

Probabilistic Roadmap Methods are Embarrassingly Parallel*

Nancy M. Amato, Lucia K. Dale
 {amato,dale1}@cs.tamu.edu
 Department of Computer Science, Texas A&M University
 College Station, TX 77843-3112

Abstract

In this paper we report on our experience parallelizing probabilistic roadmap motion planning methods (PRMs). We show that significant, scalable speedups can be obtained with relatively little effort on the part of the developer. Our experience is not limited to PRMs, however. In particular, we outline general techniques for parallelizing types of computations commonly performed in motion planning algorithms, and identify potential difficulties that might be faced in other efforts to parallelize sequential motion planning methods.

1 Introduction

Automatic motion planning has application in many areas such as robotics, virtual reality systems, and computer-aided design. Although many different motion planning methods have been proposed, most are not used in practice since they are computationally infeasible except for some restricted cases, e.g., when the robot has very few degrees of freedom (dof) [12, 16]. Indeed, there is strong evidence that any complete planner (one that is guaranteed to find a solution or determine that none exists) requires time exponential in the number of dof of the robot [23].

For this reason, attention has focussed on randomized or probabilistic motion planning methods. Notable among the randomized potential field methods is the Randomized Path Planner (RPP) of Barraquand and Latombe [7], which uses random walks to attempt to escape local minima. Recently, randomized or *probabilistic roadmap methods* (PRMs) have gained much attention for problems involving high-dimensional C-spaces [2, 3, 4, 6, 11, 14, 15, 21, 22].

Nevertheless, there exist many important applications which have not been solved by current meth-

ods, or cannot be solved within acceptable time constraints. For example, there are some problems which require extremely fast, or even real-time solution, such as animation or virtual reality applications.

1.1 Motion Planning in Parallel

In sharp contrast to the attention lavished on sequential motion planning, very little effort has been spent developing parallel planners. This is quite surprising given the high complexity of the typical motion planning problem and the frequent need for fast, or real-time solutions.

In particular, we are aware of only a few studies of parallel methods which can be applied in higher dimensional C-space. Lozano-Perez and O'Donnell [17] give a parallel algorithm for computing a discretized C-space for the first three links of a six degree of freedom manipulator. In parallel, the method precomputes the free space for each arm configuration that will be considered by a sequential search algorithm. Challou *et al.* [8, 9] formulated a parallel version of the randomized path planner (RPP) of Barraquand and Latombe [7]. Basically, in the parallel version, each processor independently runs the sequential RPP algorithm. When a processor finds a solution, it notifies all other processors they can stop their search. Even with such a straight forward parallelization, significant speedups were reported. Although this is the only work we are aware of specifically relating to motion planning, there have been some parallel methods proposed for collision detection [18, 19] and related geometric problems (see, e.g., [1, 5]).

These results illustrate that significant reductions in planning time can be obtained with parallel methods. We note, however, that none of the above mentioned motion planning methods is truly parallel. The former [17] does not parallelize the search itself, while the latter [8] cannot be expected to yield *scalable* speedups. That is, the expected running time is not inversely proportional to the number of processors used, since the fastest solution one can hope for is the time required by a fast sequential RPP execution. Thus, in order to consistently obtain scalable speedups we

*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.

need to use different approaches that actually partition the work to be performed over the set of available processors, e.g., parallel divide-and-conquer.

1.2 Probabilistic Roadmap Methods

Recently, a class of motion planning methods, known as *probabilistic roadmap methods* (PRMs), have gained much attention. PRMs are of particular interest to us as they are particularly amenable to parallel implementation. Briefly, PRMs use randomization to construct a graph in C-space (a *roadmap* [16]). 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 several 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.

Probabilistic roadmap methods, and in particular roadmap construction, stand to benefit greatly from parallelization. Because they are randomized, no roadmap node’s generation depends on any other. That is, the roadmap nodes can be produced independently on all processors in parallel. This minimizes the costly overhead of inter-processor communication which can drag down the performance of parallel algorithms [13]. The connection phase, where the heaviest costs of collision detection are incurred, is also naturally parallelizable.

Although the promise of parallel implementations has been acknowledged in the literature [10], we are aware of no specific implementations or empirical studies.

1.3 Our Results

In this paper we report on our experience parallelizing probabilistic roadmap methods (PRMs). This class of motion planning methods was selected because they

seem particularly well suited to parallelization. Indeed, we show that these planners are what is known as ‘embarrassingly parallel’ in the parallel computing community. Moreover, our rather straight-forward parallelization efforts obtained impressive, scalable speedups.

The conclusions of our study are more general, however, than just showing how this particular class of planners can be parallelized. In particular, we:

- show that significant, scalable speedups can be obtained for motion planning methods with relatively little effort on the part of the developer,
- outline general techniques for parallelizing types of computations commonly performed in motion planning algorithms, and
- identify potential difficulties that might be faced in other efforts to parallelize sequential motion planning methods.

The last point refers to the potential difficulties that will be faced when transforming existing motion planning methods for parallel execution. One important example is the use that many motion planning algorithms make of libraries for implementing common computations (e.g., collision detection). In these cases, the task of parallelizing the planner requires that the underlying libraries be amenable, and at the very least safe, for use in parallel applications. Our limited study already has identified that this is not generally true. The reason for this is that many of the libraries make frequent use of global data structures and/or static (register) variables to optimize performance. The safe use of these optimizations in parallel codes often requires making copies for each processor, which is error prone and increases memory requirements.

As workstations with multiple processors are becoming commonly available, similar problems will be encountered by ‘sequential’ methods as well. Therefore, we believe that the developers of these libraries should perhaps address these issues, and at least they should be aware of the potential problems. In some cases, explicitly parallel versions might be required.

2 Platform

The parallel machine used in this study was an SGI Origin2000 [24]. The Origin2000 is a ccNUMA machine (cache-coherent, non-uniform memory access), where sets of processors are connected to form *nodes*, and sets of nodes are connected in a hierarchical fashion by an *interconnection fabric* built of dynamically-allocated switch-connected links. Each processor was an 195 MHz R10000 microprocessor.

The Origin2000 supports a single global address space (although the memory is physically distributed among the processors). The shared address space allowed us to use a shared memory programming paradigm, which greatly simplified our programming task (i.e., we did not have to deal with the complications of message passing).

3 Parallelizing PRMs

If sequential solutions are too slow to be of practical use in some cases, they do provide a starting point for parallel solutions. This is true for PRMs which are the object of this study. Recall, a roadmap node in a PRM corresponds to a collision-free configuration (placement and orientation) of the robot. The roadmap construction proceeds in two stages: (1) generation of collision-free nodes, and (2) connection of these nodes to form the roadmap.

In our previous work, we have noted [4], as have others [10], that while the generation of nodes is very fast, the connection of them is not. Typically, 2-3% of execution time is spent in generation while the bulk of the remaining 97-98% is spent in connection. The reason for this is simple, more collision detection calls must of necessity take place there. For that reason, the biggest benefit will be achieved by parallelizing node connection. However, as it is easy to do, we also consider ways to parallelize node generation.

Even though node generation represents only a relatively small percentage of the code, effort spent parallelizing it is not wasted. In particular, since it does take such a relatively small amount of time, we could afford to spend more time generating nodes of "better" quality before turning things over to the connection phase. Our initial method is simple, but others more sophisticated that take advantage of the inherent parallelism of this phase could be implemented. Not only would they benefit, but the expensive second phase of node connection might then be more effective and produce better results. This is an example of working "smart" in addition to "hard".

3.1 Node Generation

Parallelizing the node generation phase of the basic PRM method [15] is trivial. If n nodes are desired, we simply ask each of the p processors to generate n/p of them.

PRM NODEGENERATION

(Each processor does the following)

1. for $1 \leq i \leq n/p$
2. generate a random cfg, c
3. if c is free
4. save c

5. endif
6. endfor

A (slightly) more sophisticated node generation algorithm is used in the obstacle-based PRM (OBPRM [4]). Let m be the number of obstacles.

OBPRM NODEGENERATION

(Each processor does the following)

1. for each C-obstacle X
2. $c_{in} :=$ colliding robot cfg with C-obstacle X
3. $D := n/pm$ random directions emanating out from c_{in}
4. for each $d \in D$
5. $c_{out} :=$ free cfg in direction d (if exists)
6. find contact cfg, c , on $\overline{c_{in}c_{out}}$ by binary search
7. save c
8. endfor
9. endfor

3.2 Connection

Effective parallelization of the roadmap connection phase is essential to obtain scalable speedups since this phase typically accounts for 98% of the preprocessing time. This phase is expensive due to the large number of collision detection checks made here. In node generation, the goal is to generate n nodes which are collision free and may also satisfy some other constraints such as the surface contact requirement of simple OBPRM. On the way to finding those nodes, rather gross adjustments can be made to the current "candidate" node. Consider the quickly converging refinements of OBPRM's binary search. In contrast, node connection involves checking for collision at resolutionally small increments along a path from one roadmap node to another. Only if all those incremental robot configurations are free can the two roadmap nodes be marked as *connected* in the roadmap being built.

Fortunately, formulating an effective parallel solution to the connection problem is not difficult. That our first intuition for parallelizing the connection phase is so effective emphasizes the massive inherent parallelism of PRMs.

A very simple sequential algorithm tries to connect every node to its k closest neighbors, for some constant k . An effective parallel solution then looks like this:

PRM NODECONNECTION

(Each processor does the following; each has a unique pid)

1. for each cfg c , indexed $p * (pid - 1)$ to $p * (pid)$
2. $N := k$ closest neighboring from all cfg's to c
3. for each $n \in N$
4. if local planner can connect n and c
5. save edge(n, c)
6. endif
7. endfor
8. endfor

A (slightly) more sophisticated node connection algorithm is used in the `CONNECTCOMPONENTS` algorithm of `OBPRM`. Here the k closest pairs of nodes from different connected components (of an existing roadmap graph) are considered for connection, with perhaps more expensive but more effective local planners (such as A^* -like methods) being used to attempt the connections. The goal is to reduce the number of connected components of the roadmap, by merging smaller components into larger ones.

4 Implementation Issues

4.1 Software Platform

All code was written in C++ and compiled with version 7.2.1 of the native SGI compiler. Parallel tasking was accomplished using the provided `m.fork` parallel microtasking library. One task was created for each processor used in the current execution; to save overhead, these tasks were reused by both generation and connection phases of the algorithm.

The sequential code used for collision detection was Voronoi Clip (V-Clip) [20]. To use this code in parallel, each processor was provided with its own hash table for keeping track of closest features between objects. There was no hacking of the V-Clip code.

4.2 Our Code

Our parallel methods are basically straight-forward parallelizations of our group's sequential PRM planners [4].

We parallelized the node generation and roadmap connection phases as outlined in Section 3. For example, during node generation, each processor independently generated its own set of roadmap nodes. Similarly, for the connection of roadmap nodes, each processor attempted its assigned connections. However, unlike the node generation phase, it is possible, and indeed likely, that some of the processors might generate the same connections. That is, our simplistic parallelization strategy causes some duplication of work. Also note that multiple processors (actually 2), will read each roadmap candidate node.

Since the time required to add nodes and edges to the roadmap graph data structure was insignificant as compared to the rest of the computation, we elected to serialize access to that data structure. In particular, during both the node generation and connection phases, each processor collected a list of all the additions it would like to make to the evolving roadmap graph. Then, after the parallel portion of the phase was finished, a single processor added all nodes or edges to the graph. As is seen in our experimental

results, this did not have a measurable impact on the speedups obtained (see Section 5). An additional benefit of serializing modifications to the roadmap graph was that the graph trivially remained uncorrupted (which is always an important concern when modifying shared data structures in parallel codes).

Finally, we note that our parallel node generation algorithms adhere to what is commonly known in the parallel computing literature as the exclusive-read exclusive-write (EREW) model, while our parallel connection algorithm belongs to the concurrent-read exclusive-write (CREW) model.

4.3 Collision detection libraries

As is the case with most motion planning algorithms, our sequential planners use libraries coded by others for collision detection [25, 20]. Unfortunately, as alluded to in Section 1, some difficulties were encountered when we tried to use these collision detection libraries in parallel. The underlying reason for this is that many of these libraries are optimized for performance on a *sequential* machine. For example, they use global auxiliary data structures to exploit spatial coherence between subsequent collision detection calls, or use static (register) variables for frequently accessed values. Since multiple processors will be executing collision detection calls at the same time, both of the above optimizations represent problems for parallel implementations.

The solution we adopted was to make a separate copy of the data that was modified during the collision detection call for each active processor. This was relatively simple in V-Clip [20], since the processors only needed separate copies of the hash tables they used to record the closest features between object pairs on previous collision detection calls. The modifications that would have been required in the C-Space Toolkit [25] would have been more extensive. However, we note that wholesale replication of data is not always a good solution as it may prove overly costly in terms of memory requirements.

5 Experimental Results

In this section we report the experimental performance of our parallel node generation and connection algorithms. Our measurements were made on heavily utilized multi-user systems, in interactive mode. Nevertheless, we consistently obtained measurable speedups if the problem size was large enough. However, measurements of small tasks (e.g., less than 30-35 seconds) were unstable. Recall, as previously mentioned, that node generation typically represents only 1-2% of the total roadmap construction time. For

this reason, we elected to measure the node generation and the connection phases on different problem sizes.

All algorithms were run on 1, 2, 4, 8, and 16 processors. Each experiment was run several times, and the results reported are the best times obtained. The best times on an interactive system represent those times when our experiment suffered the least interference from competing processes. A summary of the results is contained in Table 1.

The motion planning environment we considered was a three-dimensional environment containing two identical parallel blocks, each of dimension $1 \times 20 \times 20$. The 'robot' is a smaller block ($1 \times 2 \times 2$) that can barely fit in the corridor between the parallel blocks. Although this environment is fairly simple, we note that similar speedups can be expected for any environment. That is, our conclusions are not dependent upon the particular problem instance, but only on the parallel implementation of the PRM methods themselves.

For node generation, the results reported are for problem instances in which we asked the methods (PRM and OBPRM) to generate 2000K and 200K, respectively, nodes. It was necessary to use such large problems to obtain running times that were large enough to be meaningful on the system used. The results for the PRM generation are shown in Figure 4, and the results for OBPRM generation are shown in Figure 5. As can be seen, very good speedups were obtained even on such a heavily utilized system. Table 1 gives more detailed results, including the number of nodes successfully generated.

For node connection, the results reported are for problem instances in which we asked the node generation phase to generate 1500 nodes, which generally yielded approximately 1300 nodes for the connection phase. The connection algorithm used $k = 5$, so that connections were attempted between a node and the 5 other nodes closest to it. The results for the connection phase shown in Figure 6 show similar results as the node generation algorithms. Table 1 gives more detailed results, including some statistics regarding the roadmaps generated such as the number of connected components in the resulting roadmap, and the size of the largest connected component.

6 Conclusions and Future Work

Based on our experiments and experience parallelizing probabilistic roadmap motion planning methods, it seems significant, scalable speedups can be obtained with relatively little developer effort. The PRM methods have massive inherent parallelism which is easily and perhaps optimally exploited even by relatively simplistic parallelizing strategies.

In the future, we also intend to implement several

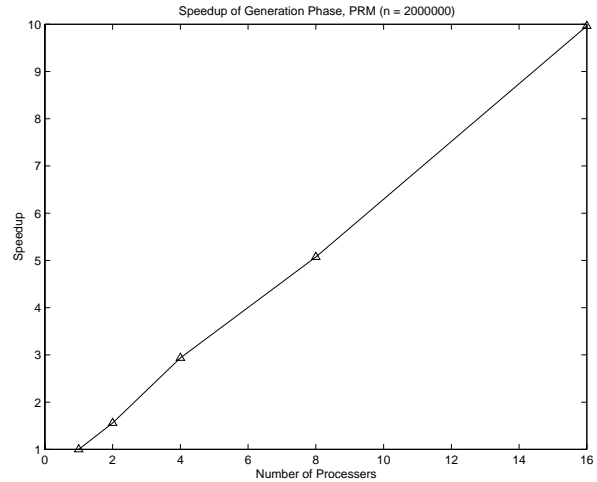


Figure 1: SGI Origin 2000 - Speedups - Generation Phase, PRM

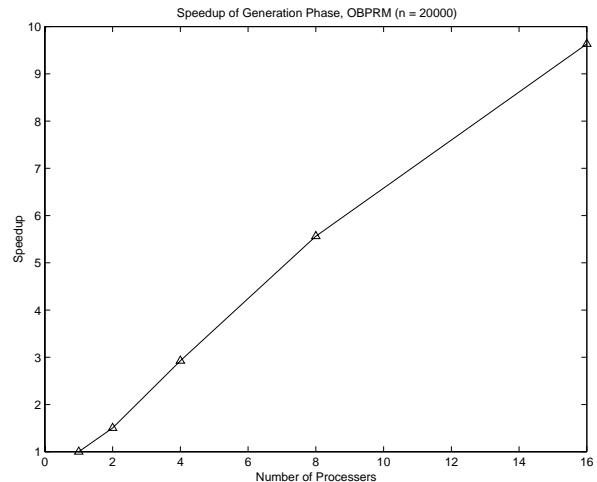


Figure 2: SGI Origin 2000 - Speedups - Generation Phase, OBPRM

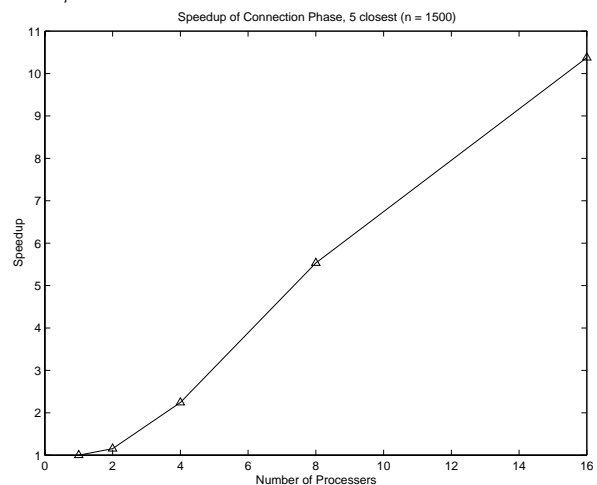


Figure 3: SGI Origin 2000 - Speedups - Connection Phase (5 closest)

Architecture: SGI Origin 2000								
Proc	Generation Phase				Connection Phase			
	PRM		OBPRM		K-Closest			
	sec	nodes	sec	nodes	sec	edges	cc	size
1	279	1718263	395	81912	83	3955	7	1292
2	179	1718254	263	81871	72	3955	7	1292
4	95	1718276	135	81704	37	3920	6	1290
8	55	1718254	71	81463	15	3955	7	1292
16	28	1718243	41	81360	8	3955	7	1292

Table 1: Preprocessing computation times and statistics for Generation and Connection phases of roadmap building. For Generation, we always requested PRM for 2000K nodes and OBPRM for 200K nodes. For Connection, we always worked problems which had previously requested 1500 nodes.

more sophisticated algorithms for generating “better” quality roadmap nodes in the generation phase and more sophisticated algorithms for selecting which roadmap nodes between which to apply resolutionally bound local planning methods in the connection phase. We expect to see continued benefit from the “embarrassing” parallelism of these PRM motion planning techniques.

Acknowledgement

We would like to thank the robotics group at Texas A&M.

References

- [1] A. Aggarwal, B. Chazelle, L. Guibas, C. Ó’Dúnlaing, and C. Yap. Parallel computational geometry. *Algorithmica*, 3:293–327, 1988.
- [2] 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.
- [3] 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.
- [4] 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)*, 1998.
- [5] N. M. Amato, M. T. Goodrich, and E. A. Ramos. Parallel algorithms for higher-dimensional convex hulls. In *Proc. 35th IEEE Symp. Foundat. Comput. Sci.*, pages pp. 683–694, 1994.
- [6] 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, Minneapolis, MN, April 1996.
- [7] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *Int. J. Robot. Res.*, 10(6):628–649, 1991.
- [8] D. Challou, D. Boley, M. Gini, and V. Kumar. A parallel formulation of informed randomized search for robot motion planning problems. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 709–714, 1995.
- [9] D. J. Challou, M. Gini, and V. Kumar. Parallel search algorithms for robot motion planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, volume 2, pages 46–51, 1993.
- [10] D. Hsu, L. Kavraki, J.-C. Latombe, and R. Motwani. Capturing the connectivity of high-dimensional geometric spaces by parallelizable random sampling techniques. In *Proc. IEEE Workshop Randomized Parallel Computing (WRPC)*, 1998.
- [11] 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.
- [12] Y. K. Hwang and N. Ahuja. Gross motion planning – a survey. *ACM Computing Surveys*, 24(3):219–291, 1992.
- [13] J. JàJà. *An Introduction Parallel Algorithms*. Addison-Wesley, Reading, Massachusetts, 1992.

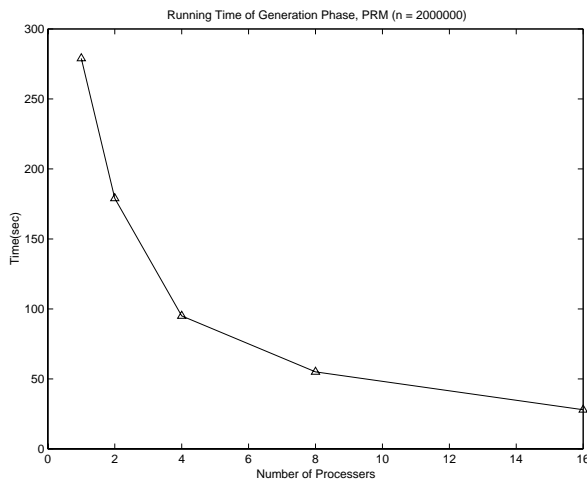


Figure 4: SGI Origin 2000 - Running Time - Generation Phase,PRM

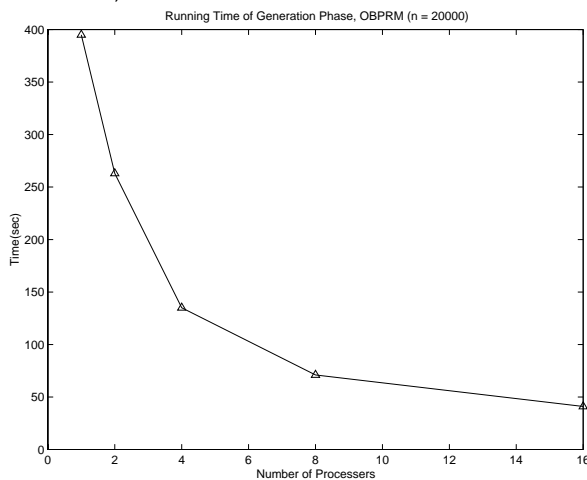


Figure 5: SGI Origin 2000 - Running Time - Generation Phase,OBPRM

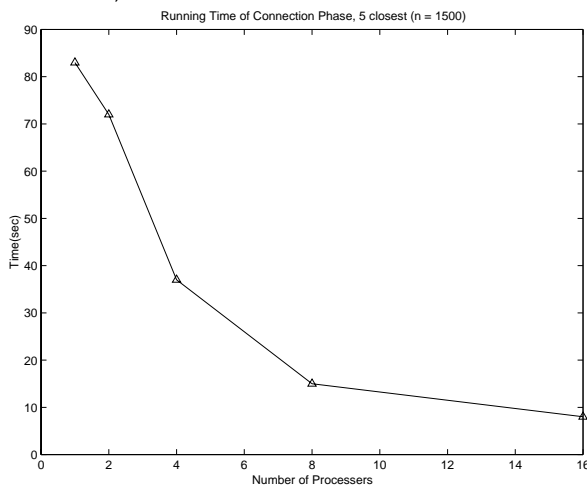


Figure 6: SGI Origin 2000 - Running Time - Connection Phase (5 closest)

- [14] L. Kavraki and J. C. Latombe. Randomized preprocessing of configuration space for fast path planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2138–2145, 1994.
- [15] 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.
- [16] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.
- [17] T. Lozano-Pérez and P. O'Donnell. Parallel robot motion planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1000–1007, 1991.
- [18] B. Martínez-Salvador, A. del Pobil, and M. Pérez-Francisco. Very fast collision detection for practical motion planning part i: The spatial representation. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 1998.
- [19] B. Martínez-Salvador, A. del Pobil, and M. Pérez-Francisco. Very fast collision detection for practical motion planning part i: The parallel algorithm. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 1998.
- [20] B. Mirtich. V-clip: Fast and robust polyhedral collision detection. Technical Report TR97-05, Mitsubishi Electric Research Lab, Cambridge, MA, 1997.
- [21] M. Overmars. A random approach to path planning. Technical Report RUU-CS-92-32, Computer Science, Utrecht University, The Netherlands, 1992.
- [22] M. Overmars and P. Svestka. A probabilistic learning approach to motion planning. In *Proc. Workshop on Algorithmic Foundations of Robotics*, pages 19–37, 1994.
- [23] J. Reif. Complexity of the piano mover's problem and generalizations. In *Proc. IEEE Symp. Foundations of Computer Science (FOCS)*, pages 421–427, 1979.
- [24] Silicon Graphics Corporation, <http://techpubs.sgi.com/library/>. *Performance Tuning Optimizatoin for Origin2000 and Onyx2*.
- [25] 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.