Faster, More Effective Connection for Probabilistic Roadmaps *

Lucia K. Dale dalel@cs.tamu.edu
Guang Song gsong@cs.tamu.edu
Nancy M. Amato amato@cs.tamu.edu

Technical Report 00-005
Department of Computer Science
Texas A&M University
January 16, 2000

Abstract

In this paper, we report on our experience attempting to improve the running times of probabilistic roadmap motion planning methods (PRMs). We show that significant speedups can be obtained with relatively little effort on the part of the developer by employing new connection strategies and more intelligent ways of invoking and utilizing existing off-the-shelf collision detection packages. We outline general techniques for determining when and which of the techniques we have developed are most useful. We also categorize each as being helpful in either a problem specific or problem independent way. Many techniques presented are of general usefulness.

*This research supported in part by NSF CAREER Award CCR-9624315 (with REU Supplement), NSF Grants IIS-9619850 (with REU Supplement), EIA-9805823, and EIA-9810937, by the Texas Higher Education Coordinating Board under grant ARP-036327-017, and by the NCSA at the University at Illinois at Urbana-Champaign. Dale is supported in part by a Department of Education GAANN fellowship.

1 Introduction

Motion planning has applications in many areas such as robotics, virtual reality systems, computer aided design, etc. Since there is strong evidence that a complete planner requires exponential time in the number of degrees of freedom (dof) of the robot [9, 12], considerable recent attention has focussed on probabilistic (randomized) roadmap methods (PRMs), which have shown promising signs for solving high dimensional problems [1, 2, 3, 5, 6, 8, 11, 15, 17].

However, even with PRMs prohibitively long running times often result. This paper presents methods and techniques useful for improving the running time of PRMs.

Because the connection phase of probabilistic roadmaps typically accounts for over 98% of the total running time, most of our suggestions are appropriately concentrated there. An important question to ask is *why is connection using such a huge percentage of the running time?* The answer is complicated. Part of the answer is often that numerous calls to collision detection are necessary during this stage of the roadmap construction. We present ways to more productively invoke collision checking. Part of the answer is often that the preliminary step of generation failed to generate nodes of a type crucial to the success of the connection step. We present new "connection" methods to address this lack, methods which in some sense cannot be categorized as belonging completely to either generation or connection steps. Finally, part of the answer is simply that smarter connection methods are needed. We present new connection phase methods. Many of the ideas presented have also been shown to have useful application to existing connection methods, as well as subtle improvements to node generation methods and local planners.

1.1 Probabilistic Roadmap Methods

Recently, a class of motion planning methods, known as *probabilistic roadmap methods* (PRMs), have gained much attention. Briefly, PRMs use randomization to construct a graph in C-space (a *roadmap* [12]). Roadmap nodes correspond to collision-free configurations of the robot. Two nodes are connected by an edge if a path between the two corresponding configurations can be found by a 'local planning' method. Queries are processed by connecting the initial and goal configurations to the roadmap, and then finding a path in the roadmap between these two connection points.

PRMs: PROBABILISTIC ROADMAP METHODS

I. PREPROCESSING: ROADMAP CONSTRUCTION
   1. NODE GENERATION (find collision-free configurations)
   2. CONNECTION (connect nodes to form roadmap)
      (repeat as desired)

II. QUERY PROCESSING
   1. CONNECT START/GOAL TO ROADMAP
   2. FIND ROADMAP PATH BETWEEN CONNECTION NODES

The roadmap construction time varies according to the problem, but can be quite significant (ranging from a few seconds to many hours). Note that the preprocessing time necessary to build adequate roadmaps for any particular problem is justified by the ability to satisfy quickly (typically within a few seconds) any number of later varied queries to the same roadmap.

The connection phase, where the heaviest costs of collision detection are incurred, is also naturally parallelizable [4]. However, efforts spent improving sequential approaches will have a tremendous impact on parallelization efforts as well.

1.2 Previous Work

Connection strategies in general fall into one of three broad categories.

One category is for connection to be completely decoupled from the generation step. With this approach, connection begins without any benefit of past knowledge. The input is a set of nodes and no new nodes are added during connection. Connection methods of this type tend to be brute force heuristics.

An example: BasicPRM [11] (without enhancement). Nodes are generated uniformly at random. Without additional knowledge, the most intelligent thing you can do for connection is to simply try to connect each node to it's k closest neighbors, for some constant k.

A second category tries to capitalize on information carried forward from the first step of generation to the second step of connection. The motivation is that information gathered when the nodes are generated may somehow be helpful in determining how or in what order connection between them is most logically attempted.

An example: OBPRM [5]. In Obstacle-Based PRM, nodes are generated for each obstacle. For connection, you try to connect each node to it's k closest neighbors on each obstacle.

A third category tries to identify, during connection, where and how new nodes might be added to the map that will help in connecting those already there. Often these types of methods are characterized by a *growth* paradigm. The goal is not to just randomly add more nodes but to try to *grow* isolated nodes or whole connected components towards one another in an attempt to increase the connectivity of the map in such a way as to more accurately reflect the topological connection of the freespace.

An example: RRT [14]. The Rapidly–exploring Random Tree starts with a seed node and grows out from it in randomly directed steps until the unexplored space is filled. In this extreme example, no node (other than the initial seed) is ever isolated.

For all of these categories, more and less successful efforts have been made to accommodate non-holonomic robots. This can either (both?) simplify or complicate local planning. With complicated robots (e.g, closed kinematic chains [15], flexible objects [10]), there are additional constraints on the allowable motion which must be satisfied. All of the methods discussed in this paper are relevant to such work, although extensive modifications may be necessary.

2 Experimental Setup

2.1 Code

Our OBPRM code is written in C++. It implements a 3D motion planning testbed for rigid bodies as well as serial linkages. Work is on–going for additional types of "robot" bodies. We have implemented (our interpretations of) various PRM generation and connection methods. Methods include Basic PRM [11], OBPRM [5], Gaussian Filter on Basic PRM [6], and RRT [14] variations.

We have installed three different collision detection libraries: C-Space Toolkit (CSTK) [18], V-Clip [16], and RAPID [7]. For some of our work, we needed a clearance value returned for non-convex bodies. Only CSTK provided this and so, for consistency, was always used in obtaining the results reported here.

The focus of our work is connection strategies, and therefore generation is only of interest as it applies to what the individual connection strategies are doing. We present compelling evidence that **if** the generated nodes are somehow **connectable**, then there are ways to utilize low–level (but sophisticated) collision detection libraries in conjunction with our methods to **then discover connections**.

2.2 Environments

Our experiments involved three environments: the *walls*, the *flange* and the *alpha puzzle*. The walls are several parallel surfaces with variously placed openings through which a small rectangular object must pass in succession (see Figure 2). The flange consists of a fixed rectangular part with a circular opening (the obstacle, 990 triangles) and a curved pipe (the robot, 5306 triangles) that must be inserted into the opening in the obstacle (see Figure 3). The alpha puzzle consists of two identical twisted, intertwined tubes (one the obstacle and one the robot, 1008 triangles each); the objective of the puzzle is to separate the tubes (see Figure 4).

2.3 Platform

All experiments were run on an SGI Octane.

3 Collision Detection Exploited

We divide our methods into categories of differing approach. The Class I techniques use existing smart collision detection libraries (CSTK) in clever ways. There is a large payoff in terms of efficiency and running time at this level.

3.1 Techniques

- **isSameCC**
  If two nodes are already in the same connected component do not try to connect them.

- **Clearance**
  Requires clearance distance returned by collision detection routines. It is redundant to check for collision within clearance distance.

  Of the three collision detection libraries we have installed in our motion planning code, RAPID does not return a clearance value and VClip only does so for convex objects. CSTK is the only one of the three which makes clearance values available for both convex and non–convex objects.

- **Resolution**
  Make order of collision checking be at an increasingly finer resolution and employ a binary search pattern of collision detection invocations rather than apply collision detection sequentially at the finest resolution.

- **LineSegmentApprox**
  Approximate complicated robots with a simpler object which would pierce the original robot's center of mass as it traveled the distance of a connection. If this gross approximation test of connectability fails, do not attempt a full resolution connection attempt. The node pairs which fail this check are deemed "seemingly not connectable".

The most primitive object upon which collision detection can be call is a triangle (three vertices) for both RAPID and CSTK. VClip needs four points to create a simplex.

3.2 Input

Input set of map nodes was generated by OBPRM using overlapping normals.

3.3 Results

Architecture: SGI Octane					
		Environments			
		sandwich		flange	
Strategy		CD#	time[s]	CD#	time[s]
isSameCC	off	80102	9.07	4614	159
	on	27375	3.48	2750	83.0
clearance	on	10320	2.0	2314	59.3
line Seg.	200	26826	3.47	2750	83.4
(clearance	20	15364	2.59	2779	83.0
off)	2	15364	2.66	3917	88.2

3.4 Discussion

From the experimental results, it is easy to see checking **ifSameCC** consistently improves the running time. This is quite reasonable since adding any extra edges to an already connected part will not help connectivity at all. However, there is small drawback in that the roadmap grown is actually a spanning tree. Therefore, the paths extracted from these roadmaps will likely be much longer and less smooth than those typically contained in a PRM roadmap. However, this problem can be dealt with later in a subsequent post-processing phase, which should be applied to the roadmap as a whole.

Checking the clearance and iteratively increasing the resolution worked very well too. When in open space, checking clearance can greatly reduce the number of collision detection calls since it enables the robot to make big steps (compared with a fixed resolution) based on clearance information. On the other hand, when the local environment is crowded, the other part of the method takes the lead, i.e. iteratively increasing resolution. Since in crowded environments it is quite likely that the path will have collisions, binary checking (instead of incremental) will often find collisions (if any) much more quickly: fixed resolution in such cases is likely overkill.

For the line segment method, there are trade-offs. One one side, line segment checking will find some unconnectable pairs very quickly and save on collision checking. On the other side, there is overhead in creating line segments on-the-fly. We applied this method only if the distance between the two nodes was larger than some threshold. The results in the table show that improvements were found with medium thresholds (e.g., around 20). This is reasonable since it should be in somewhere in the middle that the time consumed in creating line segments is compensated by the save on collision checking. Another observation is for flange environment, such method does not seem to affect the results much, this is because for flange environment, nodes are generated much denser in order to solve the problem. As a consequence of this, a node is closer to its neighbours, causing lineSegment checking unnecessary most time.

4 Simple Connection

The Class II techniques look at enhancing simple but powerful connection strategies. These are more the traditional connection strategies but with some interesting twists. No new nodes are generated or added. There is modest but measurable payoff at this level.

4.1 Techniques

- **Closest**
  This method is the "standard" connection method in PRMs. We use it as a baseline against which to compare the others in this class. For every node try to connect to the k closest neighbors. A neighbor is another node already in roadmap.

- **ClosestVE**
  For every node try to connect to the k closest neighbors. A neighbor is either another node already in roadmap or the shortest (Euclidean) distance to an "implicit" node on an edge where the edge is already in the roadmap.

- **ObstBased**
  For every node generated by/on each obstacle, try to connect it to the closest k_{other} nodes on another obstacle. For every node generated by/on each obstacle, try to connect it to the closest k_{self} nodes on the same obstacle.

4.2 Input

To compare the methods fairly against one another, it is necessary to use an obstacle based OBPRM node generation phase so the **ObstBased** technique can be fairly compared against the other two in this class. **ClosestVE** works with the straightline local planner. To allow comparisons between the methods, all were run using only straightline as the only local planner. The input node set was identical for all three techniques.

4.3 Results

Architecture: SGI Octane						
	Walls Environment ($n_1 = 50$)					
Strategy	#CD	sec	#E	#CC	bigCC	#iso
close 50	30394	0.5758	26	5	11	0
	Walls Environment ($n_2 = 1000$)					
close 5	52681	2.6351	259	47	61	23
closVE 5	78643	4.5540	558	20	258	15
obstB 5	53462	1.7506	242	64	58	32

4.4 Discussion

Closest for some k is a conservative connection method to use in the absence of any insight into a specific planning problem. Intuitively easy to understand, it is also relatively fast to run. In a difficult problem instance, however, it often fails and there's a need for smarter methods

ClosestVE necessarily incurs overhead for computing the closest point to line but when run head to head with the others ends up with fewer connected components.

This method is only applied to isolated nodes trying to connect to edges (and nodes they join) which exist before this method is called. For example, if in the first pass there are 10 edges made by connection then in the second pass 50 nodes are generated, the method attempts to connect each of the 50 nodes to the closest point on each of the pre–existing edges. The closest point may of course be one of the two endpoints, ie, non–isolated nodes in the pre–existing roadmap.

ObstBased uses information gathered during the generation phase in the connection phase. This implies some additional overhead to store with each node the obstacle to which it "belongs". Despite this additional memory requirement, the **ObstBased** connection method is a powerful one particularly in the case of crowded environments where it tends to find more of the problematic narrow passages than less informed methods [5, 3]. Except for the additional memory required to keep the extra information per node, this method is no more costly than **Closest**. It's a proven performer in crowded environment problems.

Note, **obstBased** is handicapped with respect to **closestVE**. Why? Because it never tries to connect to existing connected components, ie, those created in the first pass having nodes generated with a non–OBPRM generation method. The **obstBased** method only acts to connect OBPRM generated nodes.

5 Generation–Enhanced Connection

The Class III techniques tend to blur the line traditionally observed between the node generation and node connection phases of (PRMs). While the previous two classes allow the user to make binary decisions as to technique and approach, this class is more difficult to deploy and evaluate. This level requires some finesse on the part of the user.

5.1 Techniques

- **Components**
  This method *does not* generate any new nodes. It is used as a baseline against which to compare the others in this class. Try to connect all pairs of connected components. If both components are small (less than some parameter value of nodes), try to connect all pairs of nodes. If at least one of the components is large, we try to connect the "k" closest pairs of nodes.
  NOTE: can be combined with CheckCC

- **Expand Components**
  A slight modification of the above, this method *does* add new nodes. Whenever a connection attempt is made a new node is added whether the attempt was "successful" or not. If it was, the edge is added. If not, the last free node on the "failed" path is added as a new node in the graph and the part of the connection attempt that was collision free is added as an edge to that new node.

- **ModifiedLM**
  Laumond's idea [13] is to generate a new node at random and add to map only if it connects at least two previously disparate connected components(CCs) or it connected to no existing CC.

  Modified version only discards the new node if its distance from the center of the single CC to which it was connectable is smaller than twice the radius of that CC. In other words it will be kept only if it "expands" the CC.

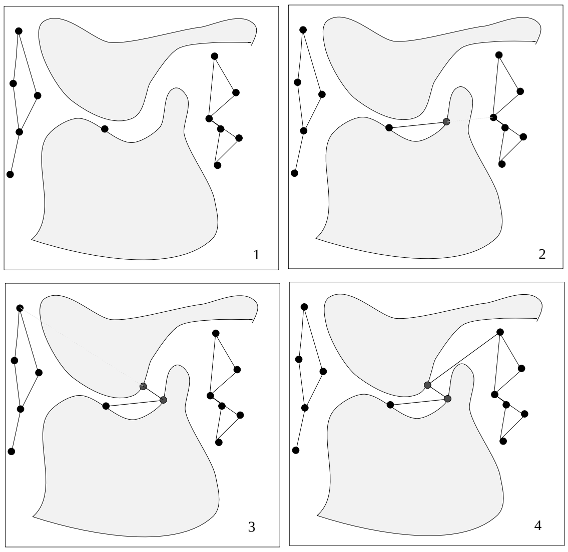

Figure 1: Expand Components Connection Strategy

5.2 Input

Input set of map nodes was generated by OBPRM using overlapping normals.

5.3 Results

Architecture: SGI Octane				
	Alpha Environment			
Strategy	sec	#CC	bigCC(2nd,3rd)	#V(E)
close20 Comp	77	32	59(29,22)	202(170)
close50	102	21	89(50,34)	202(181)
close100	206	14	102(52,34)	202(188)
close50 ModLM(X)	108	12	115(50,34)	212(200)
close50 ExpComp	143	20	516	1091

5.4 Discussion

As can be seen in the table, the generation-enhanced connection techniques can be very beneficial if used appropriately. For all these methods, the best results are obtained when one starts with a relatively small number of nodes, and then uses the `closest` strategy with a large value of k. This will grow connected components in the 'easy' regions. Applying the `components` after `closest` with a large k is not very beneficial as most of the connections have already been attempted.

Once the initial roadmap is built, the generation-enhanced methods **Expand Components** and **ModifiedLM** can be applied. They allow the generation of the nodes to be directed by the connectability of the local neighborhood. For **ModifiedLM**, new nodes are randomly generated, and they are retained if they can be used to improve either the connectivity of the roadmap or its coverage of the freespace. Our results show this did improve the connectivity somewhat over the closest with $k = 50$. For **Expand Components**, which adds nodes when connection attempts fail, the results were even better. We believe that this method, applied after closest with a large k, is the most promising.

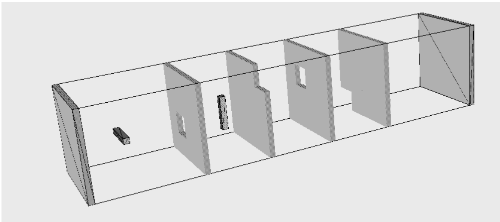

Figure 2: Walls

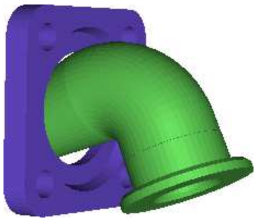

Figure 3: Flange

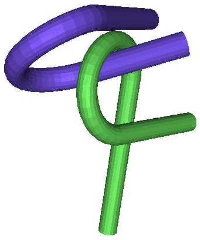

Figure 4: Alpha

Acknowledgement

We would like to thank the robotics group at Texas A&M. The alpha puzzle was designed by Boris Yamrom of the Computer Graphics & Systems Group at GE's Corporate Research & Development Center.

References

[1] J. M. Ahuactzin and K. Gupta. A motion planning based approach for inverse kinematics of redundant robots: The kinematic roadmap. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 3609–3614, 1997.

[2] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 630–637, 1998.

[3] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages 155–168, 1998.

[4] N. M. Amato and L. K. Dale. Probabilistic roadmap methods are embarrassingly parallel. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 688–694, 1999.

[5] N. M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 113–120, 1996.

[6] 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)*, 1999. To appear.

[7] S. Gottschalk, M.C. Lin, and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. Technical Report TR96-013, University of N. Carolina, Chapel Hill, CA, 1996.

[8] D. Hsu, J-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2719–2726, 1997.

[9] Y. K. Hwang and N. Ahuja. Gross motion planning – a survey. *ACM Computing Surveys*, 24(3):219–291, 1992.

[10] L. Kavraki, F. Lamiraux, and C. Holleman. Towards planning for elastic objects. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, 1998.

[11] 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.

[12] J. C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, MA, 1991.

[13] J.-P. Laumond. Stanford Workshop on Motion Support for Virtual Prototyping., 1999.

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

[15] S.M. LaValle, J.H. Yakey, and L.E. Kavraki. A probabilistic roadmap approach for systems with closed kinematic chains. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 1999.

[16] B. Mirtich. V-clip: Fast and robust polyhedral collision detection. Technical Report TR97-05, Mitsubishi Electric Research Lab, Cambridge, MA, 1997.

[17] 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.

[18] P. G. Xavier and R. A. LaFarge. A configuration space toolkit for automated spatial reasoning: Technical results and ldrd project final report. Technical Report SAND97-0366, Sandia National Laboratories, 1997.