Augmenting Planning Graphs in 2-Dimensional Dynamic Environments With Obstacle Scaffolds

Spencer Lane, Kyle Vedder, Joydeep Biswas
University of Massachusetts Amherst
140 Governors Dr.
Amherst, MA 01003
slane@cs.umass.edu, kvedder@umass.edu, joydeepb@cs.umass.edu

Abstract

In this paper, we present an extension to roadmap based path planners that allows for finer control over motions near dynamic obstacles. We utilize the fact that we know the shape of the dynamic obstacles offline and the location of the obstacles online. We supplement the roadmap graph by adding predefined graphs around obstacles, known as scaffold graphs. These graphs are inserted at query time and updated as the obstacles move throughout the environment. We performed a preliminary evaluation our approach in the RoboCup SSL domain and show similar average case performance and improved performance in the case where there are obstacles between the start and the goal.

Introduction

The ability to find a collision free path between two points is required for the vast majority of tasks in robotics, regardless of the hardware that those tasks are run on. There have been a variety of solutions to this problem that have been developed over the years, but we have chosen to focus on domains that have strict timing constraints, require high quality paths, and where the shape of the dynamic obstacles is known offline and the locations of the dynamic obstacles are known online. This kind of domain imposes constraints on the kinds of solutions that can be used. The RoboCup Small Size League (SSL) is a good example of a domain with these properties.

RoboCup SSL is a RoboCup domain that focuses on multi-agent planning and coordination. Teams consist of 6 robots that use a golf ball as a soccer ball. The game state evolves rapidly from one moment to the next, so a plan generated in the previous time step might not be valid. It is also desirable to update the robot control commands at the same rate new data is received; thus all processing and planning should be performed within 16 milliseconds. Finally, because RoboCup is a competitive domain, it is important that we produce high quality paths within that time limit.

In order to improve path planning in domains with these properties, we propose a method of supplementing roadmap based path planners by constructing roadmaps, called scaffolds, around our obstacles offline and inserting them into the graph when updating the position of those obstacles. We build on the framework introduced by Leven and Hutchinson to apply a roadmap path planner in a dynamic environment (Leven and Hutchinson 2002). Path planning is split into two steps: offline and online. In the offline step, a roadmap of the static environment is constructed. During the online step, the scaffolds around the moving obstacles are inserted and the edges blocked by obstacles are invalidated.

In order to generate the initial roadmap, we utilize an existing sampling based path planner such as Probabilistic Road Map (PRM) (Kavraki et al. 1996), sPRM (Kavraki, Kolountzakis, and Latombe 1998), or PRM* (Karaman and Frazzoli 2011).

We begin with a brief review of the related literature. We then discuss the specific modifications to PRM that we introduce in this work. We discuss the formalisms for PRM graphs and scaffold graphs, review the primitive functions used in the construction of PRM graphs, and finally introduce a set of algorithms used for integrating scaffold graphs with existing roadmaps. Finally, we evaluate the performance of our scaffold approach in a variety of situations.

Related Work

PRM was introduced in 1996 as a method of path planning for robots working in static workspaces (Kavraki et al. 1996). In 1998, Kavraki, Kolountzakis, and Latombe introduced sPRM, which was shown to be asymptotically optimal (Kavraki, Kolountzakis, and Latombe 1998). In 2011, PRM* was introduced, which created an asymptotically optimal version of PRM with a reduced query time (Karaman and Frazzoli 2011).

Most PRM variant algorithms assume that the workspace is static; however, a number of approaches exist for applying PRM in a dynamic environment. An early framework for doing so was proposed by Leven and Hutchinson (Leven and Hutchinson 2002). The key insight is to partition of the workspace into the static environment and the dynamic environment. Another approach to using PRM in dynamic environments is to use an RRT-like local planner to reconnect edges that have become invalid due to obstacles (Jaillet and Siméon 2004). van den Berg et al. place assumptions about the motion of obstacles in the environment, restricting them to a fixed set of configurations known a priori (van den Berg et al. 2005).

Within the RoboCup domain, it is common to use the execution extended RRT (ERRT) algorithm (Bruce and Veloso 2002). This extends RRTs to allow for re-planning in many

situations. ERRT is used due to its fast planning, replanning and query times. Other approaches are typically not used as they often take too long to converge to a solution.

PRM and Scaffold Definitions

In this section, we define the formalisms used in our approach. First, we review the definitions of graphs used in PRM and introduce the definition of a scaffold graph. Then, we discuss the primitive functions used in the construction of PRM graphs and supplement that with a ray cast primitive. Finally, we discuss the scaffold structure for circular obstacles and n-sided polygons.

Planning Graph

We start by defining a PRM Graph G as a set of edges E and vertices V. Each edge consists of a tuple $e = \langle v_1, v_2, a \rangle$ where $v_1, v_2 \in V$ and a is a boolean that indicates if the edge is active. For brevity of notation, a_e references that boolean for edge e. The default value of a is `True`, and a needs to be updated whenever the graph is queried as it depends on the positions of the moving obstacles.

A scaffold graph S_i is defined in a manner similar to a PRM graph with a set of edges E_i and vertices V_i. A particular scaffold S_i is associated with a particular obstacle $o_i \in O$ where O denotes the set of all obstacles in the environment. The positions of the vertices are defined relative to the obstacle. A transform T_i is defined that maps points from the obstacle reference frame to the planning frame.

This transform must consist solely of rotation and translation. The translation must be defined relative to the center of the obstacle. We define c_i to be the center of o_i

We also define a radius r_i which defines the bounding circle around the obstacle. For the sake of convenience, we refer to the kth point of the jth layer of scaffold i as $v_{i,j,k}$.

Each scaffold consists of a number of layers with the layer closest to the obstacle being defined as layer 1 and the outermost layer being defined as layer N, where N is the number of scaffold layers. Scaffolds can be a single layer. We define a subset of the full scaffold graph $S_{i,j}$ to be the the set of edges and vertices on the jth level of scaffold i. We also define the number of points per scaffold layer to be m.

A particular scaffold layer $S_{i,j}$ defines a convex polygon that encloses the associated obstacle at a fixed distance $d_{i,j}$. Each point within that layer must be $d_{i,j}$ away from the obstacle.

Scaffolds From Geometric Primitives

Circle In order to define a scaffold around a circle, we begin by defining the distance from the obstacle that the lowest layer occupies. Because the points on a scaffold layer are equidistant from the center point of the circle, the closest point to circle is the mid-point of the line segment defined by two points. If we set the distance between the mid-point and the center point of the obstacle to be the radius of the obstacle, we can derive an equation for the radius of the lowest layer:

$$d_{i,1} = \frac{2r_i}{(2cos\left(\frac{2\pi}{m}\right) + 2)^{\frac{1}{2}}} - r_i \tag{1}$$

We can then define the ith point of the jth layer as:

$$v_{i,j,k} = \begin{bmatrix} (d_{i,j} + r_i) * cos(\theta_{i,j,k}) \\ (d_{i,j} + r_i) * sin(\theta_{i,j,k}) \end{bmatrix}$$
$$\theta_{i,j,k} = (k-1) * \frac{2\pi}{m} + ((j-1) \bmod 2) * \frac{2\pi}{2m} \tag{2}$$

The second term of the angle computation is used to give each point a slight offset from those on the previous layer. We can then define the set of edges to and from a given point. We use the $\rightleftharpoons$ operator to show that an edge exists. We connect the points as follows:

When j is even:

$$v_{i,j,k} \rightleftharpoons \{v_{i,j-1,k}, v_{i,j+1,k}, v_{i,j-1,k+1}, v_{i,j+1,k+1}, v_{i,j,k+1}, v_{i,j,k-1}\} \tag{3}$$

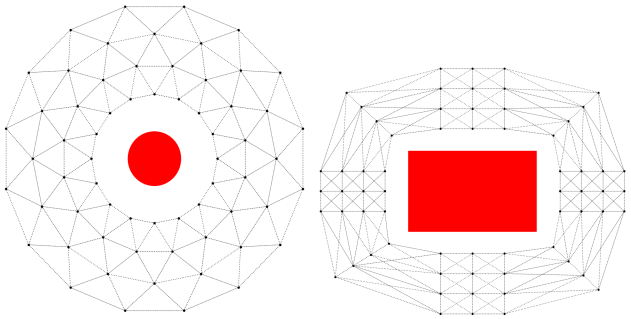
When j is odd:

$$v_{i,j,k} \rightleftharpoons \{v_{i,j-1,k}, v_{i,j+1,k}, v_{i,j,k+1}, v_{i,j+1,k-1}, v_{i,j,k-1}, v_{i,j-1,k-1}\} \tag{4}$$

In all cases:

$$v_{i,j,0} = v_{i,j,m}, v_{i,j,m+1} = v_{i,j,1} \tag{5}$$

Edges to vertices where $j = 0$ and $j = N + 1$ are obviously ignored as those vertices do not exist. A completed circle scaffold is shown in Figure 1a. Note that the circular obstacle is inflated such that a robot can sit on any of the scaffold points without colliding with the obstacle.



(a) Circular Scaffold (b) Rectangular Scaffold

Figure 1: Example scaffolds for geometric shapes. The number of layers is fixed to be 4 and the number of points per layer is fixed to be 16.

Convex Polygon In order to define a scaffold around a convex polygon, we first define the number of sides to be s. We impose the constraint that the number of points per scaffold layer, m, must be a multiple of s. We define the corner points of the polygon to be $p_1, p_2, \ldots p_s$. The corners of the polygon edge i are defined to be p_i and p_{i+1} with the corners of edge s being p_s and p_1. We also define the unit normals of the various sides and corners with $\hat{n}_i$ representing the normal of side i and $\hat{n}_{p_i}$ representing the normal vector at corner p_i.

In order to space the points evenly throughout the scaffold, we divide the points evenly between the sides of the polygon. There are $m/s - s$ scaffold vertices associated with

each side and s vertices associated with the corners. For example, a rectangle with 16 points per layer would have one point on each layer associated with each corner and 3 points per layer associated with each side.

If we associate $v_{i,j,1}$ with p_1, vertices are associated with corner points when $s(k-1)modm = 0$ and with sides otherwise. In order to define the position of the vertices associated with the corners or side, we first need to derive an expression for determining which side a vertex is associated with based on its index k. We will define the index of the side as κ. The corner points associated with side κ are then p_κ and $p_{\kappa+1}$. The value of κ is defined as follows:

$$\kappa = \lfloor (k-1) * \frac{s}{m} \rfloor mods + 1 \tag{6}$$

Where $\lfloor\rfloor$ is the floor operator. The vertices associated with a corner points have their positions defined as follows:

$$v_{i,j,k} = p_\kappa + d_{i,j}\hat{n}_{p_\kappa} \tag{7}$$

Where d_i is the distance of that layer from the obstacles and the $d_0 = 0$. We also define the position of the points associated with each side as follows:

$$\begin{aligned} \lambda_{i,j,k} &= \frac{k-1-m(j-1)-\frac{m}{s}(\kappa-1)}{m/s} \\ v_{i,j,k} &= \lambda_{i,j,k}||p_{\kappa+1} - p_\kappa|| + p_\kappa + d_i\hat{n}_j \end{aligned} \tag{8}$$

Where $\lambda_{i,j,k}$ represents the fraction along the line that the vertex covers. Using this notation, we can define set set of edges to and from a given scaffold vertex. We connect each vertex to the adjacent vertices.

$$\begin{aligned} v_{i,j,k} \rightleftharpoons \{&v_{i,j-1,k}, v_{i,j+1,k}, v_{i,j+1,k-1}, v_{i,j+1,k+1}, \\ &v_{i,j,k-1}, v_{i,j,k+1}, v_{i,j-1,k-1}, v_{i,j-1,k+1}\} \end{aligned} \tag{9}$$

As before, in all cases:

$$v_{i,j,0} = v_{i,j,m}, v_{i,j,m+1} = v_{i,j,1} \tag{10}$$

Again edges to vertices where $j = 0$ and $j = N + 1$ are ignored. A completed scaffold for a rectangle is shown in Figure 1b. Note that the obstacle is inflated such that a robot can sit on any of the scaffold points without colliding with the obstacle.

Navigation and Planning with Scaffolds

In this section, we examine how to incorporate scaffold graphs into path planning algorithms. As with other PRM approaches designed to be used in dynamic environments, we partition the obstacles into two sets, static obstacles and dynamic obstacles. The static obstacles, those that have a fixed location, are the only ones that are taken into account when generating the initial roadmap. Up until this point, we have only discussed using this approach with PRM. In practice, the scaffolds can be added to any roadmap based graph. This initial graph could be generated using a PRM variant or it could be generated using a different roadmap approach such as a fixed grid or a Vornoi decomposition. Because the method used to generate the roadmap does not affect the application of the roadmap, we instead focus on defining an algorithm for updating the graph with the scaffold points. In order to define an initial approach to combining the graphs, we first review the primitive functions used in RRT and PRM and we then define a naïve graph update algorithm.

Primitive Functions

The existing PRM and RRT approaches rely on several primitive functions. These are sampling, nearest neighbor, near vertices, steering, and a collision test. These are well described in the literature but we will review the two that we use in the scaffolding approach.

Near Vertices: The `Near`(G, p, r) method returns the set of vertices that are within a given distance of a particular point. It takes as input a graph G, a point in space p, and a positive real number r. It returns the set of all vertices $v \in V$ that are within r of the specified point p.

Collision Test: The `CollisionFree`(p_1, p_2) method returns a boolean that indicates if the line segment between p_1 and p_2 does not collide with any obstacles.

Graph Update

We separate the graph update into two steps. First, the scaffold points and edges are added to the static graph, then any edges that are blocked are invalidated.

The scaffold points are inserted into the graph in the same way that new points are added when generating a roadmap using sPRM. Each point is connected to all of the other points within a certain range of it as long as the line between the two points does not collide with an obstacle. When inserting the scaffolds, we ignore the other points within the scaffold as there are already edges defined between these points. Algorithm 1 presents the procedure for inserting a scaffold into the static graph.

Algorithm 1 SCAFFOLD INSERTION

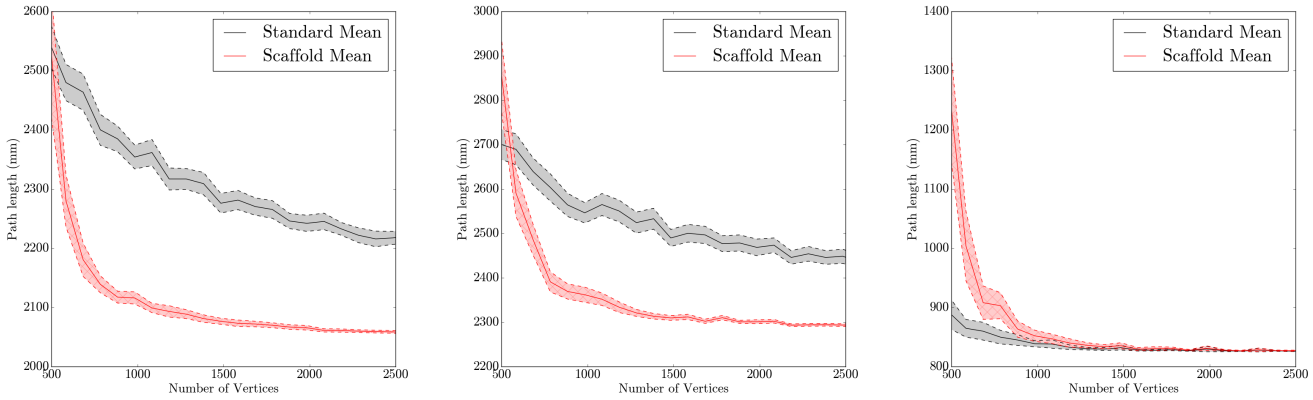
```
1:  procedure INSERTSCAFFOLD(G,S_i)
2:      Input: Roadmap Graph G and scaffold graphs S_i
3:      Output: Updated graph G that contains the scaffold
        points and edges
4:      for all v ∈ V_i do
5:          Add v to V
6:          neighbors ← Near(G, v, Max Edge Length)
7:          for all v_near ∈ neighbors do
8:              if CollisionFree(v, v_near) then
9:                  Add edge from v to v_near
10:     for all e ∈ E_i do
11:         Add e to E
```

Finally, we define the method that invalidates the blocked edges. We perform this invalidation by setting the a indicator for edges that are blocked by dynamic obstacles. As mentioned above, an edge is defined by a tuple $e = \langle v_1, v_2, a\rangle$. We use $v_{1,e}$, $v_{2,e}$ and a_e to refer to the tuple values for a particular edge e. In order to update the graph, we check each edge to see if it collides with an obstacle. This is done by calling the `CollisionFree` primitive for each edge.

Note that this approach can be further optimized and is part of our ongoing research.

Experimental Methodology

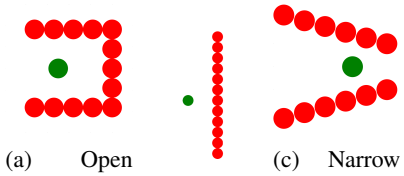
In order to evaluate our approach, we compare it against an existing PRM approach, specifically sPRM. We evaluate



(a) Path Quality, Open Box (b) Path Quality, Wall (c) Path Quality, Narrow Passage

Figure 2: Path quality over the number of vertices in the base graph for the fixed configurations

performance in a number of specific obstacle configurations as well as random field configurations. The specific configurations were designed to showcase the performance of our approach as compared to the standard algorithms. These configurations consist of a partial box, a narrow opening and a wall. While these configurations are not going to occur during the course of a RoboCup match, they are used to showcase the behavior of the scaffolds in traditionally challenging configurations. In each case, we evaluated each algorithm with randomly generated goal locations. The fixed configurations are shown in Figure 3.



(a) Open Box (b) Wall (c) Narrow Passageway

Figure 3: Obstacle layouts. The start point is shown in green. The obstacles are shown in red.

In order to directly compare scaffolding and sPRM, we create a base random graph of varying granularity ranging from 100 vertices to 2500 vertices. This graph is used for both approaches. We then add a number of vertices to the sPRM graph such that it has the same number of vertices as the base graph with the scaffolds added. We then compare the mean path lengths for both cases. A particular trial is only included in the mean calculation if solutions are found for both the scaffold PRM and sPRM graphs. We also compare the success rate as a function of the number of vertices and the ratio of the run times. For the fixed configurations we ran 100 trials at each level of granularity.

Preliminary Results and Conclusions

Figure 2 shows the path quality over the number of vertices for each of our configurations. The dotted lines and shaded areas represent 95% confidence intervals. In all three cases, the variance and mean starts significantly higher for the scaffolding approach than for sPRM but decreases as the number of vertices increases. In the case of the open box and wall configurations, the found solutions for graphs with more than 700 vertices are significantly better for the scaffold approach. In the case of the narrow pathway configuration, the scaffolding approach performs essentially the same as the sPRM approach. We believe these results can be attributed to the fact that the optimal path for the open box and wall configurations involves traversing obstacles whereas the optimal path for the narrow passageway that we constructed does not.

In addition to examining the path quality, we tabulated the amount of time it took to update the roadmap and find a path from the start to the goal. Over all of the cases, the update time with the scaffold took an average of 1.72 times as long as without. We are actively working on a more computationally efficient insertion algorithm that we believe will reduce the graph update time.

We believe that these preliminary results demonstrate that the scaffold graph approach provides an advantage over simply using a PRM graph in dynamic environments with known dynamic obstacle shapes. If the update time can be reduced, the scaffold approach presented here should provide benefits in RoboCup SSL and other similar domains.

In the future, we would like to extend the scaffold definition to obstacle shapes that are not circles or n-sided convex polygons. In particular, we want to use the distance transform in order to define scaffolds around arbitrary shapes. In addition, we would like to extend the notion of scaffolds to higher dimensional spaces. In particular, we believe that this approach could assist with manipulation in cluttered environments and with multi-agent path planning in two dimensional spaces.

References

Bruce, J., and Veloso, M. 2002. Real-time randomized path planning for robot navigation. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, 2383–2388. IEEE.

Jaillet, L., and Siméon, T. 2004. A prm-based motion planner for dynamically changing environments. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 2, 1606–1611. IEEE.

Karaman, S., and Frazzoli, E. 2011. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research* 30(7):846–894.

Kavraki, L. E.; Svestka, P.; Latombe, J.-C.; and Overmars, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation* 12(4):566–580.

Kavraki, L. E.; Kolountzakis, M. N.; and Latombe, J.-C. 1998. Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation* 14(1):166–171.

Leven, P., and Hutchinson, S. 2002. A framework for real-time path planning in changing environments. *The International Journal of Robotics Research* 21(12):999–1030.

van den Berg, J. P.; Nieuwenhuisen, D.; Jaillet, L.; and Overmars, M. H. 2005. Creating robust roadmaps for motion planning in changing environments. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, 1053–1059. IEEE.