Efficient Quadtree Configuration Space Representation in Path Planning

1st Akhil Avula
Dept. of Electrical and Computer Engineering
Los Angeles, CA
aavula@g.ucla.edu

1st Aidan Cookson
Dept. of Electrical and Computer Engineering
Los Angeles, CA
aucookson@g.ucla.edu

1st Calvin Chang
Dept. of Electrical and Computer Engineering
Los Angeles, CA
calvinchang33@g.ucla.edu

1st Daniel Truong
Dept. of Electrical and Computer Engineering
Los Angeles, CA
dktruong@g.ucla.edu

***Abstract*—Representation of the configuration space is a critical problem in the path planning pipeline. As a result, past works struggle to showcase their search algorithm for start to goal tasks because the efficiency of the mapped configuration space is a limiting factor. A common theme that soon follows this issue is the complexity of the environment and how to handle dynamic obstacles. This paper demonstrates a pruning heuristic focused approach to quadtrees that can give a more efficient representation of the configuration space. The new representation is tested with path planning algorithms, A^* and D^* lite, as well as with dynamic obstacles. The results show that quadtrees with the heuristic can be an appropriate mapping for complex environments.**

I. Introduction

Robots today are pervasive in daily life due to recent technological advancements. Concerning the problem of finding a route from point A to point B, path planning is a basic ability that is very useful in applications such as self driving, unmanned aircraft vehicles, and unmanned underwater navigation. Many algorithms [1] have been developed to find paths optimally and safely in known environments.

By representing the configuration space as a traversable set of states, a graph search algorithm, such as A* [2], can find the shortest path from our starting position to goal position. However, it is often difficult to transform the configuration space to a graph. Depending on the resolution of the graph, the path can be very close to the optimal solution, but the program will have greater memory and computational time requirements. In order to effectively avoid obstacles in real-world environments, the system must use new data to reprocess the configuration space quickly and efficiently.

This paper introduces heuristics to prune nodes from the configuration space represented as a quadtree data structure. Usually in path planning, the optimal path is close to a straight line from the start node to goal node if the obstacles are small compared to the configuration space. Additionally, obstacles surrounding the start node and goal node will most likely require extra attention. Under most conditions, these heuristics will not affect the optimality of the graph search algorithm but will improve space complexity.

II. Background

A. Path Planning

For path planning in a graph representation, the problem can be translated to a graph search problem, where a graph is the set of vertices and edges, $G = \{V, E\}$. If each edge $e \in E$ in the graph is assigned a cost, c_e, where $c_e \geq 0$, the objective is to get from s_{start} to s_{goal} by finding an optimal path $P \subseteq E$, which is a set of chosen edges that connects the two states. The cost of the path is defined as $\sum_e c_e$, for all $e \in P$. The optimal path P^* is the path such that no other path has a lower cost, $P^* \leq P$, for all possible P.

B. Configuration Spaces

A subproblem of path planning is finding a way to represent the configuration space in such a way that an algorithm can easily find a feasible and optimal path to complete the planning task. The configuration space is the entire space of configurations, or states, a robot can take. The configuration space is defined as C. For path planning, the configuration space can be displayed as a map consisting of obstacles and free states. The subset of obstacles in the configuration space C_{obs} and the free space C_{free} are both mutually exclusive sets such that $C_{obs} \cup C_{free} = C$. The starting and goal state are defined as $s_{start} \in C_{free}$ and $s_{goal} \in C_{free}$. The entire path P also needs to be in C_{free}, meaning that all edges never overlap with C_{obs}.

Algorithms like probabilistic roadmap (PRM) [3], rapidly random exploring trees (RRT) [4], and various triangulation methods [5], [6] can represent the configuration space. These algorithms turn the configuration space into a set of feasible states and traversable paths between the states, all represented by nodes and edges in a graph. PRM and RRT use stochastic methods to sample possible nodes in the configuration space and extrapolate nodes based on the nearest existing node.

Triangulation methods leverage the convexity of triangles to draw straight edges using knowledge of the obstacles.

C. Quadtrees

Quadtrees [7] is an algorithm that recursively subdivides the configuration space into regions whose size depends if an obstacle is in its respective region or not. The result is a multi-resolution partition of the space where smaller regions are allocated at the edges of obstacles and larger regions are allocated at larger spaces containing either completely free or occupied space. Although not originally used for path planning, there has been research applying quadtrees to the path planning problem [8]–[11] due to its efficient allocation of nodes. The configuration space needs to be size $2^n \times 2^n$ due to the subdivision of squares.

A quadtree is made up of gray, white, and black nodes. These nodes represent a square region of varying sizes of the configuration space. A gray node means that there is both free space and occupied space present in the region it represents. A white node represents a region of all free space and a black node represents a region of all obstacle space. A simple example of an implementation of quadtrees is presented in figures 1, 2, and 3. The key rule to implementing quadtrees is that the gray nodes will be continuously split until a minimum size is reached, giving them 4 children until there are no gray nodes remaining.

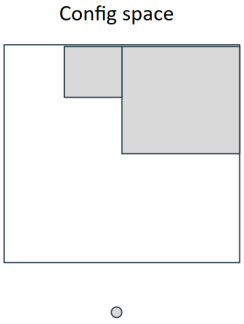

Fig. 1. First quadtree node represents the entire configuration space. The gray node splitting is expanded into 4 smaller squares, each representing a child of the original gray node.

Despite quadtrees providing a way of dividing the configuration space around obstacles, the method has shortcomings that can still be addressed [9]. The main issue with quadtrees is that the nodes are created for every part of the map, even if the agent's predicted path does not need to go to certain regions. This makes the quadtree method sub-optimal because more memory is being used for unnecessary nodes that the agent will never need to visit. Figure 4 shows how a configuration space can be overly represented on a vast unstructured environment.

The quadtree algorithm gives an overly accurate description of the configuration space because sections of the map are unreachable, and it is unlikely that the agent will need to visit every single node, depending on the start and goal nodes. This

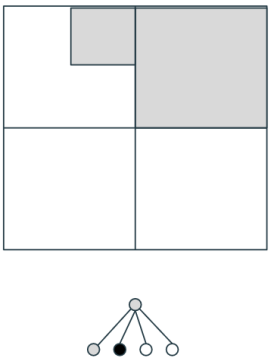

Fig. 2. Gray node splits, creating four children, each child representing a smaller square inside the parent square.

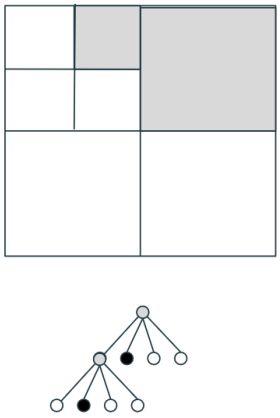

Fig. 3. Gray nodes continue to split until none of the leaves of the tree are gray. Algorithm is finished.

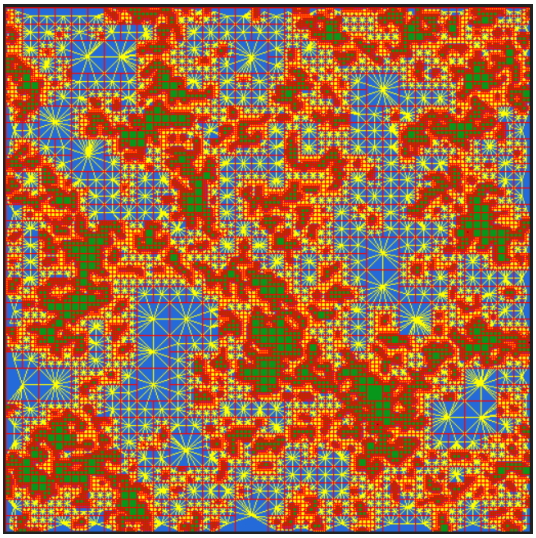

Fig. 4. Quadtrees on a Dense Environment (26000 nodes). The green land represents the obstacle space while the blue water represents the free space. Each red square represents a leaf in the quadtree.

may also add complexity when handling dynamic objects and increase the likelihood of a collision.

Hence, the goal of this project is to improve upon quadtree representation of a configuration space, using a pruning a heuristic. The goal is to remove the unnecessary nodes to reduce memory costs and ensure that the agent, a holonomic robot, has an optimal path from the start to end node, while avoiding dynamic and static obstacles.

III. Design Implementation

A. Quadtrees to Graph

Quadtrees are implemented using a recursion function to continue splitting until the leaf is not gray anymore. For path planning, only the leaves of the quadtree are necessary for the robot to traverse the configuration space. For traditional graph search algorithms to function, the quadtree graph must be converted to a new structure where neighboring leaves are connected in the graph. Thus, Samet's Quadtree neighbor finding algorithm [12] is used with a graph search algorithm, A* or D* lite.

B. Pruning Algorithm Heuristic

Heuristics of path planning are used to prevent the quadtree algorithm from allocating unnecessary nodes. The heuristics operate based on two assumptions regarding the configuration space:

- The optimal path does not stray too far from the theoretical straight line path from start to goal if obstacles are small compared to the configuration space.
- Obstacles around the start and goal node need to be determined to improve chances of finding a feasible path.

A threshold to both of the above heuristics is assigned, t_1 and t_2. The start and goal node positions are defined as pos_{start} and pos_{goal}. A quadtree node with position pos will only be split if:

$$\begin{aligned} dist(pos, pos_{start}) + dist(pos, pos_{goal}) &\leq t_1 \\ dist(pos, pos_{start}) &\leq t_2 \\ dist(pos, pos_{goal}) &\leq t_2 \end{aligned}$$

where $dist$ can be defined as the L2-norm. These thresholds are defined from the map size, $2^n \times 2^n$.

$$\begin{aligned} t_1 &= \sqrt{1.7} * 2^n \\ t_2 &= \frac{2^n}{2} \end{aligned}$$

C. Dynamic Obstacles

Dynamic obstacles on the quadtree are also implemented and tested. The obstacles randomly move on the nodes of the quadtree rather than smoothly on free space. This is to test if the pruning heuristics will effect dynamic path planning. This is a proper approximation for continuously moving obstacles because in that case, the positions of the obstacles will likely be truncated to the nearest node position for path planning.

IV. Experimentation

For testing the implemented pruned quadtrees, appropriate environment and search algorithms were implemented to provide quantitative benchmarks. Due to the particular shortcomings pruned quadtrees hopes to solve outlined in the Quadtree section II-C, the environment was implemented with Perlin Noise to model those problems. The two algorithms that were most suitable to accomplish the search task were A* and D* lite.

A. Environment Generation

Environment generation was accomplished using Perlin Noise, a type of gradient noise invented by Ken Perlin in the 1980s [13]. Perlin Noise allows for fine tuned control of the overall denseness of obstacles and characteristics of the map, and is an ideal way to flexibly generate maps with varying sizes of obstacles at different frequencies. Using this technique, naval environments can be simulated, similar to previous works. [14], [15]

B. Path Planning: A*

A* is a simple baseline path planning algorithm that is optimized for planning single start to goal tasks. As a function, A* takes a graph input consisting of its nodes and edges to output a path found by A*. The optimality of A* is dependent on its cost function:

$$f = g + h.$$

Where h is the distance from current node to goal and g is the cost-so-far from the start node to the current node. The sum results in f, the evaluating cost function. Behaviorally, A* will prioritize the path closest to the goal.

One drawback of A* is that it does not handle dynamic obstacles well. A* does not cache any node information, so when dynamic obstacles are added it must recalculate the path from scratch at every time step. This is sub-optimal, but can be avoided by using an algorithm called D* lite.

C. Dynamic Path Planning: D* lite

D* lite draws its inspiration from LPA* [16] in that it calculates a key for every node visited. This key is a 2-dimensional vector:

$$key = \begin{bmatrix} min(g(n), rhs(n)) + h(n, goal) \\ min(g(n), rhs(n)) \end{bmatrix}$$

where

$$rhs = \begin{cases} 0 & \text{if } s = s_{start} \\ \min_{s' \in Pred(s)}(g(s') + c(s' + s)) & otherwise \end{cases}$$

and g is the cost-so-far of the node, similar to the A* algorithm. The rhs of a node is a one-step lookahead of the g-value of a node, a greedy calculation to find the minimum estimated cost-so-far. This key represents the priority for each node in a priority queue, with the first element compared first, and the second element compared to break ties.

For populating the priority queue, the highest priority node is expanded, and then its neighbors are added to the queue. A

node is locally consistent when the g and rhs of a node are equal, meaning that its g-value is the minimum, consistent with its predecessor's g-values. A node is labeled locally inconsistent when $rhs \neq g$, which needs to be resolved. If $g > rhs$, then g is set to $g = rhs$ to resolve the inconsistency and the node is removed from the queue. If $g < rhs$, g is set to $g = \infty$ and the node is added back to the queue if it is still locally inconsistent. For the updated nodes, the neighboring vertices must also be updated, since the cost-so-far for those vertices will also change.

D* lite can adapt to changing environments, namely, changing edge costs, since it only implements local changes, rather than recomputing the entire graph. Thus, the effect of pruned quadtrees can be tested in a dynamic environment.

V. Results and Discussion

A. Pruned Quadtrees

The time and amount of leaves of the pruned quadtree implementation are compared with the regular quadtree implementation over 100 randomized maps in Table II. Since the pruning heuristic is dependent on the position of the start and goal node, 3 different configurations are tested, all with the same start node position, but different goal node positions. The maximum resolution quadtree node is also set to 8 pixels and the map size to 512×512.

Goal Node	Implementation	Avg.Time (s)	Avg. num. of leaves
N/A	Unpruned	0.2395	1124.16
[336,400]	Pruned	0.2516	985.49
[480,225]	Pruned	0.1834	747.39
[496,81]	Pruned	0.1378	561.47

TABLE I
TIMING AND MEMORY RESULTS OF PRUNED VS. UNPRUNED WITH START NODE AT [33, 95]

Figure 5 and 6 show a visual comparison of what the configuration space on a map looks like before and after pruning with the goal node at [480,225] and start node at [33,95].

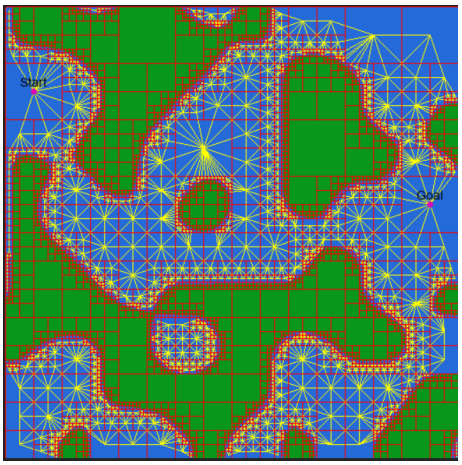

Fig. 5. Initial Quadtree Representation of Configuration Space

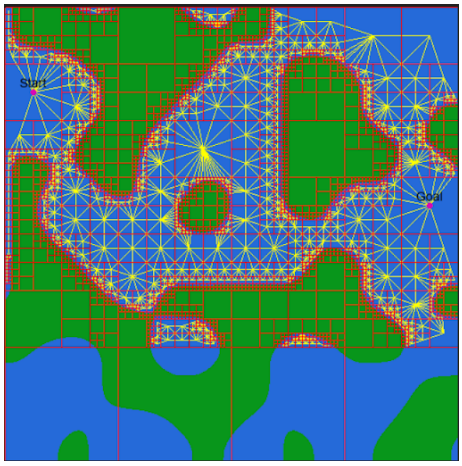

Fig. 6. Pruned Quadtree Representation of Configuration Space

B. Path Planning: A*

The A* algorithm is implemented on a static map and the number of visited nodes and the number of times the pruning makes the path planning task infeasible (cuts off needed nodes to find a path) are compared. Figures 7 and 8 show an example of the pruning algorithm working as expected. The pruning in this example does not affect the optimal path. However, there are some cases where the pruning does affect the optimal path, which are considered failed cases. An example of this can be seen in Figures 9 and 10 in the Appendix. It is noted that these failures are because the path needs to stretch far in a different direction to the goal. All of the failures occur when the goal is not at a corner of the map. It is believed that this happens because the first threshold allows further nodes to be split due to a closer proximity to start and goal nodes. However, when the goal node is not in the corner, the nodes in the far corner away from the goal node are not expanded, even if they are relatively close to the starting node.

Goal Node	# failed cases	Implementation	# nodes visited
[336,400]	0	Unpruned Pruned	242.22 234.22
[480,225]	10	Unpruned Pruned	360.62 251.5
[496,81]	0	Unpruned Pruned	245.9 245.9

TABLE II
A* RESULTS OF PRUNED VS. UNPRUNED WITH START NODE AT [33, 95] AVERAGED OVER 100 RANDOMIZED MAPS

C. Dynamic Path Planning: D* lite

For dynamic obstacles, the goal is to see if the pruned quadtree will enable D* lite to visit less nodes than if the quadtree were not pruned. On a map of size 512×512, 30 obstacles are randomly initialized on a node of the quadtree before running D* lite. The number of nodes popped from the priority queue is counted to determine the total number of nodes visited. This number is summed for 10 randomized maps comparing pruned versus unpruned. The results show

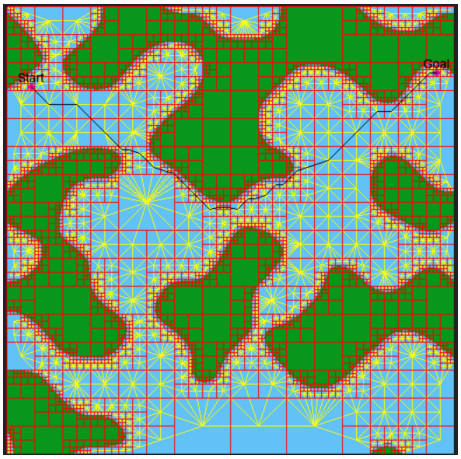

Fig. 7. A^* Path Planning on goal 1: Unpruned

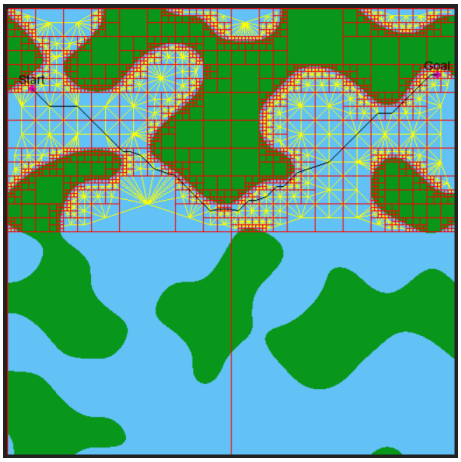

Fig. 8. A^* Path Planning on goal 1: Pruned

that D* lite visits less nodes when the quadtree is pruned. This may be because less nodes are added to the priority queue when the obstacles move due to less nodes existing in the first place. Screenshots of the D* lite algorithm working can be found in Figures 11 and 12.

Implementation	# nodes visited	% decrease of nodes visited
Unpruned	23508	32%
Pruned	16037	

TABLE III
D* LITE RESULTS OF PRUNED VS. UNPRUNED WITH START NODE AT [33, 95], GOAL NODE AT [336,400] OVER 10 RANDOMIZED MAPS

VI. CONCLUSION

Path planning algorithms play a crucial role in modern industries such as industrial robotics and autonomous systems, so it is important they are robust, safe, and efficient. A quadtree is a data structure that has the potential to significantly reduce memory allocation when representing the configuration space for path planning problems. To further improve allocation, a set of heuristics were implemented to prune superfluous nodes from the quadtree graph. On a static map, the pruned graphs took less time and resulted in less leaves on average. For path planning on a static map, the results were less clear. The pruning does not typically affect the optimal path, however sometimes the pruning removes the only feasible path. Using D* lite with dynamic obstacles, the pruning results in a 32% reduction in nodes visited, although the pruning still has the same drawback as with the static maps.

In future work, the plan is to continue optimizing the heuristic functions to reduce the number of failure cases. The minimum node size and heuristic thresholds can also be adjusted, based on properties of the generated map such as minimum choke width and largest island. Since the maps are generated using the same Perlin noise function to simulate real-world archipelagos, machine learning could also be applied to exploit shared features of the maps and adjust the graph generation hyper-parameters.

It would also be interesting to see a form of pruning which does not involve a hard threshold but rather a size limit based on the distances from start and goal nodes. This would still allow a feasible path but may lead to a less optimal path. This argument is related to the reasoning in [17] where the resolution is higher around the robot.

REFERENCES

[1] L. Yang, J. Qi, D. Song, J. Xiao, J. Han, and Y. Xia, "Survey of robot 3d path planning algorithms," *Journal of Control Science and Engineering*, vol. 2016, p. 7426913, Jul 2016. [Online]. Available: https://doi.org/10.1155/2016/7426913

[2] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[3] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, pp. 566 – 580, 09 1996.

[4] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., 1998.

[5] M. Kallmann, "Path planning in triangulations," 01 2005.

[6] Hongyang Yan, Huifang Wang, Yangzhou Chen, and Guiping Dai, "Path planning based on constrained delaunay triangulation," in *2008 7th World Congress on Intelligent Control and Automation*, 2008, pp. 5168–5173.

[7] R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys," *Acta Informatica*, vol. 4, no. 1, pp. 1–9, Mar 1974. [Online]. Available: https://doi.org/10.1007/BF00288933

[8] J. Vörös, "Simple path-planning algorithm for mobile robots using quadtrees," *IFAC Proceedings Volumes*, vol. 28, no. 20, pp. 175 – 180, 1995, iFAC Workshop on Human-Oriented Design of Advanced Robotics Systems, Vienna, Austria, 19-20 September. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474667017450452

[9] A. Yahja, S. Singh, and A. Stentz, "Efficient on-line path planner for outdoor mobile robots," *Robotics and Autonomous Systems*, vol. 32, 08 2000.

[10] A. Yahja, A. Stentz, S. Singh, and B. L. Brumitt, "Framed-quadtree path planning for mobile robots operating in sparse environments," in *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, vol. 1, 1998, pp. 650–655 vol.1.

[11] H. Noborio, T. Naniwa, and S. Arimoto, "A quadtree-based path-planning algorithm for a mobile robot," *Journal of Robotic Systems*, vol. 7, no. 4, pp. 555–574, 1990. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.4620070404

[12] H. Samet, "Neighbor finding techniques for images represented by quadtrees," *Computer Graphics and Image Processing*, vol. 18, no. 1, pp. 37 – 57, 1982. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0146664X82900983
[13] K. Perlin, "An image synthesizer," *ACM Siggraph Computer Graphics*, vol. 19, no. 3, pp. 287–296, 1985.
[14] B. C. Shah and S. K. Gupta, "Long-distance path planning for unmanned surface vehicles in complex marine environment," *IEEE Journal of Oceanic Engineering*, vol. 45, no. 3, pp. 813–830, July 2020.
[15] B. Gao, D.-m. Xu, and W.-s. Yan, "Framed-quadtree path planning for an underwater vehicle with the task of tracking a moving target," *Journal of Marine Science and Application*, vol. 9, no. 1, pp. 27–33, Mar 2010. [Online]. Available: https://doi.org/10.1007/s11804-010-8011-6
[16] S. Koenig and M. Likhachev, "Incremental a*," T. Dietterich, S. Becker, and Z. Ghahramani, Eds., vol. 14. MIT Press, 2002, pp. 1539–1546. [Online]. Available: https://proceedings.neurips.cc/paper/2001/file/a591024321c5e2bdbd23ed35f0574dde-Paper.pdf
[17] R. Kirby, R. Simmons, and J. Forlizzi, "Variable sized grid cells for rapid replanning in dynamic environments," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 4913–4918.

VII. Appendix

A. Code

Code can be found at this link.

B. Extra figures

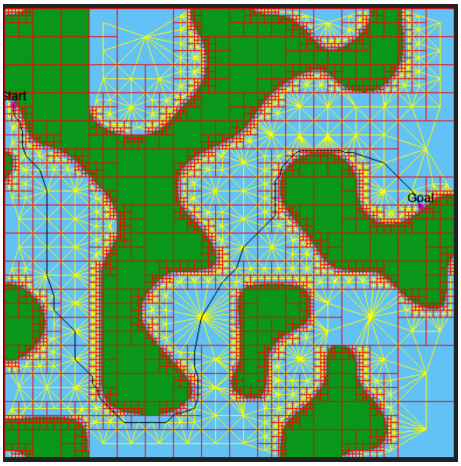

Fig. 9. Failure Example of A^* Path Planning on goal 2: Unpruned

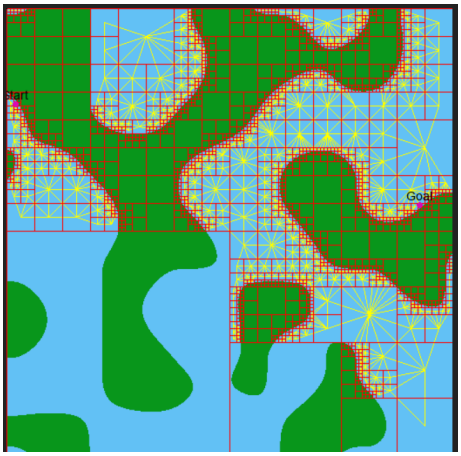

Fig. 10. Failure Example of A^* Path Planning on goal 2: Pruned

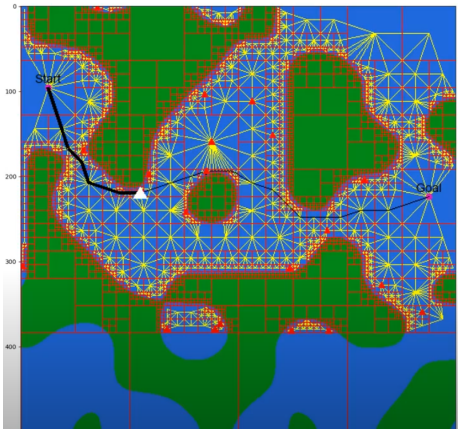

Fig. 11. D* lite Path Planning with Dynamic Obstacles: Original Path

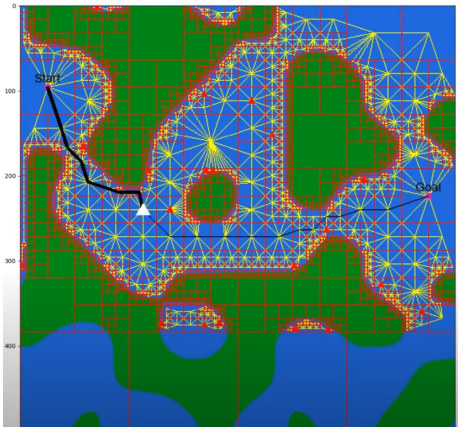

Fig. 12. D* lite Path Planning with Dynamic Obstacles: Revised Path