabilistic roadmap methods (PRMs) [14] and several tree-based methods that explore the planning space starting from one or two points, e.g., the Ariadne's Clew Algorithm [3], RRT [16], and Hsu's algorithm [12].

Although there are a number of good randomized techniques available, no single method is general enough to perform optimally for every instance of the MP problem. Rather, the performance of each technique will vary depending on the problem instance. For example, some motion planning techniques are better suited for mostly free environments while others show their real strength in cluttered regions. Also, since a given environment can contain different types of regions, there may not even be a single planner that is well suited for all regions of the problem.

An ideal motion planner would be a meta-planner using a suite of specialized planners to cooperatively solve MP problems. It would automatically apply the best-suited planner for each distinct region in the planning space and assemble a solution to the MP instance by composing the solutions for the regions. This is the objective of our research. In particular, we propose an automated framework for feature-sensitive motion planning. We use a machine learning approach to characterize and partitition C-space into (possibly overlapping) regions that are well suited to one of the planners in our library of roadmap-based motion planning methods. The partitioning is accomplished by recursively subdividing C-space until we obtain homogeneous regions as classified using a set of features measured for each region. Each, now homogeneous, region is matched with the best-suited planner in our motion planning library and the selected planners are applied to produce roadmaps for each region. Then, these regional roadmaps are combined to form a roadmap of the entire planning space. We demonstrate on a range of problems that our proposed feature-sensitive approach achieves results superior to those obtainable by any of the individual planners on their own.

2 Preliminaries

Research in motion planning is very active. Here we focus on the basic tools for MP ant the most successful randomized approaches with an emphasis on the need to integrate different methods.

Configuration space. A robot's placement can be represented as a point $(x_1, x_2, ..., x_n)$ in an n dimensional space, with a parameter x_i for each of its n degrees of freedom (DOF). The space consisting of all possible placements or configurations (feasible or not) is the robot's configuration space (C-space) [17]. The set of all feasible configurations is called C-free, while the unfeasible configurations form C-obstacles. Thus, the MP problem becomes that of finding a trajectory for a point in C-space that completely lies in C-free. And although it is intractable to compute the C-space, we can often determine whether a configuration is feasible or not quite efficiently, e.g., with a collision detection test.

Randomized motion planning techniques. Randomized MP algorithms attempt to avoid the costly computation of C-space by determining the connectivity of C-space by randomly sampling configurations. Two classes of randomized planners are graph-based and tree-based techniques.

PRMs [14, 20, 19] build a graph called a roadmap in two steps: node generation and node connection. Nodes are feasible robot configurations generated at random. In the connection stage pairs, of selected nodes are processed by simple local planners that interpolate the path between them to keep those that are feasible as edges. Different heuristics have been proposed to produce PRMroadmaps. Basic PRM[14] uses uniform random sampling to generate configurations. Other sampling methods have been devised to increase the proportion of feasible configurations generated. OBPRM[1] samples configurations near C-obstacle surfaces, Gaussian PRM[5] biases sampling towards regions close to C-obstacles. Many other heuristics have been proposed [18], [4], [24], [16]. The performance of these methods varies in different kinds of

environments, for example OBPRMperforms better in cluttered regions and basic PRMis good in open areas.

Tree-based techniques start with a known feasible configuration and grow a tree by selecting new nodes in the tree according to some strategy that either makes the tree approach a goal configuration or uniformly cover the space reachable from the nodes already in the tree. Some of these methods are particularly well suited for problems with kinodynamic constraints. Among these techniques we find the Randomized Path Planner (RPP) [2], the Ariadne's Clew Algorithm [3], RRTs [16], and Hsu's algorithm [12]

C-space subdivision. C-space subdivision consists of breaking some or all the parameters of a MP problem into intervals to obtain subsets of the C-space. Uniform cell decomposition was the basis of some of the early heuristic approaches for motion planning. C-space variables were usually divided into a number of equally sized pieces each, producing m^n cells, where n is the number of DOFs of the robot and m is the number of divisions in each dimension. This approach is only feasible for very few DOF problems.

Non-uniform C-space subdivision has been explored in two stage planners such as Decomposition-based Motion Planning [6], which has been successful in decomposing planning problems into simpler subproblems with a reduced complexity. It searches for homotopic paths in workspace, thus ignoring higher DOFs and effectively making subdivisions on the first three parameters of the C-space. Once a path is found in workspace the motions of the joints, the higher DOFs, are determined. Here, the emphasis is in breaking down a complicated high-dimensional problem into smaller problems for real-time planning.

We believe that C-space subdivision can be an effective technique to focus the effort of more powerful MP techniques better suited to solve specific pieces of the problem. But in order to be successful, the subdivision cannot be exhaustive nor uniform, it needs to be feature-sensitive and used to detect regions with somehow uniform features.

C-space characterization. Some basic studies of the performance of planners in different kinds or classes of environments have already been performed. Dale and Amato proposed a method for characterizing C-Space [10] by extracting four features from a roadmap made with basic PRMto classify the C-space as free, cluttered, narrow passage or blocked and determine what planner is best. It is also proposed to subdivide regions to focus specialized planners, but no subdivision mechanism was introduced. This approach is not fully automatic and relies on a small set of features for classification. Another study by Geraerts and Overmars [11] compares several PRMplanners in five environments with different features: a corridor, a room, a clutter, a hole, and a house. This study gives criteria for PRMusers in order to decide what PRMvariants to use.

In order to characterize C-space we need to find representative MP problems that well represent the set of all possible MP problems. We also need fair methods to rank the quality of the characterization. Finally we need to be able to analyze the performance of different planners in terms of the quality of the solution and computing resource usage. This is why it is natural to search for insight in areas that have studied large searching spaces such as machine learning and artificial intelligence.

Machine learning methods have gained attention over the past few years because of their ability to construct representations for complex and high dimensional data. For example, determining the three-dimensional structure of a protein given only knowledge of the sequence of a protein is a difficult problem that many bioinformatics groups have addressed. Using knowledge obtained by a learning method about the sequence to structure relationship, some progress has been made in this area [8].

Learning methods are attractive when the scope of data analysis is too complex for a human. The FIRST dataset is a collection of images of radio sources collected by the Very Large Array. In six years this dataset grew to consist of 32,000 images of two million pixels each. For this difficult problem, the decision tree algorithm C4.5 was trained to identify galaxies with the bent-double morphology. The decision tree was able to classify bent-doubles and find galaxies that had been overlooked during manual searches [13].

3 General Framework

We propose a framework for mapping the connectivity of the C-space of motion planning problems through the coordinated operation of a library of randomized planning methods. This framework, sketched in Figure 1 and explained in Algorithm 1, subdivides the C-space of a problem into a set of possibly overlapping homogeneous regions. It then produces a roadmap in each region using the planner best suited to that region as determined by the region's features. Finally, the regional roadmaps are integrated to form a roadmap for the general problem.

This framework is feature-sensitive. It recognizes easy regions as such so that less time will be spent mapping them, and treats difficult regions with more powerful planners.

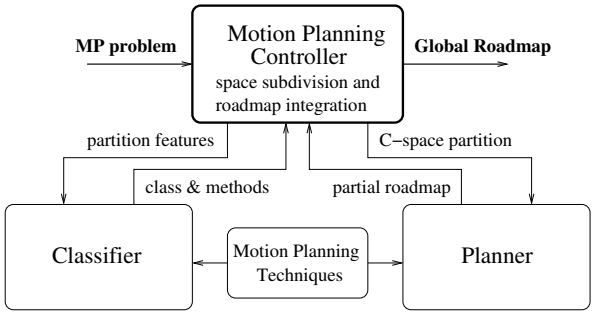


Figure 1: Block diagram of the framework

Algorithm 1 Feature-sensitive Motion Planner

Input: Environment $E : (robot, obstacles)$, set of Motion Planners M
Output: S: a C-space Roadmap for E

```
R = Set of overlapping partitions of E's C-space
for each region r_i ∈ R do
    M_r = subset of M suitable to solve r_i
    for each method m_j ∈ M do
        s_{i,j} = roadmap found by applying m_j to r_i
    end for
end for
integrate s_{i,j}'s into an overall roadmap S
```

A **Motion Planning Controller** coordinates all the modules of the framework. It recursively subdivides the environment and creates a tree representing the subdivision where nodes represent C-space regions and the children of a node correspond to the regions that result from the subdivision of their parent node.

Each node stores features from the associated region's C-space and feeds them to the **Classifier** which matches them to a class and a set of *Motion Planning Techniques* with some degree of confidence. Based on this output, the size of the subdivision, and the granularity of the subdivisions already performed, the **Motion Planning Controller** decides whether to further subdivide the current C-space region or mark it as a leaf node. Leaf node regions are sent to the **Planner** which produces a partial roadmap with the assigned *Motion Planning Technique*. Then partial roadmaps corresponding to the children of intermediate nodes are merged into a roadmap representing the parent node's region, so that larger and larger regions are integrated until the global roadmap is obtained in the root node.

The features computed and the classifier are closely interrelated. We need to be able to extract features in an efficient and inexpensive way. These features should not depend on the number of degrees of freedom (DOF) of the problem nor on the volume of the C-space of the partition. Also, features should help to describe the complexity of the partition. The classifier needs to be a classification-based method that makes understandable decisions and gives as output a class for each partition and some confidence level. Ideally the classifier should be able to directly match methods to partitions. Subdivisions are made to ease classification and improve the chances of using the proper method efficiently, and they should be stopped once a good classification was made. Our approach differs from cell decomposition in that we do not perform uniform subdivision, instead we control the subdivision of C-space by the needs of the classification.

Roadmap integration should concentrate on connecting graph components of different partitions in the overlapping portion of the respective regions.

4 Implementation

In the following sections we describe a prototype implementation of our framework. We note that each module can support different implementations.

4.1 Motion Planning Controller (subdivision and integration)

The **Motion Planning Controller** is the basis of our approach. Its goal is to subdivide C-space into regions that can be classified as so-called homogeneous, to create roadmaps for each homogeneous region using the selected planning method, and to integrate the regional roadmaps into a general roadmap. A homogeneous region is one that the **Classifier** identifies as belonging to one of the identified classes, which currently includes free, cluttered, or narrow passage. The **Classifier** is explained in detail below.

Our implementation is shown in Algorithm 2. It begins by obtaining features from the problem's C-space and having the classifier match it to a class of the environment. The region covering the problem's entire C-space corresponds to the root of the tree. It is subdivided according to the recursive 'branching' Algorithm 3.

The branching algorithm 3 subdivides the current node in two or more overlapping *subdivisions*. If it can't further subdivide the node due to limit constraints it returns the node as a leaf. Features for each subdivision are extracted so the **Classifier** can match them to a class. If a subdivision meets leaf criteria it is marked as such, otherwise the branching algorithm is recursively called for the subdivision. In our current

Algorithm 2 Motion Planning controller

Input: Environment E : ($robot, obstacles$), subdivision limit L
Output: *roadmap*: a C-space Roadmap for E

```
1: class = Classifier.match(getFeatures(E))
root, class, roadmap = Branch(E,class,L,0)
3: if roadmap == "noroadmap" then
4:    roadmap = Planner.makeRoadmap(root)
5: end if
```

implementation, leaf criteria are met when a subdivision has reached its limit in size, or when its class is the same as that of the parent whose classification is other than non-homogeneous.

If the *subdivision*s meet class merging criteria they are merged into their parent, and the merged node will become a leaf with the same base subdivision and a class that is the result of merging the classes of its *subdivision*'s classes. Merging criteria in our current implementation are met when all of the subdivisions are leaves and belong to the same class, and the parent's class is non-homogeneous; the resulting region is assigned to the same class as the *subdivision*s.

The last step is calling the **Planner** to make roadmaps in each of the children. The planner will use the best method for the class assigned by the **Classifier**. Then all the subdivisions are merged into the parent by merging the roadmaps of the children into one and composing its classes.

Since roadmaps are successively merged to create their parent's roadmap, the final result is a global roadmap assigned to the root of the tree, unless no subdivision was made. In the latter case a roadmap still needs to be made for the entire problem by the **Planner** which will use the best technique for the class assigned by the **Classifier**. This can happen due to a problem that is not feasible to subdivide because the subdividing algorithm couldn't get a subdivision or because its root node meets the leaf criteria.

It is worth noting that the **Classifier** actually constructs a simple roadmap in each node which is used to obtain the features. This roadmap is integrated into the node's roadmap by the **Planner** when making leaf roadmaps and when integrating branch roadmaps into their parents.

Feature extraction. In order to capture characteristics of C-space that will give insight into the best planning method for that type of space, we collect a set of features. These features are extracted from a small roadmap created with a uniform and quick basic PRM. Several measurements are made using this roadmap to describe the complexity of the partition.

Algorithm 3 Branch

Input: C-space subdivision D, class C, subdivision limit L, node height H
Output: [subdivision, class, roadmap]

```
1: (subdivisions, error) = subdivide(D,L)
2: if no error then
3:    for all d_i in subdivisions do
4:       class_i = Classifier.match(getFeatures(d_i))
5:       if MeetLeafCriteria(d_i,class_i,C,L,H) then
6:          p_i = [d_i, C, "noroadmap"]
7:       else
8:          p_i = Branch(d_i,class_i, L, H+1)
9:       end if
10:      add p_i to children
11:   end for
12:   if MeetClassMergingCriteria(subdivisions,C)
      then
13:      result = [D, MergeClasses(subdivisions,C),
         "noroadmap"]
14:   else
15:      for all p_i in children do
16:         if p_i.roadmap == "noroadmap" then
17:            Planner.makeRoadmap(p_i)
18:         end if
19:      end for
20:      result = mergeBranches(D, children)
21:   end if
22: else
23:   result = [D, class, "noroadmap"]
24: end if
25: Return result
```

Basic PRMis one option to extract the features but other methods can be used too.

Roadmap features. Our objective is a general method that can classify a wide range of environment types and complexities. To achieve this goal, we extract a set of features that captures the characteristics of each type of space while not being impacted by size and complexity of C-space. Specifically, these features are independent of the dimensionality and volume of the C-space being analyzed. The set of 23 features that we have chosen to characterize C-space are explained below.

General features.

- *Free node ratio:* the ratio of free nodes in the roadmap to configurations attempted. It gives insight into the "freeness" of C-space. In partitions with no obstacles, this ratio will be one whereas in regions where obstacles cover most of C-space

it will be close to zero. It is defined as:

$$FreeNodeRatio = \frac{\#free\ nodes}{\#nodes\ tried} \quad (1)$$

- *Free components:* the number of free connected components in the roadmap. It gives an idea of the connectivity of C-space. A count of one would indicate a region where a path is easily found. In highly cluttered areas, sampled nodes will be harder to connect, therefore this count will approach the number of nodes sampled.

- *Edge ratio:* the ratio of edges in the roadmap to connections attempted. In environments with free nodes that are difficult to connect, this ratio will be low. This will be the case in cluttered regions, close to the surface of complex C-obstacles, and in narrow passages. It is defined as:

$$EdgeRatio = \frac{\#edges}{\#edges\ tried} \quad (2)$$

Component-based features. These features are related to the complexity, extension, and coverage of components, which map C-free and C-obstacles, and the space between them. The maximum, minimum, mean, and standard deviation are calculated in each component, giving four features in each description. Each is averaged over the number of components. Except when noted, they are normalized by dividing them over a normalization constant $k_{normalize}$ defined below to make them independent of C-space dimensionality and size.

- *Node distance:* the distance between the nodes in the component. In packed components it will be smaller than in those covering elongated areas. For component CC_k the distance between node pairs in that component is given by

$$d = \|v_i - v_j\| \text{ for each } v_i,\ v_j\ \in CC_k\ \&i \neq j \quad (3)$$

- *Edge length:* the length of edges in the component. Components in cluttered regions will have shorter edges than those in more open space. Edge lengths are taken directly from the roadmap.

- *Distance to center:* the distance of the nodes in the component to its center of mass. Nodes in components spanning wide regions of C-space will be farther to the center of mass than packed nodes. The distance between nodes is computed as in equation 3 and the center of mass of component CC_k is given by:

$$CenterOfMass_{CC_k} = \frac{\sum v_i \in\ CC_k}{\|CC_k\|} \quad (4)$$

- *Component distance:* an approximation of the distance between components, meaning how much infeasible space is in between them. The distance between components CC_i and CC_j can be computed in different ways, for example as the distance between the closest pair of nodes of each component as follows:

$$d(CC_i, CC_j) = min\{|v_n - v_m|\} \quad (5)$$

$$\forall v_n \in CC_i, v_m \in CC_j$$

or as the distance between the center of masses of CC_i (CM_{CC_i}) and CC_j (CM_{CC_j}):

$$d(CC_i, CC_j) = |CM_{CC_i} - CM_{CC_j}| \quad (6)$$

- *Component length:* the sum of lengths of all the edges in the component. It gives an idea of the volume covered by the components. In contrast with other features, this one is normalized by dividing it over the length of all the roadmap. The length of a component CC_k already normalized is given by

$$ComponentLength(CC_k) = \frac{\sum_{e_i \in\ CC_k} \| e_i \|}{\sum_{e_j \in\ rdmp} \| e_j \|} \quad (7)$$

We normalize the features that are not already normalized by dividing them over a normalization constant $K_{normalize}$ that is related to the volume of the C-space covered in the sampling relative to the dimensions and degrees of freedom of the robot. Here we define $K_{normalize}$ as:

$$K_{normalize} = max|dimension_i| * |dof| \quad (8)$$

where $|dimension_i|$ is the length of dimension i in the partition being mapped, $1 \leq i \leq |dof|$, $|dof|$ is the number of degrees of freedom of the problem.

Sampling for feature collection should also be sufficient to capture enough features to decide whether the piece of C-space analyzed can be properly classified or not. It might be needed to relate the number of sampled nodes to $K_{normalize}$, and the number of sampled nodes in this stage needs to be kept small.

Subdivision of C-space regions. The subdivision of a partition can be obtained with different methods.

In this implementation we have an elementary partitioning strategy that randomly chooses one of the first positional dimensions of the robot to partition. A random point within the range of the chosen dimension is selected, a small ϵ is used to divide it into two overlapping ranges. However this approach has the disadvantage of not considering the features of C-space itself. In the future we plan to create partitions in between roadmap components, because that way it is more likely to find interesting subdivisions.

Subdivision roadmap generation and integration. The **Planner** is given a subdivision with a set of assigned planning techniques, it uses the implementation of the chosen planners and options to create a roadmap for the subdivision of C-space given and gives it back to the **Motion Planning Controller**. When subdivisions are to be integrated the **Planner** merges them by attempting component connection instead of simple node connections, since the regions overlap, it concentrates the connection attempts in the overlapping portions of the two regions.

4.2 Classifier

The classifier is given a set of features of a C-space partition, and it outputs a matching class and a confidence level. It is hard to derive analytical expressions defining classes of C-space or rankings for choosing good rankings for several reasons: many features that only represent a fraction of C-space can be extracted; there are many MP strategies to map C-space; and the range of MP problems is too diverse in obstacle placement and robot dimensionality. Machine learning methods can be used to perform the classification. For our particular needs of understandable decisions, discrete classifications, and ease of use, decision tree learning methods are appropriate.

C-space classes. As mentioned in Section 1, MP environments are often non-homogeneous spaces with homogeneous subcomponents. In this implementation we currently use four classes of C-space that we plan to extend in the future.

- *Free:* regions having very few obstacles.
- *Cluttered:* regions cluttered with obstacles.
- *Narrow passage:* regions are overly cluttered, but have narrow regions through which the robot can pass.
- *Non-homogeneous:* regions which the classifier cannot place in one of the other classes.

Machine learning techniques. There are many learning algorithms that work well for discreet-valued classification. From our learning method we need an algorithm that provides us with understandable decisions for verification, relatively quick learning time, and ease of use. For these particular needs, we have selected the decision tree method, C4.5 [21].

The C4.5 program creates a series of decisions based on feature values that leads to a classification of an example. The decision nodes or branches in the trees are selected by choosing the feature that "best" splits the space. Every path along the tree leads to a leaf node or classification based on the set of decisions that lead to that leaf. After a tree is constructed, C4.5 removes paths that do not apply to a general portion of the data through a process called pruning.

Classifier training. The classifier is trained with examples of free, cluttered, narrow passage, and non-homogeneous environments. Features are extracted, a decision tree is trained, ordered rules are composed from this tree, and the performance evaluated via a k-fold cross-validation approach, with k being 10% of our data. Although the current set of classification rules do not take into account all of the features we examined, these features might be utilized for more other environments such as isolated or blocked spaces.

4.3 Planner and pool of motion planning methods

The **Planner** works as a regular motion planner which uses a pool of planning techniques from which to select. It receives an environment with a robot, and a set of obstacles and a bounded region of C-space in which to plan. It is also able to merge two roadmaps into one with a component connection strategy.

Our framework is independent of the motion planning methods used as long as we have information about how well they perform in different classes of C-space. In this work we use basic PRMfor free regions and OBPRMfor cluttered, narrow passage, and non-homogeneous regions, but more methods can be easily added to the framework. In addition we plan to investigate how to make the classifier learn also about the motion planning methods so that both methods and parameters can be automatically selected and assigned to each partition.

5 Experiments

We tested our implementation on an Intel Pentium 4 2.8 GHz processor with 512 MB of RAM and 512

kB of cache, running Linux 2.4.20-9 OS. We used our group's C++ motion planning library compiled with gcc 3.2.2 and Perl to control the partitioning. MySQL was used for storing features for training.

Training the classifier. We designed a set of 200 environments: 50 free, 50 cluttered, 50 narrow passages, and 50 non-homogeneous. Features were extracted from each environment as described before and saved in a database of environments from which we obtained data to train decission trees. Ten different sets of trees were created using 180 examples for training and a distinct set of 20 examples for testing. Accuracy rates of the rules produced by these cross-validation tests ranged from 100% to 85% with an average testing accuracy of 95%. We selected the tree that was able to classify 98.9% of our training and 100% of our testing examples to use for our classification (Table 2).

Test environments. For testing, we chose four environments with different kinds of obstacles and robots that are shown in Figure 3.

In the house environment (Fig. 3(a)), there are three rooms, one with some furniture. The robot is a table that needs to be moved from outside the house to the smallest room. It has different kinds of regions: free outside the house, cluttered close to the furniture, and a narrow passage must be traversed to reach the last room.

The walls-rigid environment (Fig. 3(b)), has 5 chambers with varying size openings connecting them. Three of the chambers are filled with small obstacles and the other two are empty. The robot is a stick that traverse all the chambers.

The walls-serial environment (Fig. 3(c)), is similar to the walls-rigid, but all the chambers are free of obstacles. The robot, however, is articulated with two long links, making it harder to move between chambers than in the walls-serial.

The narrow-clutter environment (Fig. 3(d)), is an open area with a big obstacle that can be traversed through a narrow passage followed by some open space and a clutter of small obstacles. The robot is articulated with 4 small links. It needs to move through the narrow passage and the clutter all the way down to the oposite side of the environment.

Experimental design. We applied Basic PRM, OBPRM, and our framework (C&P, for classify and partition) to the environments to find the minimum number of nodes needed to solve the problem. The minimum number of nodes was found by a binary search, and it was repeated 4 times for each environment and method pair.

In Basic PRM, nodes were sampled at random, connections are done with the k-closest strategy. In OBPRM, nodes were generated with the normal alignment strategy, connections are done with k-unconnected-closest strategy, a variant of the standard k-closest which only attempts connections to different components. Our local planners are straight-line and rotate-at-s 0.5 in all experiments.

In C&P we have the three stages described above: feature gathering, roadmap generation, and roadmap integration. In feature gathering, we use random sampling and k-closest connection strategy. In roadmap integration we use k-closest and components connection strategies. In roadmap generation we assign techniques depending on the classification of the region assigned. In free regions we do not do further mapping. In nonhomogeneous regions we use random sampling along with OBPRMsampling and connect with k-unconnected-closest strategy with five times the number of nodes used to collect features. In narrow regions we use OBPRMsampling and k-unconnected-closest with as many as fifteen times the number of nodes required for collecting features. In clutter regions we use the same strategy as in narrow, but the number of nodes is only five times the number of nodes used in feature collection.

Tables 1, 2, 3, and 4 show results of representative runs. We show the number of nodes generated (act), attempted (att), the number of collision detection calls (#CDs), the number of connected components (#CCs), and we indicate whether the query was solved.

From Table 1, it is easy to see the benefit of characterization and partitioning (C&P). The number of collision detection calls required to solve the house environment using C&P is far fewer than when using BasicPRMand OBPRMby themselves. Fewer nodes are also needed to solve the problem with the C&P framework than when using each method individually. During the roadmap generation stage, after the partitions have been classified, node generation methods that are well suited for each classification can be applied.

The house environment is classified as a whole as a nonhomogeneous environment. When partitioned and classified, useful partitions are found. When the narrow partition is identified we are able to focus many more OBPRMnodes in that difficult area. Similarly, when the free partitions are found, we can assume that this is an easily mapped area.

The results for the Walls Rigid environment can be seen in Table 2. OBPRMis able to solve the query with far fewer nodes than BasicPRM, although OBPRMrequires many more collision detection calls. Although C&P also requires many more collision detection calls than BasicPRM, C&P is able to solve the

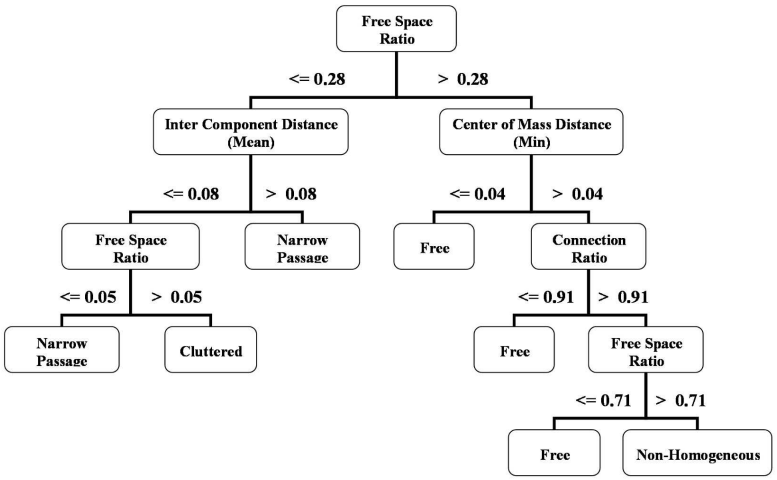
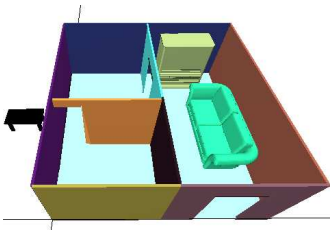
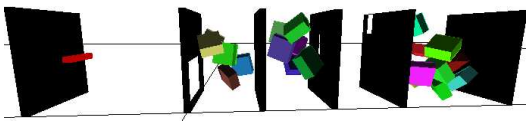


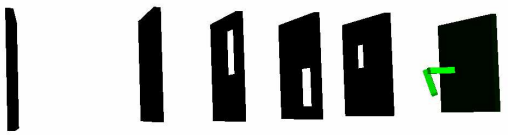
Figure 2: Decision tree selected for classification



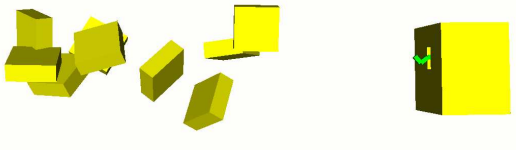
(a) House (6 DOF)



(b) Walls-rigid (6 DOF)



(c) Walls-serial (7 DOF)



(d) Narrow-clutter (9 DOF)

Figure 3: Test environments

query with only 257 nodes. This is far fewer than the number of nodes required by either OBPRMor BasicPRM. Difficult or unclear areas of the environment are also discovered during characterization.

As can be seen in Fig. 3, this environment is quickly solved by C&P. Significant narrow portions of the environment are identified and we can thus focus on those difficult areas. It is important to note that the robot in this environment was articulated and narrow passages are still identified.

Table 4 shows the strength of C&P. After the entire environment is classified as nonhomogeneous, the planner focuses on the second half of the environment around the narrow passage. The concentrated effort on the narrow passage allows C&P to use considerably fewer collision detection calls than OBPRM. BasicPRMwas never able to find a solution for the narrow region in the alloted nodes (20,000).

When possible, we join partitions that are labeled with the same class. In our early experiments we didn't enable this feature, but we realized that it was also important not to subdivide regions that belong to the same class.

6 Conclusions

We have proposed an automated framework for feature-sensitive motion planning. A prototype implementation of our machine learning classification and partitioning approach was shown to achieve results su-

Table 1: Performance metrics for House environment

Method	Nodes$_{act/att}$	#CDs	#CCs (partitions detected)	solved?
BasicPRM	5798/5798	1,385,004	9 1 large CC spanning all rooms	yes
OBPRM	489/491	3,562,679	20 1 large CC spanning all rooms	yes
C&P:total	226/375	724,861	3 1 large CC spanning all rooms	yes
partitioning	102/225	90,917	29 (1 narrow, 4 free, many non-homogeneous)	n/a
roadmap gen.	124/150	326,076	30	n/a
roadmap int.	0/0	307868	3	yes

Table 2: Performance metrics for Rigid Body Walled environment

Method	Nodes$_{act/att}$	#CDs	#CCs (partitions detected)	solved?
BasicPRM	3000/3000	2,422,749	11	yes
OBPRM	1200/1179	11,107,232	48	yes
C&P:total	257/420	7,635,344	10	yes
partitioning	143/285	164,302	35 (1 narrow, 3 free, many non-homogeneous)	n/a
roadmap gen.	114/135	72,979	30	n/a
roadmap int.	0/0	7,398,063	10	yes

perior to those obtainable by any of the individual planners on their own.

In the future, we plan to extend our framework to handle more C-space classes such as blocked and isolated regions. We also plan to incorporate more motion planners into our framework. Machine learning can also be used to learn about a planner's performance in different types of C-space and to select an appropriate planner and its parameters based on previous experience.

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. Proceedings of the Third Workshop on the Algorithmic Foundations of Robotics (WAFR), Houston, TX, 1998.

[2] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *Int. J. Robot. Res.*, 10(6):628–649, 1991.

[3] P. Bessiere, J. M. Ahuactzin, E.-G. Talbi, and E. Mazer. The ariadne's clew algorithm: Global planning with local methods. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, volume 2, pages 1373–1380, 1993.

[4] R. Bohlin and L. E. Kavraki. Path planning using Lazy PRM. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 521–528, 2000.

[5] V. Boor, M. H. Overmars, and A. F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1018–1023, 1999.

[6] O. Brock and L.E. Kavraki. Decomposition-based motion planning: A framework for real-time motion planning in high-dimensional configuration places. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2001.

[7] R. A. Brooks and T. Lozano-Pérez. A subdivision algorithm in configuration space for findpath with rotation. In *Proc. Int. Conf. Artif. Intel.*, pages 799–806, 1983.

[8] C. Bystroff and D. Baker. Prediction of local structure in protiens using a library of sequence-structure motifs. In *J. Mol. Biol.*, volume 281, pages 565–575, 1998.

[9] J. F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1988.

[10] L. K. Dale and N. M. Amato. Probabilistic roadmaps – putting it all together. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1940–1947, 2001.

[11] Roland Geraerts and Mark H. Overmars. Comparative study of probabilistic roadmap planners. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, Dec 2002.

[12] D. Hsu, R. Kindel, J-C. Latombe, and S. Rock. Randomized Kinodynamic Motion Planning with Moving Obstacles. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages SA1–SA18, 2000.

Table 3: Performance metrics for Serial Walls (2-link) environment

Method	**Nodes**$_{act/att}$	**#CDs**	**#CCs (partitions detected)**	**solved?**
BasicPRM	17801/17801	3,183,940	20 3 large CCs	no
OBPRM	816/821	2,830,528	10 1 large CC, 8 CCs between 12 & 60 nodes	yes
C&P:total	590/795	1,460,845	2 1 large CC	yes
partitioning	168/345	67,695	35 (2 narrow, 5 free, many non-homogeneous)	n/a
roadmap gen.	422/450	81,263	145	n/a
roadmap int.	0/0	1,311,887	2	yes

Table 4: Performance metrics for Narrow environment

Method	**Nodes**$_{act/att}$	**#CDs**	**#CCs (partitions detected)**	**solved?**
BasicPRM	19461/19461	40,056,265	110 2 large CCs	no
OBPRM	350/350	23,142,065	49 1 large CC	yes
C&P:total	271/435	5,966,310	60 1 large CC	yes
partitioning	68/225	1,017,267	29 (2 narrow, 1 cluttered, many non-homogeneous)	n/a
roadmap gen.	203/210	2,129,146	143	n/a
roadmap int.	0/0	2,819,897	60	yes

[13] C. Kamath, E. Cantu-Paz, I. Fodor, and N. Tang. Classifying bent-double galaxies. In *IEEE Comput. in Sci. & Eng.*, volume 4, pages 52–60, 2002.

[14] L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, 12(4):566–580, August 1996.

[15] O. Khatib. Real–time obstacle avoidance for manipulators and mobile robots. *Int. J. Robot. Res.*, 5(1):90–98, 1986.

[16] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 473–479, 1999.

[17] T. Lozano-Pérez and M. A. Wesley. An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacle. *Communications of the ACM*, 22(10):560–570, October 1979.

[18] C. L. Nielsen and L. E. Kavraki. A two level fuzzy prm for manipulation planning. Technical Report TR2000-365, Computer Science, Rice University, Houston, TX, 2000.

[19] Overmars and P. Svestka. A probabilistic learning approach to motion planning. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages 19–37, 1994.

[20] M. Overmars. A random approach to path planning. Technical Report RUU-CS-92-32, Computer Science, Utrecht University, The Netherlands, 1992.

[21] J.R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[22] J. Reif. Complexity of the piano mover's problem and generalizations. In *Proc. IEEE Symp. Foundations of Computer Science (FOCS)*, pages 421–427, 1979.

[23] J. T. Schwartz and Micha Sharir. On the "piano movers" problem II: General techniques for computing topological properties of real algebraic manifolds. *Adv. Appl. Math.*, 4:298–351, 1983.

[24] S. A. Wilmarth, N. M. Amato, and P. F. Stiller. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1024–1031, 1999.