EECS 465/ ROB 422 Final Project
Search-Based Planning
ANA*: Anytime Nonparametric A*

Yung-Ching Sun
ycs@umich.edu

Abstract

In this project, the A* algorithm and Anytime Nonparametric A* (ANA*) algorithm was implemented to solve 2D navigation problems for the PR2 robot. ANA* is one of the variants of A* algorithm that can find the suboptimal solution in each iteration to gradually find the optimal solution for path planning problem and can find the solution faster than the A* algorithm. The comparison of the solution cost vs. time for A* and ANA* was shown in Section 3. Experiments and Result. Additionally, an admissible heuristic function is crucial for A* and its variants to find the optimal path. Therefore, several choices of the heuristic function will also be compared in this project.

1. Introduction

A* algorithm and its variants are widely used heuristic-based path planning algorithms in robotics and navigation problems. A* and its variants are simple to be implemented and highly effective at searching optimal or suboptimal solutions. As a result, they are commonly applied in various real-world applications, such as path planning for autonomous vehicles, navigation in games, route guidance on maps, and robot path planning.

1.1 A* Algorithm

A* algorithm [1] is the foundation of many A* variants and is one of the most fundamental heuristic-based path planning algorithms. It is simple to implement and aims to find a path that minimizes the value of

$$f(n) = g(n) + h(n) \quad (1)$$

, where $g(n)$ is the cost from the start node to node n, and $h(n)$ is the heuristic function for node n. A* is guarantee to find the optimal solution if $h(n)$ is admissible, which means $h(n)$ never overestimate the true cost to get to the goal [2]. However, the time complexity of A* is approximately $O(b^d)$ in the worst case [3], where b is the maximum branching factor of the search tree and d is the depth of the least-cost solution. In such cases, A* can be computationally expensive to find an optimal solution. To address this, numerous A* variants have been developed in order to improve the algorithm efficiency while ensuring an optimal or a suboptimal solution can be found.

1.2 Weighted A*

Weighted A* [4] is one of the common variants of A*. It is similar to A* while sacrifices the optimality to make A* faster. Weighted A* uses the weighted heuristic function with a weighted value $\varepsilon > 1$ to expand the node with minimal

$$f(n) = g(n) + \varepsilon \cdot h(n) \quad (2)$$

, makes the algorithm bias towards the nodes that are closer to the goal and allows it to find the suboptimal solution in a shorter time. Yet, choosing the weighted value ε is a trade-off between the solution optimality and runtime. When the ε value is bigger, the weighted A* could find a solution faster while losing the optimality.

1.3 Anytime A*

Anytime A* is another variant of the A* algorithm that uses the anytime approach to enhance Weighted A*. Given a parameter ε, it can find an initial solution quickly. It will continuously find an improved solution each time it retrieves a state from OPEN set, and eventually find the optimal solution. Anytime Repairing A* (ARA*) is one of the common Anytime A* algorithm. It will find an initial solution with an initial ε, and gradually reduce the value of ε to improve the solution and reduce the suboptimality bound. However, a limitation of Anytime A* algorithms is that we need to select an appropriate initial value ε one for a good performance.

1.4 Anytime Nonparametric A*

The Anytime Nonparametric A* (ANA*) algorithm [5] is designed to resolve the limitations of traditional Anytime A* that require user to define parameters. Unlike other anytime A* variants, ANA* does not require any ad-hoc parameter to achieve good performance. It dynamically adapts the searching process by expanding the node n in an *OPEN* queue with the maximal value of $e(n)$

$$e(n) = \frac{G - g(n)}{h(n)} \tag{3}$$

, where G is the current best cost, $g(n)$ is the path cost from start to the current node n, and $h(n)$ is the heuristic estimate for current node n. This $e(n)$ value is equal to the maximal value of ε from the Weighted A* such that $f(n) \leq G$. Therefore, the node n with maximal $e(n)$ provides the greediest search. And the ANA* algorithm iteratively update the value $e(n)$ allow it to efficiently and gradually improve the solution. It is also proven that each time a node n selected for expansion from the *OPEN* queue, its $e(n)$ value bounds the suboptimality of the current best solution [5]. Thus, current best solution in ANA* algorithm will gradually decrease and found the optimal solution.

The ANA* provide the dynamic and parameter-free approach to effectively search for optimal solution in path planning problems. Unlike Weighted A* or Anytime A* algorithms, which require users to manually set the value of ε, the ANA* algorithm will dynamically adjust the ε value to obtain and improve the solution, and eventually get the optimal solution. Additionally, it is capable of finding an initial solution quickly. This is very useful for a wide range of applications. For example, the simple 2D navigation problem, such as those in this project, or in autonomous vehicle navigation, users do not have to set the parameters for planning problem. ANA* can provide a fast initial solution and continuously improve the solution until it is optimal. Overall, ANA* is useful for variety applications in real-world scenarios.

2. Implementation

2.1 Overview

Figure 1 shows the pseudocode of the Anytime Nonparametric A* (ANA*) Algorithm. ANA* is essentially built with two functions: ImproveSolution() and ANA*(). The ANA*() function is the main function of the ANA* algorithm, while the ImproveSolution() function is designed to gradually search for a better solution. In my implementation, I utilized the Node class from HW3 to represent the node configuration and a priority queue as OPEN queue to store the nodes for exploration.

2.2 *ImproveSolution*() Function

ImproveSolution() function is the key function that gradually finds a better solution for the problem. In my implementation of *ImproveSolution*(), I used a priority queue as the *OPEN* queue. Since the priority queue in Python retrieves nodes in ascending order of priority, while the ANA* algorithm wants to explore the node with maximal value of $e(n)$ value, where

$$e(n) = \frac{G - g(n)}{h(n)} \tag{4}$$

```
IMPROVESOLUTION()
 1: while OPEN ≠ ∅ do
 2:     s ← arg max_{s∈OPEN}{e(s)}
 3:     OPEN ← OPEN \ {s}
 4:     if e(s) < E then
 5:         E ← e(s)
 6:     if ISGOAL(s) then
 7:         G ← g(s)
 8:         return
 9:     for each successor s' of s do
10:         if g(s) + c(s, s') < g(s') then
11:             g(s') ← g(s) + c(s, s')
12:             pred(s') ← s
13:             if g(s') + h(s') < G then
14:                 Insert or update s' in OPEN with key e(s')

ANA*()
15: G ← ∞; E ← ∞; OPEN ← ∅; ∀s : g(s) ← ∞; g(s_start) ← 0
16: Insert s_start into OPEN with key e(s_start)
17: while OPEN ≠ ∅ do
18:     IMPROVESOLUTION()
19:     Report current E-suboptimal solution
20:     Update keys e(s) in OPEN and prune if g(s) + h(s) ≥ G
```

Figure 1. The Anytime Nonparametric A* (ANA*) Algorithm [5]

Note that I used $1/e(n)$ as the priority key to store the nodes in *OPEN* queue. This ensure that the node with the highest $e(n)$ value in *OPEN* will be explore first.

At the beginning of the *ImproveSolution*(), I retrieved the node with highest $e(n)$ value from *OPEN* queue as current node n. Then, check if the $e(n)$ value is smaller than the current suboptimality E. If it is, the current suboptimality E will be updated, and the updated E and its corresponding timestamp will be recorded into lists for later plotting. Note that to make it easier to compare and identify the convergent of the suboptimality, I only add data points where $E < 100$ into the lists.

Then, I check whether the node n has reached the goal. In my implementation, a node n is considered to have reached the goal if its Euclidean distance to the goal configuration is less than 0.125. If it has reached the goal, it will construct the path with *parents* dictionary that records the nodes visited with their predecessor and calculated the total cost of this path as the new G; then, returns the current best cost G, current suboptimality E, and the corresponding path to *ANA**().

If the current node n does not reach the goal, it will begin to expand the neighboring nodes. In this implementation, I used *for loop* to iterated expand "8-connected" space as the neighbor nodes n' with step size = 1.25. After expanding the neighbor node, check if this configuration collides with obstacles and skip the nodes that colliding with obstacles in the environment. Next, check whether the g-cost of the neighbor node $g(n') > g(n) + c(n, n')$, where $c(n, n')$ is the distance between current node n and this neighbor node n'. I used Euclidean distance to calculate $c(n, n')$, the equation is defined as

$$c(n, n') = \sqrt{(n_x - n'_x)^2 + (n_y - n'_y)^2 + \min\left(|n_\theta - n'_\theta|, 2\pi - |n_\theta - n'_\theta|\right)^2} \quad (5)$$

If $g(n') < g(n) + c(n, n')$, this neighbor will be skipped as it will not lead to a better solution. Otherwise, this neighbor node's g-cost will be updated to $g(n) + c(n, n')$, and the *parents* dictionary records the current node n as the previous node for this neighbor node n'. This *parents* dictionary is later used to reconstruct the path when the goal configuration is reached.

Then, check whether $g(n') + h(n')$ is smaller than the current best cost G, where $h(n')$ is the estimated cost from the neighbor node n' to the goal node using a heuristic function. Section 3.2 Heuristic Function will explain how I choose the heuristic function for my experiments. If $g(n') + h(n') \geq G$, this neighbor node cannot further improve the solution, so this neighbor node exploration will be ended. If $g(n') + h(n') < G$, this neighbor node n' and its corresponding $e(n') = (G - g(n'))/h(n')$ will be added to the *OPEN* queue. If the node already exists in the *OPEN* queue, its $e(n')$ value will be updated. The ImproveSolution() will be run repeatly until the *OPEN* queue is empty.

2.3 *ANA**() Function

In my implementation of *ANA**(), I initialized the current best cost $G = 10000$ and current suboptimality $E = 10000$, along with the start node and goal node for each environment and an OPEN priority queue with $e(n_{start})$ value. While the *OPEN* queue is not empty, the *while loop* will repeatly run the *ImproveSolution*() function to find and return a better solution G with the current best path and current suboptimality E. Then, it will print out the current suboptimality E and current best cost G to show the user that it has updated a new solution, and then save the new solution cost and the timestamp into two separate lists for plotting. Next, the algorithm will update the key $e(s)$ in the *OPEN* queue with the new best cost G returned from *ImproveSolution*() and prunes the nodes that if $g(n) + h(n) \geq G$. The reason that we prune the nodes if $g(n) + h(n) \geq G$ is because those nodes are not helpful for further improving the solution.

3. Experiments and Results

3.1 Experimental Environment Setup

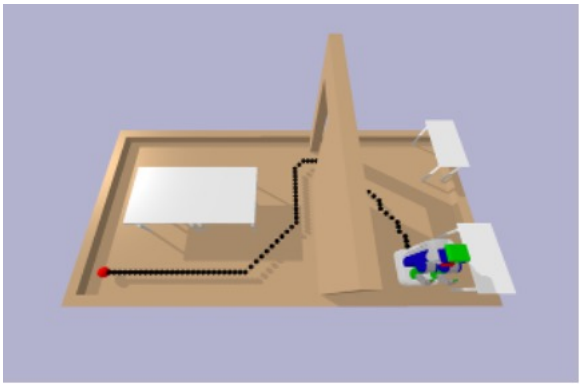
In this project, my A* and ANA* implementations were tested on five different 2D navigation problems with "8-connected" space for the PR2 robot. The navigation problems environments were built with PyBullet. All experiments were implemented in Python 3, executed on Ubuntu 24.04, and conducted on a laptop equipped with a 2.4 GHz quad-core Intel Core i5 processor and 8GB of LPDDR3 RAM.

The navigation problem environments I designed are shown in Figure 2. The red sphere marker in the simulated environment represents the position of the start configuration, while the PR2 robot is standing at the goal configuration. The path connected by black sphere markers indicates the optimal path found by the A* algorithm with the Euclidean heuristic.

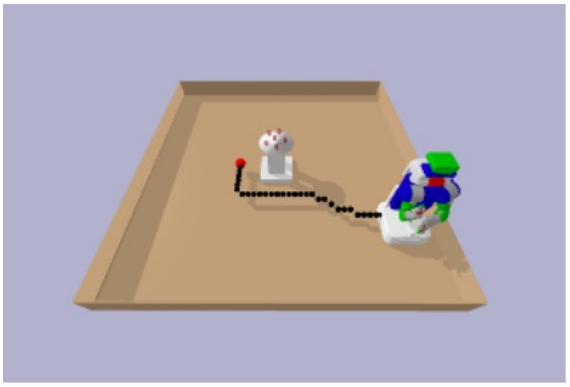
Figure 2(a) shows Environment 0, which is from HW3 and was used to test the A* algorithm. Figure 2(b) shows Environment 1, which is also from HW3 and was used to test the RRT algorithm. Since Environment 1 contains only a small round table as obstacle, I will use it as a near-obstacle-free case for comparison with other environments in subsequent experiments.

The design of Environments 3 and 4, Figure 2(c) and 2(d), was to place obstacles between the start and goal configurations. In both A* and ANA* algorithms, nodes closer to the goal are prioritized for exploration. To challenge the algorithms, I tried to place obstacles along the direct path from the start to the goal configuration. This forces the algorithm to explore nodes farther from the goal. Based on this principle, I built Environments 3 and 4 to evaluate the performance of my implementations.

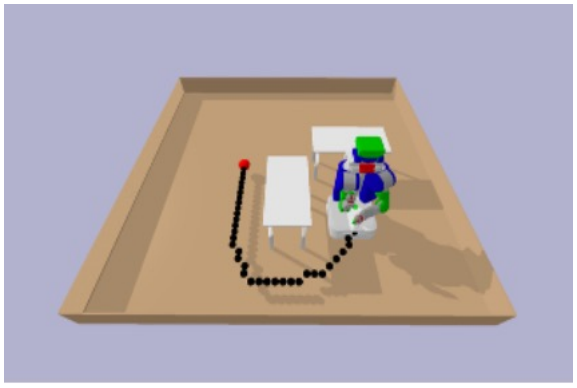
In the experiments, I set the robot movement step size as 0.125 and the threshold for robot to reach goal to be 0.125.



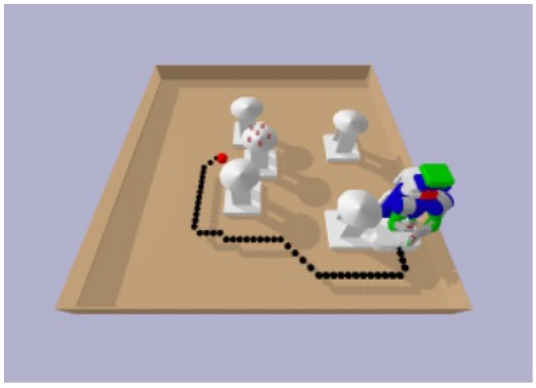
(a) Environment 0 (from HW3)



(b) Environment 1 (from HW3)



(c) Environment 2



(d) Environment 3

Figure 2. Navigation Problem Environments

3.2 The Heuristic Function

In this project, several heuristic functions were tested with both the A* and ANA* algorithms, and their performance was compared based on the time taken to converge to the optimal solution and whether they successfully achieved the optimal solution.

In the $2D$ space, the most common heuristic function is Euclidean heuristic, which calculate the straight-line distance between node n and goal g. The Euclidean heuristic function for node n and goal configuration g is defined as equation (6) and this is the heuristic function we used in HW3. In this project, the result from A* using this Euclidean heuristic will be regarded as optimal path and optimal cost.

$$h(n) = \sqrt{(n_x - g_x)^2 + (n_y - g_y)^2 + \min\left(|n_\theta - g_\theta|, 2\pi - |n_\theta - g_\theta|\right)^2} \tag{6}$$

In the real-world scenarios, the cost of translation and rotation should be different. Moreover, the unit and the numerical value of translation and rotation are not the same. Hence, my first and second custom heuristic function's rotation term is weighted by a value w_θ. My first custom heuristic $h_1(n)$ set $w_\theta = 1.5$ and the second custom heuristic $h_2(n)$ set $w_\theta = 0.6$, they are shown as equation (7) and (8). From the lecture, we know that a consistent heuristic is always admissible [3] and if the heuristic always not bigger than the true cost, the heuristic is admissible. Since $h_2(n) \leq h(n)$, $h_2(n)$ are both admissible heuristics.

$$h_1(n) = \sqrt{(n_x - g_x)^2 + (n_y - g_y)^2 + 1.5 \cdot \min\left(|n_\theta - g_\theta|, 2\pi - |n_\theta - g_\theta|\right)^2} \tag{7}$$

$$h_2(n) = \sqrt{(n_x - g_x)^2 + (n_y - g_y)^2 + 0.5 \cdot \min\left(|n_\theta - g_\theta|, 2\pi - |n_\theta - g_\theta|\right)^2} \tag{8}$$

My third custom heuristic function is a variant of the original Euclidean distance function. The translation and rotation distance calculation terms are not changed, while the rotation distance term is moved outside the square root. However, this heuristic is not guaranteed to be admissible since $\sqrt{a^2+b^2}+c \geq \sqrt{a^2+b^2+c^2}$ for $c \geq 0$ and hence $h_3(n) \geq h(n)$.

$$h_3(n) = \sqrt{(n_x - g_x)^2 + (n_y - g_y)^2} + \min(|n_\theta - g_\theta|, 2\pi - |n_\theta - g_\theta|) \quad (9)$$

The fourth custom heuristic function I tested is the octile distance function. The octile distance function is a method to calculate the distance between two nodes in a grid-based map and allow the 45° diagonal movement, it is defined as equation (10).

$$h_{octile}(n) = \max(|n_x - g_x|, |n_y - g_y|) + (\sqrt{2}-1)\cdot \min(|n_x - g_x|, |n_y - g_y|) + \min(|n_\theta - g_\theta|, 2\pi - |n_\theta - g_\theta|) \quad (10)$$

Since this project is using "8-connected" space and the robot are allowed to move diagonally at 45°, some people will use octile distance to be heuristic function. However, as the action cost function and the true cost function for my implementation is using Euclidean distance function. From Figure 3, we can see that the octile distance is larger than Euclidean distance if $\Delta x \neq \Delta y$. In this case, the octile distance is not guaranteed to be admissible and consistent; and thus, it might not guarantee the optimality for A*.

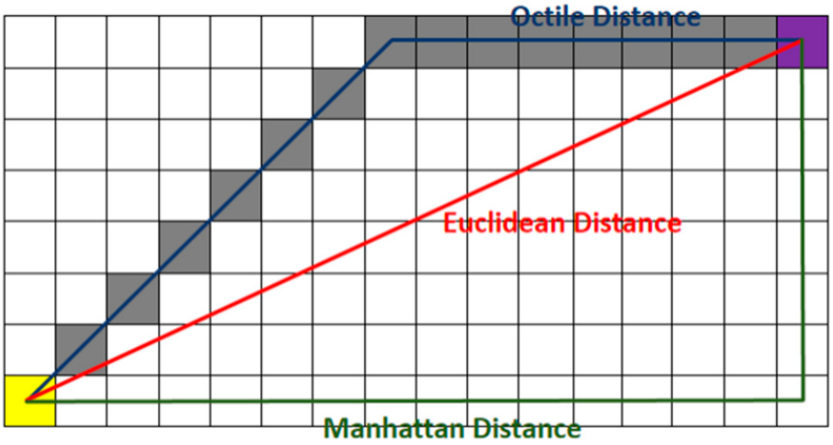


Figure 3. Illustration of octile distance and Euclidean distance [6]

In order to choose a suitable heuristic function for the demo and subsequent navigation experiments, I tested all custom heuristic functions with both ANA* and A* in the Environment 0 and compared their performance. The runtime to find the optimal solution for each heuristic is shown in Table 1, the best solution they found is shown in Table 2, and the Suboptimality vs. time is shown as Figure 4.

From Table 1, we can see that ANA* found the best solution faster than A* with all custom heuristic and the Euclidean heuristic. In addition, we can notice that the runtime of the algorithm using h_1 *and* h_3 are less than using Euclidean. This is because these heuristics are larger than Euclidean function h, so the algorithm will tend to explore the nodes that are closer to the goal, making the search process faster. For h_2, it is smaller than the Euclidean function h, the algorithm places greater emphasis on the actual cost $g(n)$, which makes it find the solution slower. Table 2 shows that both A* and ANA* with different heuristics can reach the optimal solution in Environment 0's navigation problem. From Figure 4, we can observe that the suboptimality of ANA* with heuristic h_2 converges to 1 sloest. This is because the other heuristics are all larger than the Euclidean heuristic, while Considering the admissibility of the heuristic functions, I will use h_2 to conduct the navigation experiments in other environments. Since h_2 is guaranteed to be admissible as $h_2(n) \leq h(n)$.

	$h(n)$ Euclidean	$h_1(n)$ $w_\theta = 1.5$	$h_2(n)$ $w_\theta = 0.6$	$h_3(n)$ Rot out	$h_4(n)$ octile
ANA*	289.22	278.53	306.43	238.62	297.75
A*	342.53	322.74	350.62	287.99	374.56

Table 1. Total runtime for ANA* and A* with different heuristics in Environment 0

	$h(n)$ Euclidean	$h_1(n)$ $w_\theta = 1.5$	$h_2(n)$ $w_\theta = 0.6$	$h_3(n)$ Rot out	$h_4(n)$ octile
ANA*	10.67	10.67	10.67	10.67	10.67
A*	10.67	10.67	10.67	10.67	10.67

Table 2. The best solution cost for ANA* and A* with different heuristics

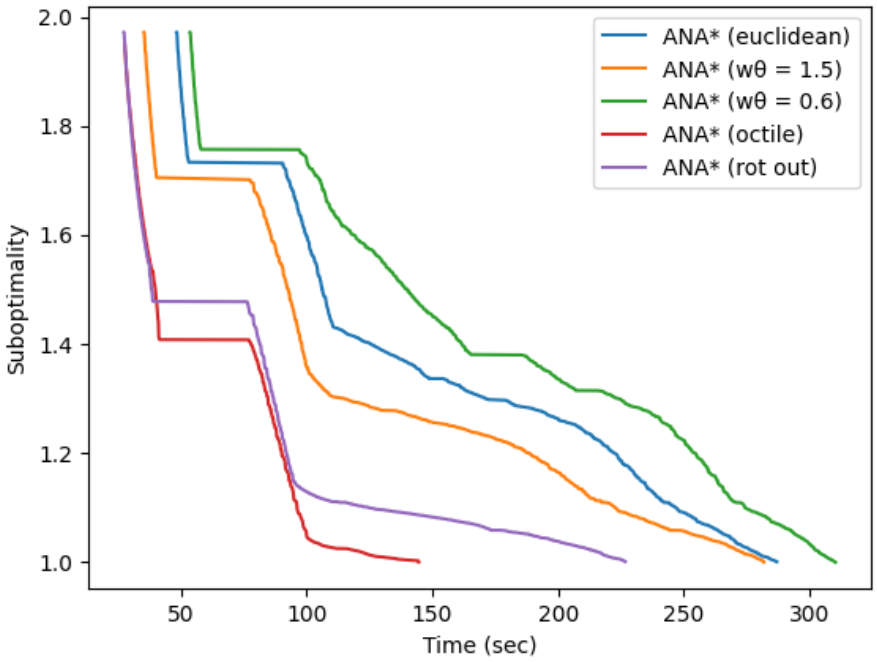


Figure 4. Suboptimality vs. Time (sec) for ANA* with different heuristics in Environment 0

3.3 Navigation Problem Results

The experimental results are shown in Table 3, Table 4, Figure 5, and Figure 6. Table 3 shows the runtime for ANA* and A* with different heuristics to find the optimal solution within each environment, it also includes the runtime for ANA* to find each suboptimal solution. We can observe that ANA* can always find an initial solution in a very short time. This is because at the beginning of the ANA* algorithm, G is very large, making the algorithm similar to the Weighted A* minimizing $f(n) = g(n) + \varepsilon \cdot h(n)$ with a large value of ε, so the search process for the initial solution is very greedy. Thus, the initial solution can be found very fast but not guarantee the optimal solution.

From Table 3, the total runtime of ANA* and A* seems to be close. However, ANA* actually found the optimal very quickly but requires additional time to converge $e(n)$ to 1 and to explore the remaining nodes in *OPEN* queue to check whether there is a better solution. This is because when the suboptimal

solution is close the optimal solution, there are many nodes in *OPEN* that need further exploration. ANA* needs to gradually examine these nodes and refine the solution to ensure there are no better solution, which is time consuming. In contrast, when the suboptimal solution is far from the optimal solution, ANA* can more easily find nodes that lead to an improved solution, so it requires less time. This can also be observed in Figure 5, where ANA* reached the optimal solution much faster than A*, although it takes some time to end the algorithm.

Algorithm (Heuristic)	Environment 0	Environment 1	Environment 2	Environment 3
ANA* (Euclidean)	**50.66**/ 277.12	5.11/ **40.09**/ 102/25	6.36/ 40.52/ **72.58**/ 91.56	10.75/ **209.36**/ 262.73
ANA* (Custom h_2)	**52.43**/ 307.76	5.58/ **37.42**/ 135.12	4.89/ 36.52/ **85.44**/ 109.36	10.98/ 140.39/ **295.74**/ 308.71
A* (Euclidean)	299.69	114.94	88.16	300
A* (Custom h_2)	322.66	140.92	81.35	323.56

Table 3. Time (sec) for ANA* to find a suboptimal or optimal solution and A* to find the optimal solution; the bolded text is the time ANA* found the optimal solution

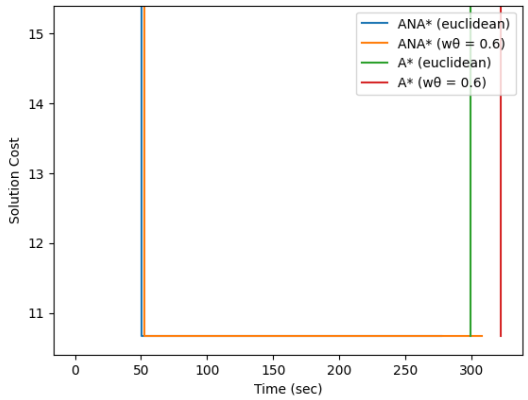
Algorithm (Heuristic)	Environment 0	Environment 1	Environment 2	Environment 3
ANA* (Euclidean)	10.67	5.08/ 5.04	6.87/ 5.64/ 5.46	6.92/ 6.82
ANA* (Custom h_2)	10.67	5.08/ 5.04	6.87/ 5.64/ 5.46	6.92/ 6.86/ 6.82
A* (Euclidean)	10.67	5.04	5.46	6.82
A* (Custom h_2)	10.67	5.04	5.46	6.82

Table 4. Optimal solution cost for ANA* and A* with different heuristics in each environment

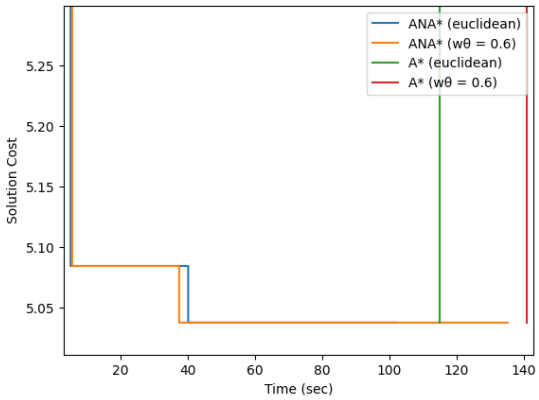
Table 4 shows that both ANA* and A* with two different heuristics can find the optimal solution in each environment. In Environment 0, ANA* found the initial solution, which is also the optimal solution within 50.66/ 52.43 seconds for Euclidean heuristic and custom heuristic, while A* needed 299.69/ 322.66 seconds to find the solution. In Environments 1, 2, and 3, ANA* found a suboptimal solution initially, but gradually improved the solution and converged to the optimal solution. It eventually found the optimal solution after a few improvements. This process is what *ImproveSolution*() function does, to gradually find a better solution, and it is guaranteed the optimal solution has been found when the *OPEN* is empty [5].

From Table 3, the total runtime of ANA* and A* seems to be close. However, ANA* actually found the optimal very quickly but requires additional time to converge $e(n)$ to 1 and to explore the remaining nodes in *OPEN* queue to check whether there is a better solution. This is because when the suboptimal solution is close to the optimal solution, there are many nodes in *OPEN* that need further exploration. ANA* needs to gradually examine these nodes and refine the solution to ensure there are no better solutions, which is time-consuming. In contrast, when the suboptimal solution is far from the optimal solution, ANA* can more easily find nodes that lead to an improved solution, so it requires less time. This can also be observed

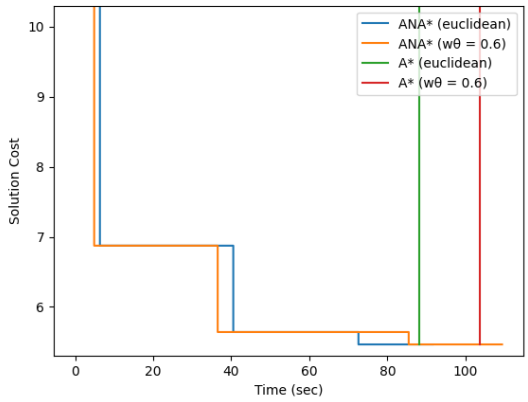
in Figure 5, where ANA* reached the optimal solution much faster than A*, although it takes some time to end the algorithm.



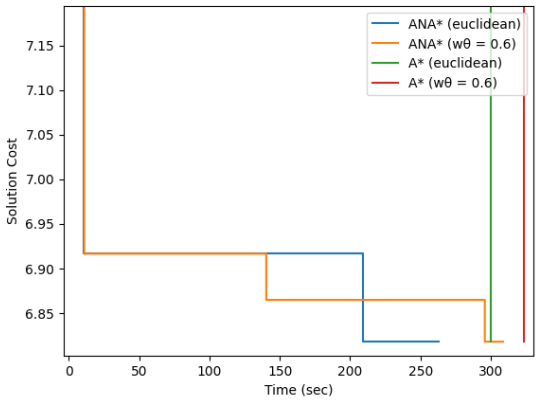
(a) Environment 0



(b) Environment 1



(c) Environment 2



(d) Environment 3

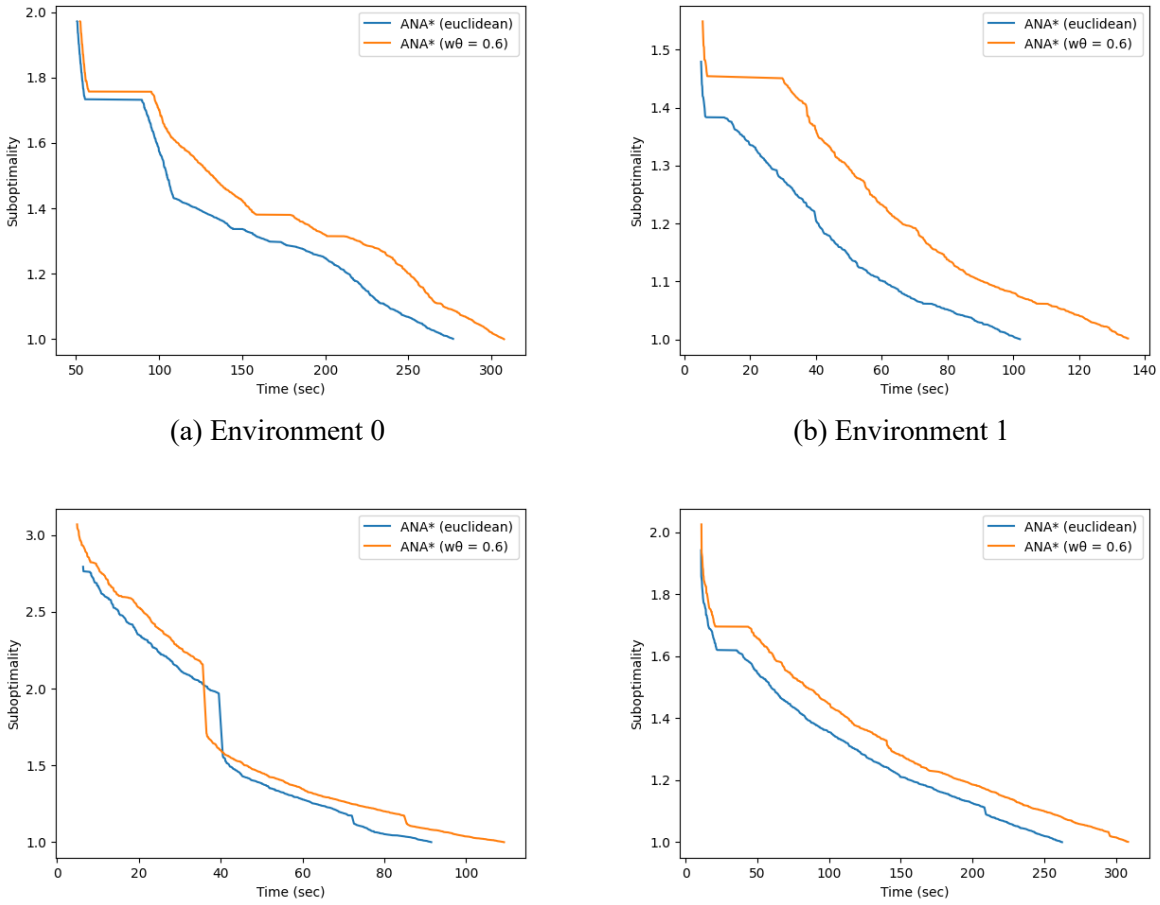
Figure 5. Solution Cost vs. Time (sec) for algorithms to solve navigation problems in Environment 0~3

Moreover, Figure 5 shows that ANA* requires less total runtime than A*. This is because ANA* adopts a highly greedy strategy at the beginning of the algorithm, it not directly finding the optimal solution be to find a suboptimal solution. It then gradually refines the solution using the $e(n)$-value bound. After finding a suboptimal solution, ANA* prunes nodes if $g(s) + h(s) \geq G$, this saves much time by avoiding to explore the high-cost nodes. Yet, A* spends time on exploring unnecessary nodes and only want to find the best optimal solution, without effectively utilizing suboptimal solutions. Hence, A* requires more time than ANA* to reach the optimal solution.

From Table3, Figure 5, and Figure 6, we can see that in different environments, the custom heuristic $h_2(n)$ with $w_\theta = 0.6$ takes longer time to find the optimal solution and has a longer total runtime. This is because $h_2(n) > h(n)$, where $h(n)$ is Euclidean heuristic, making $h_2(n)$ less greedy than $h(n)$. Thus, it requires more time to find a solution.

The experimental results from different environments show that in the more complex environments with more obstacles blocking the path, specifically Environment 0 and Environment 3, both ANA* and A* require more time to complete the searching process. However, ANA* can reach the optimal solution in less time compared to A*. We can conclude that ANA* is more efficient at finding the optimal solution in different environments. Additionally, it can be observed that when the total path length is longer

(Environment 0), ANA* require some time to converge the suboptimality and terminate the algorithm after finding the optimal solution. This is because a longer total path length has more nodes to explore, causing ANA* to spend additional time examining whether a better solution exists.



(a) Environment 0

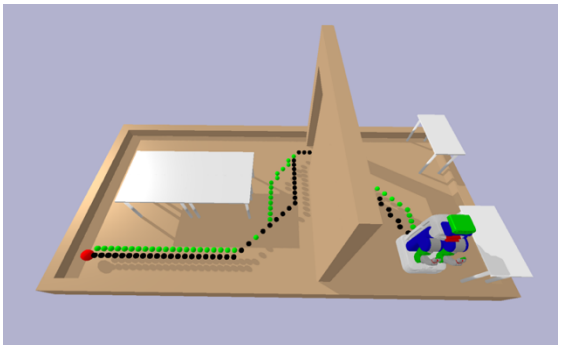
(b) Environment 1

(c) Environment 2

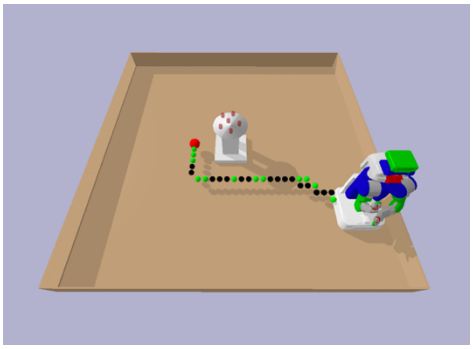
(d) Environment 3

Figure 6. Suboptimality vs. Time (sec) for ANA* with different heuristics to solve navigation problems in each environment

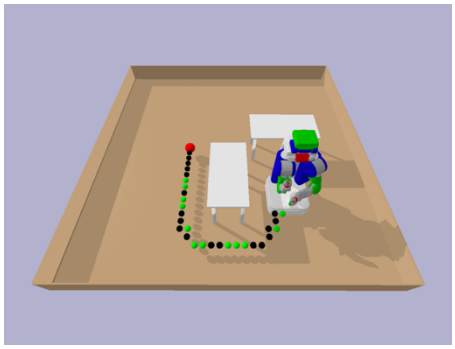
Figure 7 shows the optimal path found by the ANA* with Euclidean heuristic (green dots) and A* with Euclidean heuristic (black dots). In Environment 0 and Environment 3, the optimal paths they planned are slightly different. In Environments 1 and 2, their optimal paths are nearly identical. The difference in the optimal paths they found might be caused by their different path-searching methods and different heuristic functions. ANA* initially uses a highly greedy strategy to find a suboptimal solution and then continuously expands to find a better solution, while A* directly searches for the best solution from the beginning. Moreover, different heuristic functions will also affect the path-planning strategy. As a result, although both algorithms achieve the same optimal cost, their corresponding optimal paths can be different. This phenomenon is more obvious in complex environments with longer paths, such as Environments 0 and 3, while in simpler environments like Environments 1 and 2, the optimal paths they found are very similar.



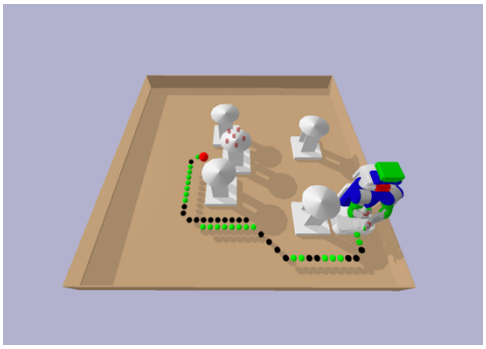
(a) Environment 0



(b) Environment 1



(c) Environment 4



(d) Environment 3

Figure 7. The optimal paths found by ANA* (green dots) and A* (black dots)

4. Conclusions

From the experimental results, we can conclude that:

1. A suitable heuristic function is important for A* and ANA* algorithms to be efficient and to find the optimal solution.
2. ANA* can find an initial suboptimal solution very fast due to its greedy strategy at the beginning of the algorithm.
3. ANA* requires less time to find the optimal solution compared to A*.
4. ANA* can gradually improve the solution using the *ImproveSolution*() function and eventually find the optimal solution.
5. ANA* is able to find the optimal solution faster than A*, even in a complex environment.
6. In environments with a longer path length (e.g., Environment 0), ANA* needs more time to converge the suboptimality.

To sum up, in this project, ANA* is more efficient than A* in all environments, as it can find the optimal solution faster, and a good heuristic function is essential for the effectiveness of A* and its variants.

References

[1] Hart, Peter E., Nils J. Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths." *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968): 100-107.

[2] Dechter, Rina, and Judea Pearl. "Generalized best-first search strategies and the optimality of A." *Journal of the ACM (JACM)* 32.3 (1985): 505-536.

[3] Dmitry Berenson. “Searching for a Path.” *Introduction to Algorithmic Robotics lecture slides* (2024)

[4] Pohl, Ira. "Heuristic search viewed as path finding in a graph." *Artificial intelligence* 1.3-4 (1970): 193-204.

[5] Van Den Berg, Jur, et al. "Anytime nonparametric A." *Proceedings of the AAAI conference on artificial intelligence*. Vol. 25. No. 1. 2011.
[6] Wang, Fujie, et al. "Research on Path Planning for Robots with Improved A* Algorithm under Bidirectional JPS Strategy." *Applied Sciences* 14.13 (2024): 5622.