Enhancing Motion Planning Efficiency: A Hybrid Approach Combining A* and RRT Algorithms

Christina Cavalluzzo
Boston University College of Engineering
110 Cummington Mall, Boston, MA
chrcava@bu.edu

Abstract

Environment navigation and pathfinding are an essential part of robotics, especially within environments with obstacles in the way. The A search algorithm is one of the most widely used algorithms in pathfinding. It was developed in 1968 by Peter Hart, Nils Nilsson, and Bertram Raphael and is still extremely popular today. Another very popular motion planning algorithm is the Rapidly-exploring Random Tree (RRT) algorithm, developed by Steven M. LaValle in 1998.*

The A search algorithm can be combined with elements of the RRT search algorithm to create a more optimal motion planning algorithm. Both algorithms have advantages in motion planning, where A* excels in heuristic-driven path optimization and RRT is good at navigating complex, high dimensional, obstacle rich environments. This paper introduces an innovative hybrid approach that integrates A*'s heuristic-driven cost evaluation with RRT's random sampling capabilities. By combining these strengths, this new algorithm achieves more efficient pathfinding in structured environments. Comparing this new algorithm to a regular A* algorithm shows how the combination of the A* and RRT outperforms only A*.*

1. Introduction

Motion planning algorithms are a crucial component in robotics, used for finding the optimal path in an environment. The A* search algorithm is a graph traversal and pathfinding algorithm popular due to its completeness, optimality, and optimal efficiency. It has a structured approach, where each node is evaluated based on an exact cost from the start and a heuristic cost to the goal. Although A* is complete, it lacks the ability to effectively accept high dimensionality or perform well in obstacle rich environments due to the need for a more structured and denser graph. Because of A*'s grid-like environment, it is not good at quickly evaluating the many nodes around many obstacles. Because of this, A* is prone to being extremely time consuming.

On the other hand, the RRT algorithm is able to better work around these problems due to its use of random sampling allowing for there to be no grid and is able to be more free in complex spaces. RRT is also better suited for obstacle-rich environments due to its tree-like branching, allowing the path to bypass obstacles quickly without over exploring areas like A* would do. RRT is able to produce a path much faster than A* because of this.

This paper will explore the A* search algorithm, initializing the start and goal nodes, implementing a priority queue, and each node will have an exact cost from the start node and a heuristic cost to the goal node. Starting at the start node, the algorithm will loop through choosing the lowest cost neighboring node, checking if it is the goal, and then expanding to new neighboring nodes. Once the goal node is reached, this algorithm, like A*, will reconstruct the path by backtracking from the goal to the start and identifying the shortest path through the costs. Instead of this algorithm choosing neighboring nodes like A*, the new algorithm will randomly choose nodes, as seen in RRT algorithms. RRT generates random sample points in the search space and from this will identify the node in the existing tree closest to the goal and move to that. This algorithm will also have checks in the path from the nearest node to the new node to see if there is a collision with any obstacles, and if there is a collision a different node will be chosen.

This hybrid algorithm is original because it combines A*'s heuristic-driven path optimization and the random node generation from RRT. A* efficiently moves towards the goal because of the heuristic function, and RRT explores spaces quickly through its randomness and adaptability. Combining these two aspects will create a more efficient path. From implementation of this hybrid algorithm, it will be compared with a regular A* algorithm. The time taken to run the algorithm for the same environment will be compared. The success rate will also be compared. The environment will consist of stationary obstacles.

1.1. Related Work

The A* Path Planning algorithm has been widely used and adapted to improve navigation in complex environments, but there are still limitations, especially when it comes to long computation times and in high-dimensional, obstacle-dense spaces. There have been many attempts to strengthen the algorithm and address different challenges. Chen and Wang [2] suggested a combination of the A* algorithm with the Sparrow Search algorithm to shorten the path lengths. Although this method improved the algorithm's efficiency, it lacked flexibility in unstructured spaces. Yan et al. [8] integrated the Velocity Obstacle (VO) algorithm with A* to handle better obstacle avoidance in dynamic settings for unmanned surface vehicles. Their approach remains restricted to structured environments, limiting the scope to not be able to handle dynamic and unpredictable spaces.

[1] and [12] introduced more search directions in the A* algorithm to improve pathfinding by exploring more directions in the search process. Their process optimize pathfinding in grid-like structures but lack the ability to deal with high dimensionality. Similarly, Liu and Gong [3] explored heuristic approaches for search in rescue, and although this worked better in grid structures, it is not well suited for handling more complex environments. Seo et al. [7] worked on hardware improvements to design a shortest-path A* model that improves speed but does not address dynamic environments.

There are many works that look to improve A* to make the paths smoother and more flexible. Yang and Cai [4] looked at memory-efficient paths aiming to reduce computation while staying effective. Li et al. [5] looked at variable-step A* for faster planning in structured warehouse layouts showing the need for smoothness and flexibility. Combinations of A* with Dynamic Window Approach (DWA) [6] or DWA with RRT [9] have shown improved adaptability but remain limited in handling complex, unstructured environments.

Other notable achievements include combining A* with distance constraints [11] which improves efficiency in some scenarios by restricting the search space. Another approach is hierarchal improvement of A* where a more optimized A* algorithm was created for parking path planning in large parking lots [13]. Additionally, [10] proposes to combine RRT with the Artificial Potential Field (APF) algorithm, further strengthening the adaptability and efficiency of pathfinding in dynamic environments.

Building on these ideas, this project combines A*'s heuristic-driven path optimization with RRT's random sampling. This hybrid algorithm addresses the limitations of A* by reducing over-exploration and path rigidity, creating a more effective solution for high-dimensional, unstructured, and dynamic spaces. While this paper only explores static environments, future extensions of the developed hybrid algorithm will expand to other environments.

2. Algorithm Development

2.1. The A* Algorithm

The A* search algorithm is a widely used algorithm for pathfinding and graph traversal. It is used extensively in robotics, videogames, and artificial intelligence for finding the shortest path between a start and a goal. The A* search algorithm is an extension of Dijkstra's algorithm and improves on it by incorporating heuristic costs to estimate the total cost to reach the goal. This makes A* both complete (it will always find a solution) and optimal (it will give the best path possible).

The A* algorithm works by having two sets of nodes: open and closed. The open set is the set of nodes yet to be evaluated, and the closed set is the set of nodes that have already been evaluated. A* starts from an initial start node then calculates the total cost for each node. The total cost is estimated by

$$f(x) = g(x) + h(x) \tag{1}$$

where f(n) is the total cost of the path, g(n) is the cost from the start node to the current node, and h(n) is the heuristic cost estimate from the current node to the goal node. Nodes are chosen by order of increasing f(n), and ties are broken randomly. This process continues until the goal is reached or the open set is empty suggesting that no solution exists. By using a heuristic cost, the A* search algorithm effectively narrows down the path options to find an optimal path between the start and goal positions.

2.2. Environment

The environment used in these tests is a world of spheres called SphereWorld. This is a 2D static environment built with circular obstacles and a circular boundary. Within this environment, a gridded structure is created for the A* search algorithm with nodes and vectors between them. Within the environment, the grids surround the obstacles but do not interfere or intersect with them or the boundary. Because of this, the paths will, for the most part, only bounce between nodes and not cross through obstacles.

2.3. Modified A* and RRT Algorithm

For this paper, the A* search algorithm was modified by including elements of the RRT algorithm. This modified algorithm combines the optimal pathfinding of the A* algorithm with the exploration efficiency and randomness of the RRT algorithm. This hybrid approach incorporates random sampling to enhance the exploration efficiency while maintaining the cost and heuristic properties of A*. The goal of this new algorithm is to address the challenges of high dimensionality and more complex environments and reducing run time.

Algorithm 1: Modified A* Algorithm with RRT Elements

```
1 Add the starting node $n_{start}$ to $O$, set $g(n_{start}) = 0$, set the
  backpointer of $n_{start}$ to be empty, initialize $C$ to an empty array
2 repeat
3       Randomly pick $n_{best}$ from $O$
4       Remove $n_{best}$ from $O$ and add it to $C$
5       if $n_{best} = n_{goal}$ then
6           Return path using backpointers from $n_{goal}$
7       end if
8       Sample random points S in the search space and identify
        the node in S that is nearest to $n_{best}$ and collision free
9       for all $x \in S$ that are not in $C$ do
10          if $x$ not in $O$ and no collision exists in the path
            from $n_{best}$ to $x$ then
11              Set the backpointer cost
                $g(x) = g(n_{best}) + c(n_{best}, x)$
12              Set the backpointer of $x$ to $n_{best}$
13              Compute the heuristic $h(x)$
14              Add $x$ to $O$ with value $f(x) = g(x) + h(x)$
15          else if $g(n_{best}) + c(n_{best}, x) < g(x)$ then
16              Update the backpointer cost
                $g(x) = g(n_{best}) + c(n_{best}, x)$
17              Update the backpointer of $x$ to $n_{best}$
18          end if
19      end for
20 until $O$ is empty
```

In the modified version of A*, the algorithm begins by initializing the start with an empty open list and closed list and backpointer. The main loop selects nodes from the open list and cycles through estimating the costs and determining the best next node to travel to. The algorithm terminates when the goal is reached.

Lines 3, 8, and 10 in the modified algorithm introduce randomness from RRT. Line 3 randomly selects a node from the open list, helping to explore the space more efficiently. In line 8, the algorithm samples random points in the search space and identifies the node that is both nearest to n_best and collision free. This replicates part of the RRT algorithm by searching for new feasible nodes based on their distance to the current node. In line 10, the algorithm checks to see if the selected node x is in the open list, and whether it collides with any obstacles. If the node is not in the open list and it does not collide with any obstacles, the algorithm carries on to the next steps to set the backpointer and compute the costs.

By integrating random exploration from RRT into the cost-driven A* search algorithm, the modified hybrid algorithm can explore an environment quicker. This combination is meant to be more useful in more complex environments where the space is dynamic or complex. The random sampling and validation steps allow the algorithm to quickly progress in promising regions of an environment while ensuring optimality.

3. Results

3.1. Integrating Step 3 of Algorithm

The first test done with the modified A* algorithm that incorporates RRT elements was to include the randomness from RRT through step 3 of the algorithm. This testing was done in a static 2D environment.

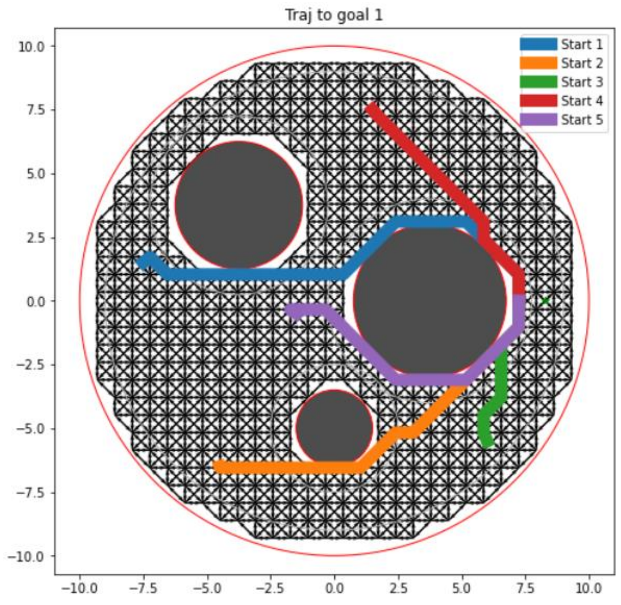

Fig. 1. Regular A* search algorithm in the environment.

Figure 1 shows the regular A* search algorithm performing in the SphereWorld environment. As shown, the paths travel from the start node to each of the goal nodes, following the grid structure and around the obstacles. The paths are as direct as the grid cells allow.

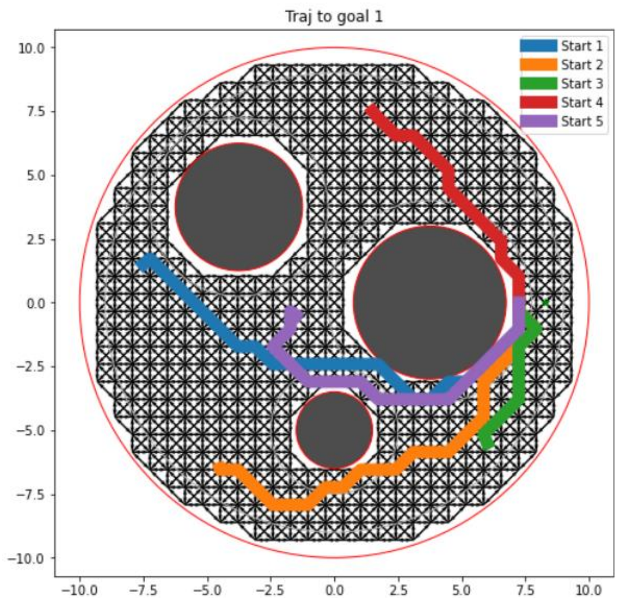

Fig. 2. Modified A* search algorithm with RRT in the environment when step 3 of the new algorithm is randomized.

Figure 2 shows the modified A* search algorithm with step 3 of the new algorithm incorporated, including elements of the randomized properties of the RRT algorithm. As seen, the paths in this run are less straightforward than the regular A* algorithm.

Because of the randomized factor, many runs of these algorithms were done and compared. For about half of the trials, the A* combined with RRT algorithm ran with less computation time. Out of 10 trials, this modified algorithm performed faster. It was also calculated that on average, the A* combined with RRT algorithm was faster. For a moderate grid structure in the environment (not too many or too few cells), the modified algorithm performed faster on average by about 0.02 seconds.

Unfortunately, the modified algorithm performed less smoothly in these tests than the regular A* algorithm. Because this testing only modifies step 3 in the algorithm, it did not take into account any path optimization to reduce cost for the path. Both algorithms were successful in reaching the goal in all trials.

3.2. Integrating Steps 8 and 10 of the Algorithm

The next testing that was done was to implement steps 8 and 10 from the pseudo code of the modified A* and RRT algorithm. Here, step 3 was kept the same as the regular A* algorithm to test different segments of the code. Again, this algorithm was applied to the same 2D static environment.

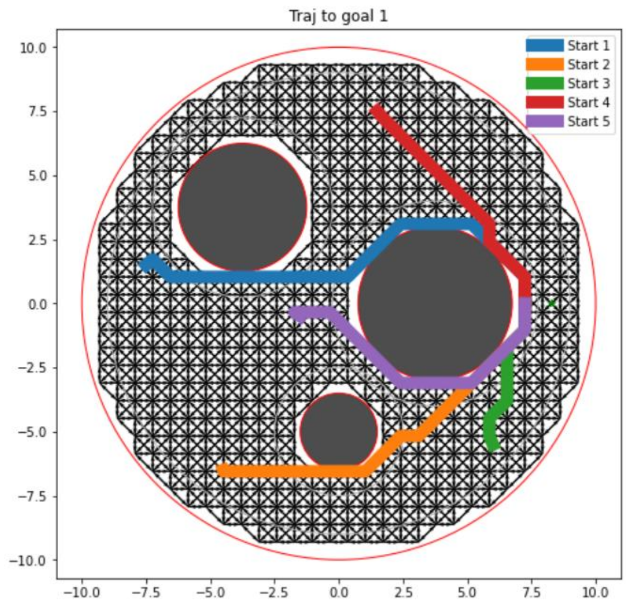

Fig. 3. Modified A* search algorithm with RRT incorporated with steps 8 and 10 of the new algorithm randomized.

Figure 3 shows the paths applied to the midified A* search algorithm with RRT implemented in steps 8 and 10. This figure is comparable to figure 1, showing how both the regular A* algorithm and the modified algorithm produce the most optimal path. Figure 3 is also seen to be much smoother and more direct than figure 2, showing that once the optimization and cost are taken into account, the algorithm will produce the most optimal path.

Although both the regular A* algorithm and the modified algorithm show similar paths and are both optimal, through repeated testing, the modified algorithm proved to be better. Over 10 trials, 7 resulted in the modified algorithm computing faster than the regular A* algorithm. Also, on average, the modified A* and RRT algorithm computed faster by about 0.03 seconds.

Through these trials, it was proven that the modified A* and RRT algorithm had a better computational time for the same result. Both algorithms were successful in reaching the goal in all trials.

3.3. Integrating the Entire Modified Algorithm

The final testing done was to integrate the entire modified algorithm without checking obstacle collision and without optimizing paths. This incorporated lines 3, 8, and 10 of the algorithm, and was done in the same 2D static environment but with different visualization as the other tests.

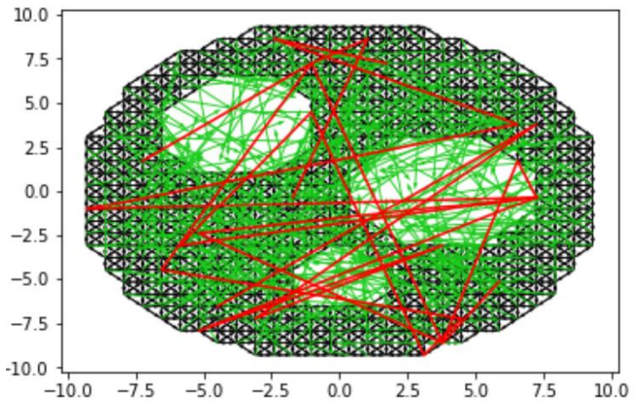

Fig. 4. Modified A* and RRT algorithm without checking obstacle collision and optimizing paths.

Figure 4 shows the modified A* algorithm which incorporates lines 3, 8, and 10 of the pseudo code. As seen, the path crosses into obstacles and does not provide a smooth path. Although these tests do not successfully plan a path from a start to a goal, they do compute much faster by half the computation time than the regular A* algorithm, showing promise for future iterations of this modified algorithm.

4. Conclusion

This paper discusses the integration of A* and RRT into a hybrid algorithm. By combining A*'s heuristic cost driven evaluations and RRT's random sampling methods, the algorithm achieves a balance between optimization and efficient path planning. Initial testing implementing randomness into step 3 of the modified algorithm proved the effectiveness of RRT's sampling in reducing over-exploration. Although this resulted in less smooth paths, it suggests the need for more testing to further reduce computation time.

Other modifications and testing was done by incorporating the RRT randomness into steps 8 and 10 of the algorithm. By merging these algorithms more together, a smoother and more direct path was produced compared to the initial testing. Computation time was also less for the modified algorithm. The randomness allows the code to skip over many of the testing and comparisons that A* requires, which is why the computation time was reduced in the modified algorithm. Because the testing was done on a smaller scale than many other options, when scaled up, the modified algorithm with A* and RRT merged will perform faster by a lot more than in these experiments.

Although the modified algorithm does not explore unstructured and dynamic environments, it can be applied to these spaces and holds optimism that it will perform well in future work. This new modified algorithm, already being more computationally less expensive and more time efficient, it holds promise to also improve upon A*'s other issues, such as enhancing flexibility, efficiency, and cost. Future work could also explore adaptive heuristics, further smoothing out the path, as well as testing in real-world scenarios.

5. References

[1] W. Bao, J. Li, Z. Pan and R. Yu, "Improved A-star Algorithm for Mobile Robot Path Planning Based on Sixteen-direction Search," 2022 China Automation Congress (CAC), Xiamen, China, 2022, pp. 1332-1336, doi: 10.1109/CAC57257.2022.10055356.

[2] Y. Chen, P. Wang, Z. Lin and C. Sun, "Global Path Planning Method by Fusion of A-star Algorithm and Sparrow Search Algorithm," 2022 IEEE 11th Data Driven Control and Learning Systems Conference (DDCLS), Chengdu, China, 2022, pp. 205-209, doi: 10.1109/DDCLS55054.2022.9858435.

[3] Xiang Liu and Daoxiong Gong, "A comparative study of A-star algorithms for search and rescue in perfect maze," 2011 International Conference on Electric Information and Control Engineering, Wuhan, 2011, pp. 24-27, doi: 10.1109/ICEICE.2011.5777723.

[4] Y. Yang and Q. Cai, "Research on the A-star Algorithm Based on Path Finding," 2023 IEEE 3rd International Conference on Data Science and Computer Application (ICDSCA), Dalian, China, 2023, pp. 199-202, doi: 10.1109/ICDSCA59871.2023.10392293.

[5] W. Li, Y. Cong, H. Du and W. Zhu, "Variable search domain improved A-star algorithm for AMR in warehouse environments," 2023 7th CAA International Conference on Vehicular Control and Intelligence (CVCI), Changsha, China, 2023, pp. 1-6, doi: 10.1109/CVCI59596.2023.10397341.

[6] X. Li, X. Hu, Z. Wang and Z. Du, "Path Planning Based on Combinaion of Improved A-STAR Algorithm and DWA Algorithm," 2020 2nd International Conference on Artificial Intelligence and Advanced Manufacture (AIAM), Manchester, United Kingdom, 2020, pp. 99-103, doi: 10.1109/AIAM50918.2020.00025.

[7] W. -J. Seo, S. -H. Ok, J. -H. Ahn, S. Kang and B. Moon, "An Efficient Hardware Architecture of the A-star Algorithm for the Shortest Path Search Engine," 2009 Fifth International Joint Conference on INC, IMS and IDC, Seoul, Korea (South), 2009, pp. 1499-1502, doi: 10.1109/NCM.2009.371.

[8] H. Yan, Q. Zhu, Y. Zhang, Z. Li and X. Du, "An Obstacle Avoidance Algorithm for Unmanned Surface Vehicle Based on A Star and Velocity-Obstacle Algorithms," 2022 IEEE 6th Information Technology and Mechatronics Engineering Conference (ITOEC), Chongqing, China, 2022, pp. 77-82, doi: 10.1109/ITOEC53115.2022.9734642.

[9] G. Wang, C. Jiang, G. Tao and C. Ye, "Dynamic path planning based on the fusion of improved RRT and DWA algorithms," 2023 4th International Conference on

Mechatronics Technology and Intelligent Manufacturing (ICMTIM), Nanjing, China, 2023, pp. 534-538, doi: 10.1109/ICMTIM58873.2023.10246738.

[10] S. Fu, "Robot Path Planning Optimization Based on RRT and APF Fusion Algorithm," 2024 8th International Conference on Robotics and Automation Sciences (ICRAS), Tokyo, Japan, 2024, pp. 32-36, doi: 10.1109/ICRAS62427.2024.10654464.

[11] V. P. Damle and S. Susan, "Dynamic Algorithm for Path Planning using A-Star with Distance Constraint," 2022 2nd International Conference on Intelligent Technologies (CONIT), Hubli, India, 2022, pp. 1-5, doi: 10.1109/CONIT55038.2022.9847869.

[12] Z. Zhang, S. Wang and J. Zhou, "A-star algorithm for expanding the number of search directions in path planning," 2021 2nd International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT), Shanghai, China, 2021, pp. 208-211, doi: 10.1109/AINIT54228.2021.00049.

[13] L. Cheng, C. Liu and B. Yan, "Improved hierarchical A-star algorithm for optimal parking path planning of the large parking lot," 2014 IEEE International Conference on Information and Automation (ICIA), Hailar, China, 2014, pp. 695-698, doi: 10.1109/ICInfA.2014.6932742.