An Algorithm for Construction of Probabilistic Road Maps with Reduced Numbers of Redundant Edges

Sims Osborne
Department of Computer Science, University of North Carolina at Chapel Hill

Abstract

We introduce an algorithm for probabilistic roadmap construction, k-nearest-multiD, which improves upon the existing k-nearest-PRM algorithm by reducing the number of redundant edges in a PRM. We test our algorithm experimentally and compare it to the common form of the k-nearest algorithm on two configuration spaces with many input parameters.

1 Introduction

1.1 Background and Goals

Probabilistic Road Maps, or PRMs, have a long history of use in motion planning algorithms [6]. As global route planners, they are probabilistically complete and, once created, can be used for route planning between any two accessible points in a given environment.

Every PRM algorithm has two steps in common. First, given a configuration space, select obstacle-free points until a desired number of points or some other requirement has been satisfied. Second, connect the points. The result is a probabilistic road map which can be queried using common graph path-finding algorithms to find a path between different nodes of the resulting graph.

Unfortunately, while PRMs are probabilistically complete [5], they are often required to be very large in order to successfully map a space such that a path can be found between any two points for which a feasible path exists. A large PRM takes longer to create, requires more space to store, and requires more time for each query.

One particular problem that occurs with PRMs is the tendency of closely-placed nodes to form strongly connected subgraphs while remaining unconnected to other nodes which are only slightly further away. We attempt to address this problem by introducing a new algorithm, k-nearest-multiD, which forces nodes to connect to other nodes in multiple directions. We believe this algorithm, referred to as multiD for short, will reduce the number of redundant edges in a graph, reducing the necessary size of a PRM required for a given level of connectivity. Our algorithm is discussed in detail in Section 2.

1.2 Related Work

In the classical version of the algorithm, a pre-determined number of free vertices are selected from the sample space. For each vertex, the algorithm then attempts to connect it to other vertices within a radius $r$ to which it is not already connected. [6, 4]. A simplified version, (s)PRM connects vertices to nearby vertices without regard to whether they are already connected [5, 4]. Another variant, which is more similar to our proposed new method, is $k$ – nearest (s)PRM, attempts to connect a vertex to its nearest $k$ neighbors, where $k$ is a positive integer, rather than relying on radius to determine nearby nodes([4, 7]).

Karaman and Frazzaoli propose $PRM^*$ and $k$ – nearest $PRM^*$, in which the radius that defines near or the integer $k$, respectively, are defined as functions of the size of the sample space and the number of vertices within the sample space. They find that $PRM^*$, $k$ – nearest $PRM^*$, and sPRM are asymptotically optimal in addition to being probabilistically complete, but $PRM$ and $k$ – nearest $PRM$ are not ([4]). Our proposed algorithm is a variation of the k-nearest-PRM and k-nearest-PRM* algorithms.

While most PRM approaches construct work by connecting vertices to form collision-free paths, Bohlin and Kavraki’s LazyPRM takes a different approach: LazyPRM initially assumes all paths are feasible. When a query is made on a LazyPRM road map, the algorithm first finds the shortest path, then checks that path for feasibility. If the path is not feasible, the offending vertices and edges are removed, and the algorithm makes another attempt. If no feasible path can be found, the algorithm either reports failure or adds more vertices to the underlying map and restarts the process ([2]).

More recently, Akbaripour and Masehian introduced Semi-Lazy PRM, in which feasibility of a random number of paths $m$ is checked during the roadmap construction phase [1].

The Open Motion Planning Library (OMPL) includes implementations of PRM*, LazyPRM, and LazyPRM*. LazyPRM* is similar to LazyPRM, but uses the “star” strategy of determining the number of nodes to connect to based on the total number and density of existing nodes. [10].

In addition to changing how a PRM is constructed, work has been done on making the use of PRMs more efficient.

Paden, Nager, and Frazzoli investigate reducing the run-time of individual path queries by storing a small number of minimum spanning trees for a given set of vertices and using those trees to supplement path queries [8].
Figure 1: The multiD algorithm forces each node to connect to others in multiple directions, ideally creating a graph such as the one on the right when a basic k-nearest or radius-based algorithm might create the graph on the left.

Dobson and Bekris take a different approach to making PRMs more efficient—they present an algorithm that accepts a dense road map as input and outputs a significantly less dense road map that provides equal or nearly-equal coverage. Their approach can be combined with multiple methods for creating the original PRM [3].

Moving further away from the basic PRM algorithm are approaches that construct vision-based maps, in which a region is subdivided into areas with clear visibility which are then connected [11, 9]. Note that [11] gives a deterministic rather than probabilistic approach.

2 The Algorithm k-nearest-multiD

2.1 Concept

The algorithm k-nearest-multiD attempts to increase the overall connectedness on a PRM while reducing the number of redundant connections by seeking to connect each vertex to others in multiple directions. Given a set \( V \) of vertices in the obstacle-free space of a given map, a value for kMax, and a set of directions \( D \) that partition the circle centered at some vertex \( v \) into equal segments, the algorithm attempts to connect each vertex \( v \) to the nearest \( \frac{k\text{Max}}{|D|} \) additional vertices in each direction.

In practice, the simplest way to specify \( D \) is to specify a number and have the algorithm compute appropriate boundaries for each direction.

Algorithm 1 k-nearest-multiD

\[
V \leftarrow \text{set of random points in FreeSpace} \\
D \leftarrow \text{set of directions} \\
k \leftarrow \lfloor \frac{k\text{Max}}{|D|} \rfloor \\
\text{for all } v \in V \text{ do} \\
\quad \text{for all } d \in D \text{ do} \\
\quad \\
\quad \quad U \leftarrow \text{kNearest}(v \in V, d, k) \\
\quad \quad \text{for all } u \in U \text{ do} \\
\quad \quad \\
\quad \quad \\
\quad \quad \quad \text{if } \text{CollisionFree}(v, u) \text{ then} \\
\quad \quad \quad \quad \text{add } (u, v) \text{ to edge set} \\
\quad \quad \quad \end{if} \\
\quad \quad \end{for} \\
\quad \end{for} \\
\end{for}
\]

Figure 1 gives a conceptual illustration of the multiD algorithm.
2.2 Analytic Properties

Lemma 1. The algorithm $k$-nearest-multiD runs in $O(dn \log(n))$ time. For all practical applications, this amounts to $O(n \log(n))$._proof_. Karaman and Frazzoli show that both the $k$-nearest-PRM and $k$-nearest-PRM* algorithms run in $O(n \log(n))$ time. Our result follows from 1) the only difference between multiD and the traditional kNearest algorithm being having to check for connections in $d$ directions separately and 2) the presumption that $d$ is a constant and small relative to $n$.

Lemma 2. Given graphs $G$ and $G'$ that share a vertex set $V$ and have edge sets $E$ and $E'$, respectively, where $E$ is constructed from $V$ using the $k$-nearest algorithm with $kMax = k$ for some positive integer $k$ and $E'$ is constructed using the $k$-nearest-multiD algorithm using the same $kMax = k'$ for some positive integer $k'$ and a given number of directions $d$, $E$ is a subset of $E'$ so long as

$$\left\lfloor \frac{k'}{d} \right\rfloor \geq k.$$

Proof. Let $E_v$ be the subset of $E$ of vertices that originate from $v \in V$, and let $E'_v$ be the subset of $E'$ of vertices that originate from $v \in V$.

In $k$-nearest-multiD, $\left\lfloor \frac{k'}{d} \right\rfloor$ gives the number of vertices in each direction to which some $v \in V$ will attempt to connect.

Consider first the case where for some vertex $v \in V$, the set of $k$ nearest vertices $U$ are all in the same direction. In this case, $E_v \subseteq E'_v$ so long as $\left\lfloor \frac{k'}{d} \right\rfloor \geq k$ (if $\left\lfloor \frac{k'}{d} \right\rfloor = k$, $E_v$ and $E'_v$ will be identical).

Consider second the case where the $k$ nearest vertices $U$ to $v$ lie in multiple directions. Then in each direction from $v$, there must be fewer than $k$ vertices connected to $v$ via edges in $E_v$. Since $v$ attempts connections to the $\left\lfloor \frac{k'}{d} \right\rfloor$ nearest vertices in each direction in $k$-nearest-multiD, $E_v \subseteq E'_v$ so long as $\left\lfloor \frac{k'}{d} \right\rfloor \geq k$.

The above holds for all $v \in V$. The result follows.

Lemma 3. The algorithm $k$-nearestMultiD is probabilistically complete so long as $\left\lfloor \frac{k}{d} \right\rfloor \geq 2$. Proof. Karaman and Frazzoli prove that the $k$-nearest algorithm (which they refer to as $k$-sPRM) is probabilistically complete for all $k \geq 2$ [4]. The result follows from Lemma 2; for all $(k, d)$ such that $\left\lfloor \frac{k}{d} \right\rfloor \geq 2$, the edge set of a PRM constructed according to $k$-nearest-multiD contains the edge set of a probabilistically complete PRM.

2.3 Algorithm Implementation

As part of this research, the algorithms $k$-nearest and $k$-nearest-multiD were implemented in MATLAB, building on MATLAB’s existing robotics.PRM class, which constructs a PRM by attempting connections to other nodes within a given radius (the sPRM algorithm in [4]).

While MATLAB does not offer the same number of options as the Open Motion Planning Library does for PRM construction and analysis, it has the advantage of being easy to install, modify, and use, both in terms of actually creating various PRMs and of having data visualization and analysis tools to help interpret results.

3 Experiments

The algorithms $k$-nearest and $k$-nearest-multiD were tested for their ability to construct PRMs that included a connected, navigable path between two points in a given map given a variety of vertex sets $V$ and $kMax$ values. For the multiD algorithm, four directions were used in all trials. We use the term algorithm variant to mean one of the two algorithms with a given $kMax$ value.

For $kMax$ values, we used 8, 16, and $k^* = 2e \log(n)$. The use of the last value means we are testing Karaman and Frazzoli’s $k$-nearest-PRM* in [4] against its multiD variant. Given the node set sizes and $kMax$ values we tested, $k^*$ is always the largest $kMax$ value. Note that our references to traditional $k$-nearest algorithms include $k$-nearest-PRM*; we are using traditional to differentiate from multiD.

Let $V_{n, r, m}$ be set of $n$ vertices randomly placed in the free space of a given map $m$ constructed with some random seed $r$. Let $R$ be the set of all random seeds used in a given map and $N$ the set of all vertex set sizes used in a given map. When not needed for disambiguation, $m$ is omitted from the subscript. For each map, $|N| \times |R|$ such vertex sets were created, using each vertex set size and each random seed once. Random seeds were recorded so that particular maps may be easily recreated if needed.

Each algorithm variant in consideration was used to construct a PRM for every given vertex set. The following results were collected for each algorithm variant:

1. Whether the PRM give a navigable route between two given points on the map, for each $n$ and $r$. The given start and end locations are shown in figures 2 for ComplexMap and 3 for circleMap.
2. Over all $r$ values, the proportion of PRMs that give a navigable route between two given points for each $n$ value.
Figure 2: A PRM constructed on MATLAB’s ComplexMap (obstacles inflated by 1 meter) with 500 nodes and kMax=12, constructed using the k-nearest-multiD algorithm. The figure shows a computed path from (3, 0) to (49, 38).

Figure 3: A PRM constructed on CircleMap with N nodes and kMax=12, constructed using the k-nearest-multiD algorithm. The figure shows a computed path from (4, 3) to (48, 44).

3. If a navigable route exists, its euclidean length, for each $n$ and $r$ value.
4. Over all $r$ values, the average navigable route length for each $n$ value.
5. How many edges each PRM contains for each $n$ and $r$ value.

The algorithm variants were then evaluated in terms of completion percentage, path length, and number of edges as a function of $n$, and of completion and path length as a function of the number of edges.

MATLAB’s ComplexMap The first full set of experiments were carried out using MATLAB’s included ComplexMap occupancy grid, with all obstacles inflated by 1 meter from their default size. An example of this map is shown in figure 2. Preliminary experiments showed that PRMs constructed on another MATLAB occupancy grid, SimpleMap, or on a non-inflated version of ComplexMap, produced a high enough level of connectivity for all algorithm variants that it was not possible to make useful comparisons between different algorithm variants. Node set sizes ranged from 50 to 1000 inclusive, in intervals of 50.
Figure 4: Edges in PRMs of ComplexMap as function of Nodes, in multiple algorithm variants. Note the separation between data sets and that for ever given $k$ value, the multiD algorithm has considerably fewer edges.

CircleMap A second set of experiments was carried out using a map created for this research, CircleMap. An example of CircleMap is shown in figure 3. CircleMap was created in the hope that the multiD algorithms would perform well on it compared to the standard k-nearest Map. CircleMap consist of multiple large elliptical regions connected by narrow corridors. Traditional PRM methods are known to be weak in finding viable paths through narrow passageways [2]. When nodes are distributed in CircleMap, there will naturally be many nodes in the circles and few in the corridors. If connections between nodes are made without regard to what direction they are in, they tendency should be for nodes within each circle to be connected to each other with many redundancies, but to be unconnected to slightly further away from nodes in corridors. Ideally, the multiD algorithm would both cut down on redundant connections within circles and encourage connections between nodes in circles and nodes in corridors.

Note that despite their names, CircleMap is considerably more complex than ComplexMap. Node set sizes used with CircleMap ranged from 500 to 1000, inclusive, in intervals of 50. We briefly tested node set sizes below 500, but found that there frequency of connection was too low to produce useful data.

4 Experimental Results

4.1 Overview

Our results are shown primarily in a series of graphs. In each graph, the datasets associated with a multiD algorithm are marked with a diamond, while the datasets associated with the standard k-nearest algorithm are marked with a circle, for scatter plots, or unmarked, for line plots. Colors assigned to each algorithm variant are consistent across all graphs. We designate our algorithms by giving the kMax value, which may be $k^*$, and the variable multi, which is true for multiD variants and false otherwise. All graphs depict data from the same sets.

4.2 Edge Count as a function of Nodes

Our first, and perhaps most dramatic, result is that the number of edges in a PRM correlate very strongly both with the number of nodes in the PRM and the exact algorithm used to create the PRM. While we did expect that the multiD algorithms would create fewer edges for a given number of nodes than the standard k-nearest algorithm using the same $kMax$ value, we were surprised by both how clear the distinction was between the two algorithms was for each kMax value and also by how clear the distinction was between PRMs using different kMax values.

Plots of Edge Count as a function of Node Count are shown in figures 4 and 5. In both plots, 100 different PRMs were created for each node count of each algorithm variant.

We have not seen any works that give an in-depth analysis of Edge Count as a function of Node Count. Many works focus primarily on the number of nodes, treating edges almost as an afterthought. That is unfortunate, given that when queries are...
4.3 Pathfinding Performance and Number of Nodes

Completion Rates When viewed as a function of vertex size, the path completion rate for PRMs of ComplexMap constructed according to the k-nearest and multiD algorithms were very similar given the same kMax values. Results are shown in figure 6.

For CircleMap, however, the multiD algorithm variants showed noticeably better performance, though for larger kMax values the multiD and traditional algorithms showed more similar performance. Results are in figure 7.

Path Lengths When path length is viewed as a function of vertex set size, the multiD algorithms showed slightly poorer performance (longer paths) than the traditional algorithms on both maps, with the difference being more pronounced on CircleMap. Also on both maps, the difference between multiD and traditional algorithms was considerably greater for larger
kMax values, with the difference more obvious on CircleMap. Results are shown in figures 8 and 9.

However, path length may not be as good a measure of performance as completion rate, since path length only accounts for PRMs on which a completed path is found. The longer average path lengths on the multiD algorithms could be caused by long paths found with the multiD algorithms that correspond to no path found in the traditional algorithms, given that the multiD algorithms generally have a higher completion rate.

### 4.4 Pathfinding Performance and Number of Edges

So far, we have seen that, comparing the multiD and traditional algorithms, multiD yields fewer edges for a given number of nodes and a higher completion rate, which is desirable, but also longer path lengths, which is not desirable. The obvious next question is how pathfinding performs on the multiD algorithm as a function of the number of edges.

**Completion Rates** Not surprisingly, the multiD algorithms do perform better in terms of completion rate than their traditional counterparts when compared by edge count. However, there are still surprises to be had. Results are shown in figures 10 and ??.

In ComplexMap, the algorithm variants with low $k_{Max}$ values outperform those with higher kMax values; $k^*$ (not the multiD version) has the poorest performance in terms of number of edges of all the variants tested. This result seems to imply that for the algorithm variants with large kMax values, many of the edges are redundant and do not add much connectivity to the map. This theory is supported by the multiD variants persistently performing better in terms of edge count, as mentioned above.

While CircleMap shows the same tendency as ComplexMap for the multiD algorithms to outperform ComplexMap, larger kMax values in CircleMap do outperform the smaller values. Part of this difference in results may be due to data for CircleMap starting with 500 nodes, rather than 50, but this seems unlikely since smaller kMax values outperform larger ones on
ComplexMap even as the completion rate reaches 100%. However, we believe a more likely cause is the closed structure of CircleMap, which causes nodes that are nearby in terms of distance to be unreachable. With a small kMax value, the closest k nodes in CircleMap to a given node are likely to be unreachable; since the reachable nodes will be further away, a larger kMax value is required to get a substantial number of connections.

Path Length When viewing path length as a function of edge count, the multiD algorithms do slightly better on ComplexMap. Notice that on ComplexMap, the algorithms with smaller kMax values are sometimes better.

On CircleMap, all four variants of k=8 and k=16 have similar performance. The two k* variants show a great improvement over the smaller kMax values, but have little difference between them.

4.5 Summary of Results

Figure 14 shows a summary of how using the multiD algorithm affects performance, according to our results. We found that the multiD algorithm does reduce the number of edges for a given number of nodes. In terms of algorithm performance, the multiD algorithm improves the likelihood that a usable path will be found between two points, but results on the ability of the multiD algorithm to find a shorter path than the traditional k-nearest algorithm were mixed.
Figure 12: Path length as a function of edges. Not shown for readability reasons are path lengths approaching 110 meters for edge counts under 200.

Figure 13: Path length as a function of edges in CircleMap.
5 Conclusion and Future Research Directions

We can say with a very high level of confidence that the multiD algorithm reduces the number of edges created in a PRM, compared to using a traditional k-nearest algorithm. Furthermore, since using multiD did not reduce pathfinding performance (with the exception of path length on ComplexMap), we feel confident in saying that the multiD algorithm consistently reduces the number of low-quality edges and sometimes increases the number of high-quality edges.

Compared to the basic algorithm, multiD showed a greater improvement on the obstacle dense CircleMap than it did on the relatively open ComplexMap. We suspect this result will be true in general; multiD will have performance similar to kNearest on open maps, but will show improvement in obstacle dense maps.

Unfortunately, the performance gains we have seen with multiD are small compared to those provided by the SPARS algorithm given in [3]. Even though multiD shows improvements compared to k-nearest algorithms, improvements which would likely be greater in more complex configuration spaces, we do not think the multiD algorithm would be competitive with the SPARS algorithm.

5.1 Future Research

We first discuss general ideas for future research, and then conclude with a specific list of ideas that might be feasible within a relatively short time period.

MultiD in more directions In our experiments, we only tested the multiD algorithm with four directions. I would like to test it in many directions. My intuition is that increasing the number of directions while holding kMax constant would cause algorithm effectiveness to initially increase, then continually decrease.

Standardized measures of configuration spaces and performance In conducting this research, the single greatest deficiency we found in the existing literature was a lack of means by which to classify spaces to be mapped. A given space may be referred to as obstacle-dense or open, but that statement gives no location about the positions of obstacles; if a given space has a specific set of obstacles, pathfinding performance will likely be very different depending on where exactly those obstacles are placed. In our opinion, any research that could develop a system to classify configuration spaces such that PRM algorithms could be expected to give similar performance on spaces with similar classifications would revolutionize the field. Unfortunately, such a system seems likely to be extremely difficult to develop.

A related, more manageable goal would be to perform a comprehensive survey of existing algorithms on existing benchmarks. While OMPL does provide a comprehensive benchmark suite for PRMs [10], there seem to be a lack of papers providing a broad survey of the benchmark performance of different algorithms.

MultiD algorithms on more difficult maps While our data do seem to suggest that multiD gives a larger advantage configuration spaces in which it is more difficult to find a clear path, that suggestion is based on only two maps, of which even the obstacle-dense CircleMap was relatively simple. Papers in the literature frequently conduct experiments involving tens or hundreds of thousands of nodes, whereas the largest node set we used was 1,000. Given more time, we would like to run similar experiments to those we have already done on more difficult maps with much larger node sizes.

Doing so would require a change in experimental setup, most likely to OMPL. MATLAB is not a fast language, and the PRM program we used within MATLAB was not very efficient, making larger scale experiments impractical. Larger scale experiments would, though, let us draw much stronger conclusions.

Correlation between node count and edge count Our data showed a very high correlation, for each algorithm and each map variant tested, between number of nodes and number of edges. How consistent is this relationship across different maps
and with different algorithm families? The question could be investigated both analytically as a graph theory problem and experimentally, running many algorithms over many different maps and analyzing the results.

**Statistical Analysis** One thing lacking from this paper is a statistical analysis of the results. Given more time, I would like to formally analyze the data I collected for significance, rather than simply eyeballing it. A statistical analysis would also benefit from operating on OMPL, since its more efficient operation would allow for collection of larger data sets.

**To-Do List**

Given two months’ more time, these are the goals I would like to accomplish, in priority order.

- Implement multiD algorithm with for any number of directions, rather than merely four.
- Run experiments on more difficult maps requiring much larger node sets for success.
- Provide a formal statistical analysis of results.
- Study the relation between node count and edge count in more detail, both analytically and experimentally.
- Give an in-depth comparison of multiD and other current algorithms using a standard set of benchmarks, such as OMPL.

**References**


