# Parallelizing Probabilistic Roadmap Variants

Kevin Li[1], Nicole Neil[2], Kevin Grant Li[3], Joseph Han[4], Avigayil Helman[5]

***Abstract*— The team's goal is to parallelize a few of the many existing variations of Probablistic Roadmap Methods (PRM) algorithms and evaluate their time complexity in order to analyze potential in theoretical efficiency gains. Standard PRM and Adapative Neighborhood Connection (ANC) PRM are implemented to represent a simplified robotics scenario in which a path is generated from a start goal, around a series of obstacles, to an end goal. These implementations are completed both in serial and parallelized fashion, and for the latter using GPU work and CPU multithreading capabilities. In general, the parallel versions are show efficiency gains compared to their serial counterparts (up-to 40% reduction in runtime). However, there are nuances and further areas to explore in order to best capture and maximize efficiency gains.**

## I. Introduction

Robotic motion planning problems have received considerable attention, especially as robots become an integral part of modern life with the services they can provide in industrial, residential, and other settings. These problems attempt to find a sequence of control inputs and sequence of movements for a robot, given a description of its dynamics, environment, constraints, initial state, and a single or set of goal states. Over the last few decades, many different optimization-based motion planning methods have been developed and tested, including direct transcription and iLQR (iterative Linear Quadratic Regulator) methods. Even with these methods, however, efficiently solving motion planning problems continues to be difficult, even with the parallelization of specific operations. An alternative to these optimization-based trajectory methods is sample based motion planning methods [1].

Thus the team's primary motivation is to investigate Sample Based Motion Planning methods (SBMP) as opposed to the Optimization Based Motion Planning methods (OBMP) that were taught in the Columbia University course COMS BC3159 by Brian Plancher. The goal is to parallelize Probabilistic Roadmaps (PRM) variants and evaluate their time complexity to reach conclusions about their efficiency. Sample based motion planning approaches naturally admit more parallelism than optimization based approaches (but come with their own scalability weaknesses). There have been a series of recent works exploring how to implement these on the GPU [2].

Ultimately, the team chose to implement standard PRM and ANC PRM in serial and parallel for a simplified robotics scenario that generates the shortest path from the start goal, around obstacles, to the end goal. We then compared the time complexity of the implementations, concluding that the parallelized versions were indeed more efficient. In general, to complete larger tasks, the parallel versions are more efficient than the serial versions (up-to 40% reduction in runtime). However, we have identified problems in parallelizing certain components of these algorithms and shown performance gains in certain portions to be especially optimal.

Our code repository with execution instructions can be found at:
https://github.com/kgl2123/PRM-Project-Final/.

## II. Related Work

1) Standard PRM has been implemented numerous times since the algorithm's invention in the 1990s, and it is known that it is possible to parallelize the basic version of the PRM algorithm to achieve significant, scalable speed-ups [3] [4]. The main drawback of PRM is that it produces less than optimal solutions due to the sampling of node generation. This issue was investigated as this team tested various sample sizes for node generation and parallelized the process. This group's goal was to implement the basic algorithm of PRM in serial and parallel to then compare their runtimes.
2) ANC PRM has been implemented as a variant as PRM in which a general connection framework adaptively selects a neighbor finding strategy from a candidate set of options [5]. This PRM variant learns which strategy to use by examining each success rate and cost, thus allowing the selection to change over time [5] [6]. We implemented serial and parallelized ANC.

## III. Background

### A. Optimization Based Motion Planning (OBMP)

Optimization-based motion planning is a category of methods used in robotics to plan the motion of a robot in a given environment. It involves creating the path that minimizes an objective function while satisfying the given constraints, such as collision avoidance and kinematics. OBMP is especially useful for tasks where it is important to find the best possible solution, such as in manufacturing, autonomous vehicle operation, and other tasks that require precision [7].

[1]Kevin Li is with Columbia University New York, NY. kl3285@columbia.edu
[2]Nicole Neil is with Columbia University New York, NY. njn2122@columbia.edu.
[3]Kevin Grant Li is with Columbia University New York, NY. kgl2123@columbia.edu.
[4]Joseph Han is with Columbia University New York, NY. jh4632@columbia.edu.
[5]Avigayil Helman is with Columbia University New York, NY. abh2177@columbia.edu.

The given scenario for OBMP is set up as an optimization program with an objective function that measures the cost of the planned movements and considers the constraints that represent the particular motion requirements of the robot. The objective function can be a combination of different cost functions, such as distance traveled, energy consumed, or time taken, depending on the application of the OBMP algorithm. The optimization problem is then solved using various optimization techniques such as nonlinear programming, quadratic programming, and convex optimization. The solution derived from the chosen programming method provides a continuous trajectory for the robot to follow.

In general, optimization-based motion planning has many advantages over other motion planning methods, as it is able to generate optimal solutions that satisfy a specific range of constraints, including constraints that are difficult to model and ones that change over time. However, optimization-based motion planning has some significant limitations [7]. It can be very computationally expensive, especially for scenarios that are high-dimensional problems or have large environments. It can also be challenging to define the objective function and constraints for complex robotic systems. Overall, it is still suitable for a wide range of robotics applications.

### *B. Sample Based Motion Planning (SBMP)*

Sample-based motion planning is another category of methods used in robotics to plan the motion of a robot in a given environment. It is a popular approach for finding feasible collision-free paths for robots in complex and high-dimensional environments. SBMP constructs a graph, known as a roadmap, that represents the potential movement paths of the configuration space of the robot. The roadmap is constructed by randomly sampling configurations and connecting nearby configurations that are collision-free [8].

The two most common sample-based motion planning algorithms are the Probabilistic Roadmap (PRM) and the Rapidly-exploring Random Tree (RRT). The PRM algorithm constructs a roadmap by randomly sampling configurations and connecting them if a collision-free path can be found between them [3]. In contrast, the RRT algorithm constructs a tree structure by incrementally growing the tree in the direction of a randomly sampled configuration, with the tree growth biased towards the less densely explored areas of the configuration space [9].

In comparison to optimization based motion planning, sample based motion planning has several advantages. It works well for high-dimensional and complex environments in scenarios where the optimization problem becomes computationally complex and cannot be solved. It can also more easily consider non-convex obstacles when finding the best path than most OBMP methods. Due to its probabilistic nature, it can provide a probabilistic completeness guarantee, which means that a solution can be found with a high probability, unlike OBMP [10].

It makes sense then that one of the main challenges in sample-based motion planning is the trade-off between computational efficiency and the quality of the generated paths. The number of samples required to construct a very accurate roadmap can be very large, which then leads to long computation times. Even then, the generated paths are often not optimal or smooth. These challenges have been addressed by various extensions and modifications to previously developed sample-based motion planning algorithms, such as PRM and RRT by including path smoothing and some optimization [11]. Thus, while it has limitations in terms of computational efficiency regarding node sampling and path quality, the probabilistic completeness guarantee and the flexibility in handling non-convex obstacles makes SBMP useful for many robotics applications [12].

### *C. Probabilistic Roadmaps (PRM)*

The standard PRM method is an offline algorithm that can also be implemented online. On a high level, PRM involves connecting randomly sampled points from the state space. A graph is subsequently constructing a graph (roadmap) by connecting all nodes. Nodes with edges that collide with obstacles are not connected. The shortest path connecting the initial state and the goal state is then determined using chosen graph traversal methods [3].

PRM High Level Pseudocode [13]:

1) For a given configuration space, randomly generate n nodes that are collision-free.
2) Connect nodes to form a roadmap. Edges should not collide with obstacles in configuration space.
3) Query processing: Determine shortest path between initial and goal states and return result.

Given the versatility of the standard PRM framework, many variations exist. Alterations to the base code by changing node connection methods and/or shortest path query methods can drastically affect computation efficiency and determined results. Some PRM variations include [6]:

PRM, K-closest connection: For each node, the k closest neighbors based on a distance metric (i.e. Euclidean distance between two nodes) are connected.

PRM, r-closest connection: For each node, all neighbors within a radius r of the node determined by some distance metric are connected.

PRM, K-closest K-Rand: For each node, k neighbors are randomly selected from the $k_2$ closest nodes.

PRM, r-closest, k-rand: For each node, select k random neighbors from within a distance r.

ANC PRM: Generates a set of candidate neighbors for a node q for PRM connections using a list of neighbor finders. A selection probability for each neighbor finder is learned overtime depending on success rate and cost.

In addition to adaptability, the standard PRM algorithm is inherently highly parallelizable: Node generation and connection, for instance, can be parallelized [4].

### *D. Adaptive Neighborhood Connect (ANC PRM)*

The key feature of ANC PRM is the adaptive selection of neighboring nodes, which allows for a more efficient and effective construction of the roadmap. Unlike the standard

PRM algorithm, which connects all nodes within a distance determined by the specific nearest neighbor search method used, ANC PRM adjusts the radius dynamically based on the local density of nodes. To be precise, ANC PRM defines a density function that estimates the number of nodes within a certain radius, and then determines the optimal distance for connecting neighboring nodes by utilizing a variety of nearest neighbor search methods. This adaptive selection of neighboring nodes helps to prevent over-connectivity in dense areas, while ensuring sufficient connectivity in relatively sparse areas [5].

### *E. Nearest Neighbor Search*

1) k-nearest neighbors: For each node, the k closest neighbors based on a distance metric (i.e. Euclidean distance between two nodes) are connected.
2) r-nearest neighbors: For each node, all neighbors within a radius r of the node determined by some distance metric are connected.
3) $k_2$-closest k-random neighbors: For each node, k neighbors are randomly selected from the $k_2$ closest nodes.
4) r-closest k-random neighbors: For each node, k random neighbors are selected within a distance of radius r.

## IV. Design / Implementation / Algorithm

### *A. Overall Motivation*

This team's proposed scenario is inspired by autonomous vehicle operation, which is a very complex problem to solve. These vehicles drive in unique, dynamic environments that require it to perform real-time motion planning in order to maneuver in the space safely and successfully. Algorithms based on probabilistic roadmaps have been developed and used for these vehicles' motion planning by incorporating system dynamics and a variety of functions [14].

To adapt this real life application of PRM for the scale of this project, the scenario of autonomous vehicle parking among other cars was simplified to shortest path generation around a series of obstacles, between a designated start and end point. To do this, inspiration was drawn from an RRT skeleton code from Columbia Course COMS 3997-F22. The team implemented standard PRM and ANC PRM in serial and parallel. The algorithms were coded in a C++ and Python and a combination of CUDA and pyCUDA were used for the parallelized algorithms. Pygame was used to display the environment and generated paths during the development and testing of the code, but it was removed for the final test of time complexity due to its long additional runtime. Each implementation then had its time complexity measured for a comparison of their efficiency.

### *B. Completed Approaches*

1) Serial Standard PRM with k-nearest neighbors search
2) Serial ANC PRM with neighbors search involving:
   - k-nearest neighbors
   - r-nearest neighbors
   - $k_2$-closest k-random neighbors
   - r-closest k-random neighbors
3) Parallelized Standard PRM with k-nearest neighbors search. These components were parallelized:
   - GPU-parallelized node sample generation and obstruction checking
   - GPU-parallelized nearest neighbor search
   - GPU-parallelized edge formation
4) Parallelized ANC PRM with the above four neighbor searches. These components were parallelized:
   - GPU-parallelized node sample generation and obstruction checking
   - CPU-parallelized adaptive neigborhood connection and edge formation

### *C. Configuration Space*

The configuration space was constructed as a two dimensional plane [2] of positive x and y coordinates that house the start and end goal, obstacles, nodes, edges, and constructed potential paths. Once the configuration space was generated, the start and end goals were generated as point in the plane. Then rectangular obstacles were each generated as two points connected by a line with a given width. This configuration space setup is used for every implementation of PRM.

### *D. Overall Implementation of PRM*

Nodes are randomly generated as points in the 2D c-space. Collision checks are performed between all of the generated nodes and the obstacles. The nodes that collide are removed and the ones that do not are saved to a list of valid nodes. A nearest neighbor search (k-nearest neighborhood) for each node is performed. Collision checks are performed between all of the potential edges and the obstacles. The edges that collide with obstacles are removed and the ones that do not are saved to a list of valid edges. The list of valid nodes and the list of valid edges forms a Graph to be traversed. Dijkstra's Algorithm for weighted graph is then used for graph traversal to find the shortest path generated between the start and end goal. This is consider the resultant path.

### *E. Parallelization Potential for Standard PRM*

These following components of the standard PRM implementation were were parallelized: node-sampling, neighborhood connection, and edge formation.

In the serial implementation, an iterative loop based on the number of samples to generated is used to generate the sample nodes one by one. This is a O(N) time complexity. We saw this task was easily amenable to parallelization. By launching a kernel that separated responsibility of generating samples across multiple threads, all samples could be generated concurrently. This reduces computational complexity to O(1).

In the serial implementation, for neighborhood connection, a nested for loop was implemented so for each node, it would compare its distance to every other potential node until it finds its closest neighbors. As the neighbor finder for each node is done sequentially after the previous node, and each node has to be compared against all other neighbors, this

constitutes a time complexity of O($N^2$). This also constitutes a task that was attractive to parallelize. By launching a kernel that separated responsibility of building the neighborhood across multiple threads, the neighborhood connection for all nodes could be completed concurrently. This reduces computational complexity is O(N).

In the serial implementation, for edge collision checking, a loop was implemented to check each edge against all obstacles in the configuration space. This time complexity is O($N^2$). By launching a kernel that separated responsibility of checking edges samples across multiple threads, all edges could be checked concurrently, which reduces time complexity to O(N).

Early on the research, the team agreed that parallelization for the Graph Traversal component would not optimize the work. Graph Traversal component in PRMs involves traversing a roadmap of feasible configurations to find a collision-free path between the start and goal configurations. This involves performing a series of graph search algorithms, such as Dijkstra's algorithm or A* search. These methods are inherently sequential in their computations. The overall search process cannot be easily parallelized without introducing significant communication overhead and synchronization issues. This is because the search algorithms require access to the entire roadmap, and updating the roadmap during the search can introduce race conditions and other issues. This is the reason why in the resulting parallel benchmarking code, the Dijkstra's algorithm is actually still implemented in serial. The resulting runtime results also reflect this reality.

### *F. Overall Implementation of ANC*

For Adapative Neighborhood Connection, many neighbor connectors (k-nearest neighbor, r-closest neighbor etc.) are considered in tandem. A selection probability for each neighbor finder is learned over time depending on success rate (and costs). This heuristic originally randomly chooses between the different neighbor connectors but as learning occurs, the neighborhood connections adapts to converge on typically one or two preferred methodologies. Edges are constructed as each node is considered.

### *G. Parallelization Potential for ANC*

We attempted to parallelize ANC by separating the work in first, completing computation work for each neighbor connector on separate threads, and second, separating the work of all the nodes across different threads. These efforts were made to parallelize ANC using pyCUDA. However, we discovered that the memory overhead could potentially undermine our efficiency goals, especially since the launched kernel would be limited to a small number of threads (i.e., single digits). This limitation arises because ANC requires learning and depends on the connection results of individual nodes. Consequently, we determined that multithreading was a more suitable approach for parallelizing ANC.

During the benchmarking of parallel ANC, we encountered the restrictions imposed by Python's Global Interpreter Lock (GIL), which prevents multiple threads from running concurrently. As a result, the implemented parallel ANC algorithm does not achieve true parallelism. The limitations of GIL also account for the longer run times can be observed in our results. We expect an actualized parallel implementation to show reduced run times than its serial counterpart.

## V. Visualization Results

We aimed to visualize the results of the PRM and ANC PRM Sample-based Motion Planning algorithms. We completed the visualization with python and pygame. Any machine with those capacities suffice to run these components.

We did not visualize results of the parallelized functions. This was due to the primary interest in parallelization for analysis of runtime efficiency gains in benchmarking. Technically, this was also the case due to the necessity of using the Google Colab iPython Notebooks to access the GPU runtime environment, which is incompatible with pygame.

### *A. ANC Results Analysis*

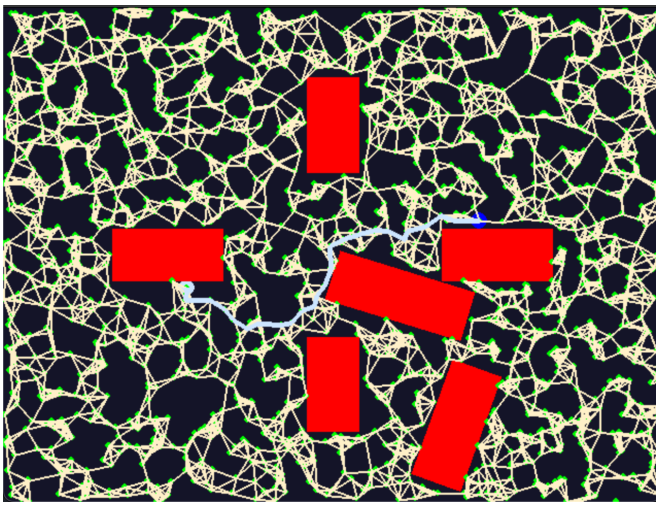

Fig. 1. ANC k-nearest dominant result.

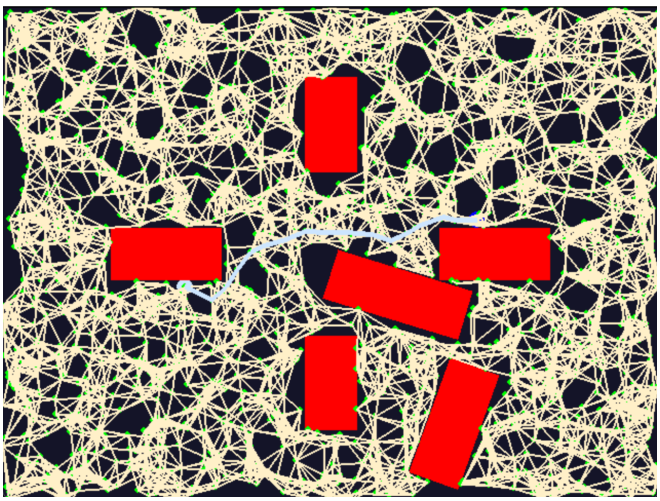

Fig. 2. ANC k2-closest-k-random dominant result.

The Adaptive Node Connections (ANC) algorithm is designed to work with various environments, resulting in different outcomes depending on the chosen dominant neighbor finder. As illustrated in Figure 1 ANC k-nearest and Figure 2 ANC k2 above, ANC can generate distinct graphs for the same spatial configuration due to the inherent randomness (i.e., randomly sampled points). To enhance computational efficiency and planning, multithreading is employed, allowing ANC to run on several threads (each processing a portion of nodes) and average the possible outcomes, as demonstrated in Figure 3 ANC-thread.

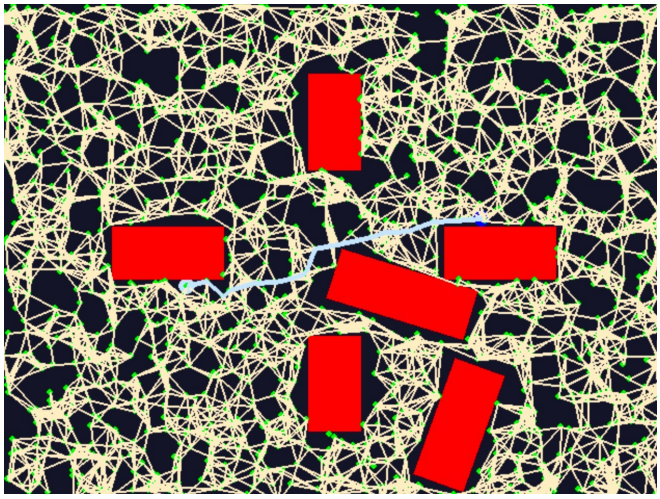

Fig. 3. ANC thread based parallelism result.

| | kNN | rNN | k2 k-rand | r k-rand |
|---|---|---|---|---|
| relative weight | 0.4 | 5.44e-06 | -0.60 | 3.99e-06 |
| frequency | 671 | 135 | 877 | 104 |

Fig. 4. Table showing ANC comparison. k-nearest = kNN, r-nearest = rNN, k2-closest k-rand = k2 k-rand, r-closest k-rand = r k-rand

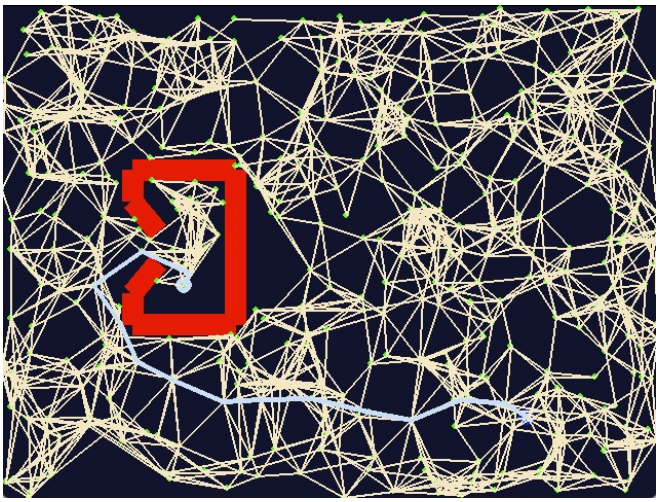

Fig. 5. ANC PRM BugTrap.

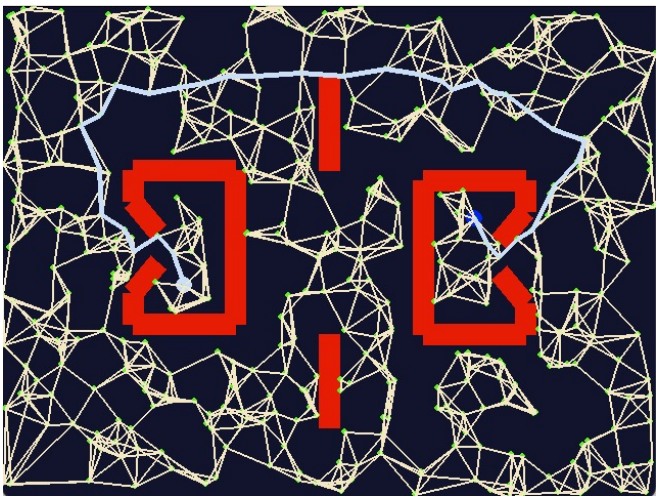

Fig. 6. PRM Double BugTrap.

### *B. SBMP vs OBMP - Results Analysis*

Although optimization based motion planning is usually very good at generating a very accurate shortest path, there are scenarios in which it is incapable of calculating paths due to the location of the start and end goals relative to the environment's obstacles. The following figures show examples of how ANC PRM is able to calculate paths in scenarios where OBMP is not.

The ANC PRM BugTrap Example has the start goal housed inside a series of obstacles that form a nearly closed loop around it. The end goal is located in the c-space opposite the opening of the loop, in a spot that would cause a typical OBMP algorithm to draw a shortest path from the start through the loop of obstacles to the end goal. Here however, the ANC algorithm correctly draws a path that does not collide with the obstacles and connects the end goal to the start goal. This shows one way that SBMPs will succeed where typically OBMPs will fail.

The PRM Double BugTrap Example has the start goal housed inside a series of obstacles that form a nearly closed loop around it. The end goal is also housed inside a series of obstacles that form a nearly closed loop around it. The openings of the two loops face away from each other and there are additional obstacles in between the two loops. Typical OBMP algorithms would attempt to build a trajectory with the shortest path straight through the loop obstacles to connect the start and end goals. Here however, the PRM algorithm correctly draws a path that does not collide with the obstacles and connects the end goal to the start goal. This shows another way that SBMPs will succeed where OBMPs will fail.

The ANC Zero Obstacles Example has the start goal and end goal placed on opposite ends of the c-space with zero obstacles in the environment. With no obstacles in the way of node, edge, and path generation, the ANC algorithm is able to find a decently optimal path between the start and end goals. Increasing the number of samples allows the ANC algorithm to generate a path that is closer to the Euclidean shortest path that would be generated by an OBMP algorithm.

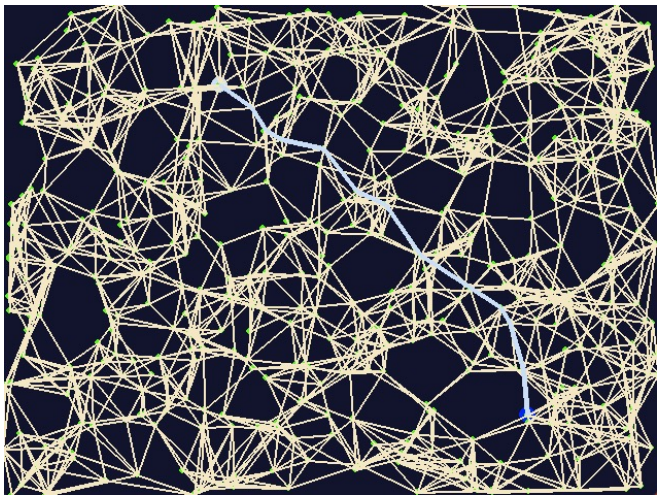

Fig. 7. ANC Zero Obstacles.

## VI. Benchmarking Results

We aimed to analyze the run-time results of PRM for different implementations (standard and ANC). We were interested in comparative results between standard and parallel implementations and understand if expected optimizations were achieved.

### A. Methodology

We completed all benchmarking testing simulations with the following hardware specifications:

- Google Colab iPython Notebooks.
- Google Colab default runtime GPUs (for components that implemented GPU Parallelization)

We also removed any GUI (pygame) related components for benchmarking results.

All simulation scenarios had identical configuration spaces. The configuration space with relevant definitions for starting and goal positions, obstacles, and configuration dimensions are provided in the benchmarking_scenario_specs.txtfile that is found in the GitHub repository.

All of the benchmarking was done with 100 runs except for the entries highlighted in green in table of Figure 8 for PRM runtime results. These were done with 10 runs due to testing time constraints.

The results of the benchmarking for PRM matched the team's expectations, and this is visible in the order of magnitude change between the number of samples run and the measured runtimes of each of the compared serial and parallel functions.

### B. Analysis of Standard PRM Benchmarking

In general, the resulting runtime complexities comparisons between serial and standard PRM implementation fall in line with the team's expectations.

For the sampling function, the team predicted a O(N) to O(1) collapse in time complexity. This is seen through the specific results of the runtime sampling function. For serial, as samples are generated iteratively in a loop structure, the time complexity increase is directly proportional to the increase in samples generated. For parallel, the results show a non-directly proportional scaling of runtime results. At first, the runtime difference between generating 100 versus 10000 samples is negligible because parallelization allowed all samples to be generated concurrently in different threads, rather than sequentially. However, the results did not directly scale as we further increased the number of samples generated. This could be because of I/O, memory, and related overhead in preparing for launching, running, and returning from CUDA kernels. Optimizing this component is of interest to the team in future work.

For the standard neighborhood function, the team predicted a $O(N^2)$ to O(N) collapse in time complexity. This is seen through the specific results of the Neighborhood Connection function. For serial, as nodes are generated in a iterative nested-loop structure, the time complexity increase is squared proportional to the increase in samples generated. For parallel, the results show a directly proportional scaling of runtime results as expected. This is the expectation as for all nodes, the connection to other nodes are done concurrently. The increased runtime results directly scale proportional to the increased number of samples generated. This is the strongest candidate that the team identified for optimization based on parallelization in PRM.

While edge collision is implemented in serial and parallel, there is a limited change between the two implementations. This is primarily due to the limited number of obstacles (6) in the configuration space. As we scale the number of obstacles in the space, we can expected additional comparative runtime gains for the parallel implementation compared to serial version, similar to that seen of the Standard Neighborhood Connection.

Dijkstra's algorithm, as mentioned in the Design Section of this report, was chosen to be implemented serially in both the serial benchmarking code and parallel benchmarking code. As a result, you see similar results across the two benchmarking tests.

As for the total runtime, we can see as the number of samples generated increases, the parallel implementation increasingly perform comparatively better against its serial counterpart. The results show an average 40% decrease and runtime on 10000 sample runs.

The observed gains can be attributed because the benefits in decreasing the algorithmic computational complexity with concurrent operations compensates for I/O and memory overhead necessary to perform GPU parallelization work, which over time, is amortized into the runtime complexity.

### C. Analysis of ANC PRM Benchmarking

For our implementations, we do not observe a marked improvement in runtime performance with the parallelized ANC PRM compared to the serial ANC PRM approach.

As noted in the Design section of this report, while ANC Neigborhood Connection and Edge Formation functions in the parallelized implementation were completed with Python

| Standard PRM | Sampling | | Standard Neighborhood Connection | | Edge Collision | | Dijkstra | | Total | |
|---|---|---|---|---|---|---|---|---|---|---|
| Samples | *Serial* | *Parallel* | *Serial* | *Parallel* | *Serial* | *Parallel* | *Serial* | *Parallel** | *Serial* | *Parallel* |
| 100 | 0.001 | 0.22708 | 0.00453 | 0.00485 | 0.01083 | 0.0139 | 0.00962 | 0.01198 | 0.02694 | 0.26043 |
| 1000 | 0.01018 | 0.23064 | 0.51595 | 0.04303 | 0.11357 | 0.13579 | 1.20457 | 1.20853 | 1.85703 | 1.63582 |
| 10000 | 0.10311 | 0.30626 | 109.77594 | 0.6312 | 1.36027 | 1.78124 | 128.12088 | 148.70885 | 239.59601 | 151.70467 |
| 100000 | 1.07961 | 0.71715 | | | | | | | | |
| 1,000,000 | 10.93192 | 5.25248 | | | | | | | | |
| | | | | | | | | | | |
| Expected Time Complexity: | O(N) | O(1) | O(N^2) | O(N) | O(N^2) | O(N) | | | | |

Fig. 8. Benchmarking for Time Complexity of Serial and Parallel Standard PRM (seconds).

| ANC PRM | Adapative Neighborhood Connection + Edge Collision Checks | |
|---|---|---|
| Samples | Serial | Parallel |
| 100 | 0.010551 | 0.13785 |
| 1000 | 0.88562 | 2.70886 |
| 10000 | 26.42296 | 48.8616 |

Fig. 9. Benchmarking for Time Complexity of Serial and Parallel ANC PRM.

standard multithreading, the limitations due to restrictions of Python's Global Interpreter Lock, which prevents multiple threads from running concurrently. This prevents proper parallelization of the computations and such limitations are displayed in result table. The parallelized version takes far longer compared to the serial version. This is probably due to overheads in lock coordination and thread management that is imposed on such implementation.

## VII. Conclusion and Future Work

In this work, the team implemented different variants of PRM in serial and parallel. In terms of time complexity, the contained components of standard PRM variant showed runtime gains in line with expected computational complexity flattening because of parallelization of the work. When scaled to a signficantly large enough task objective to overcome initial costs in I/O and Memory overhead, we saw up to a 40% efficiency gain from serial to parallel implementations of the standard PRM algorithm.

A substantial area of further research is in work on parallelization the ANC algorithm. As described in the Design section of this report, the team chose not to GPU-parallelize this because of inherent dependence and connection. The Python standard multithreading not only failed achieve tangible parallelization and the associated expected gains, but also additionally introduced increased overhead. We want to explore parallelization work for ANC with inherent reliance and connection of the computational work in mind. A direction of interest in using C++ and its standard multithreading capabilities to complete such exploration. We think C++ is a promising candidate to approach this as it does not subject multithreading to the same concurrency restrictions as Python.

There are many promising directions for future work, particularly in terms of parallelizing and implementing more PRM variants [15]. One interesting variant to investigate is Dynamic Roadmaps (DRM), which would have a variety of static and dynamic obstacles generated randomly, thus creating a dynamic environment for the nodes and paths to be generated in [16].This would more closely match the inspiration of autonomous vehicle parking, which has path generation in an environment that requires careful consideration of the positions and movements of both stationary and moving obstacles, which is very hard to predict [14].

It would also be interesting to investigate the effects of combining different elements of Sample Based Motion Planning and Optimization Based Motion Planning. One such way to do this would be to optimize the process of choosing the k and r values for finding nearest neighbors. This would be integrated into the PRM nearest neighbor search process for different variants to see how they compare in terms of accuracy of the shortest path and computation speed and complexity. The team would also explore optimization of space complexity for all of the previous implementations of PRM and ANC PRM as well as any other investigated variants of PRM in order to compare the differences in efficiency in these terms.

## VIII. Team Contributions

### A. Kevin Li

- Conducted preliminary research and literature review of SBMP field.
- Implemented serial ANC PRM and all neighbor connectors.
- Implemented parallel ANC PRM.
- Completed testing, benchmarking, and result analysis for ANC PRM.
- Contributed to the reporting and analysis of research conducted for this project.

### B. Nicole Neil

- Conducted preliminary research and literature review of SBMP field.
- Completed testing, benchmarking, and result analysis of Standard PRM.
- Completed report sections and LaTeX formatting: Abstract, Introduction, Related Work, Background, Design/Implementation/Algorithms, Visualization Results, Benchmarking Results, Conclusion and Future Work, Team Contributions, and References.

- Completed presentation creation and formatting.

### C. Kevin Grant Li

- Conducted preliminary research and literature review of the SBMP field.
- Implemented and tested Standard Serial Version of PRM.
- Implemented sampling components of the Standard Parallel Version of PRM.
- Completed testing, benchmarking, and result analysis for Standard Serial Version of PRM.
- Contributed to the reporting and analysis of research conducted for this project.
- Completed report sections and LaTeX formatting: Design, Visualization Results, Benchmarking Results, Conclusion, Team Contributions, and Running Code.
- Completed presentation creation and formatting.

### D. Joseph Han

- Conducted preliminary research and literature review of the SBMP field.
- Implemented and tested Parallelized Standard PRM.
- Contributed to the reporting and analysis of research conducted for this project.

### E. Avigayil Helman

- Shared and introduced RRT reference code. Completed sample and edge collision checking.
- Completed preliminary benchmarking using python time it function wrapped around relevant code blocks.

## IX. Running Code

All code is in the GitHub Project Repository, Final Folder. Visualized implementations and benchmarking simulations are contained in this folder. The relevant instructions are provided in readMe.txt files.

Our code repository with execution instructions can be found at: https://github.com/kgl2123/PRM-Project-Final/.

## References


[1] Wikipedia, "Motion planning," page Version ID: 1150007592. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Motion_planning&oldid=1150007592

[2] J. Pan and D. Manocha, "GPU-based parallel collision detection for fast motion planning," vol. 31, no. 2, pp. 187–200, publisher: SAGE Publications Ltd STM. [Online]. Available: https://doi.org/10.1177/0278364911429335

[3] A. Khokhar. Probabilistic roadmap (PRM) for path planning in robotics. [Online]. Available: https://medium.com/acm-juit/probabilistic-roadmap-prm-for-path-planning-in-robotics-d4f4b69475ea

[4] N. Amato and L. Dale, "Probabilistic roadmap methods are embarrassingly parallel," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 1, pp. 688–694 vol.1, ISSN: 1050-4729.

[5] C. Ekenna, S. A. Jacobs, S. Thomas, and N. M. Amato, "Adaptive neighbor connection for PRMs: A natural fit for heterogeneous environments and parallelism," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1249–1256, ISSN: 2153-0866.

[6] A. Short, Z. Pan, N. Larkin, and S. van Duin, "Recent progress on sampling based dynamic motion planning algorithms," in *2016 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, pp. 1305–1311.

[7] B. Plancher, "COMSBC3159_001_2023_1 lecture slides."

[8] T. Chinenov. Robotic path planning: PRM & PRM*. [Online]. Available: https://theclassytim.medium.com/robotic-path-planning-prm-prm-b4c64b1f5acb

[9] H. Choset, "Robotic motion planning: RRT's."

[10] Z. Kingston, M. Moll, and L. E. Kavraki, "Sampling-based methods for motion planning with constraints," vol. 1, no. 1, pp. 159–185. [Online]. Available: https://www.annualreviews.org/doi/10.1146/annurev-control-060117-105226

[11] P. Abbeel, "Sampling-based motion planning."

[12] O. Elmofty. What is sampling-based motion planning? [Online]. Available: https://medium.com/@oelmofty/what-is-sampling-based-motion-planning-a693a534a2a8

[13] H. Choset et. al., "Probabilistic roadmap path planning."

[14] E. Frazzoli, M. A. Dahleh, and E. Feron, "REAL-TIME MOTION PLANNINAGIAAF2o00r0 4056 AGILE AUTONOMOUS VEHICLES."

[15] P. Leven and S. Hutchinson, "Toward real-time path planning in changing environments."

[16] J. P. van den Berg and M. H. Overmars, "Roadmap-based motion planning in dynamic environments."