EECS 498-001
Search-based Planning Final Project Report

Yuchen Wu
wuyc@umich.edu

Abstract

In this final project, Anytime Non-parametric A (ANA*) and A* have been applied to different 2D navigation problems and executed based on different heuristics to evaluate their performances. Anytime variants of Dijkstra's and A* shortest path algorithms can quickly produce a sub-optimal solution and then improve it over time, while ANA* is a variant of the anytime A* algorithm that does not require ad-hoc parameters. The algorithm iteratively reduces the weighting value (ε) to expand the most promising node and adapts the greediness of the search as path quality improves. It has been found that ANA* can reach the optimal solution faster compared to A* given the heuristic is close enough to the true cost function. Nevertheless, ANA*'s performance is more prone to discrepancies between heuristic function and true costs.*

1. Introduction

The A* path planning algorithm is a graph search algorithm that is often used in many applications of robotics due to its completeness and guaranteed optimality, given an admissible heuristic function. However, one shortcoming of the A* algorithm is that the worst-case time complexity can be $O(b^d)$, where b is the branching factor, and d is the depth of the path being searched. In various robotics applications, however, this will lead to a very long computation time for the algorithm to provide an optimal path.

To reduce the waiting time for the optimal solution, anytime A* algorithms quickly produce an initial yet sub-optimal solution and then improve over time until the optimal solution is reached. Most anytime A* algorithms are based on Weighted A*, which inflates the heuristic node values by a factor of $\varepsilon >= 1$ to trade off running time versus the solution's optimality [4]. In particular, Weighted A* repeatedly expands the "open" states that have a minimum value of:

$$f(s) = g(s) + \varepsilon * h(s) \tag{1}$$

where g(s) is the current-best cost from the starting state to the current state s, and h(s) is the heuristic function, an estimate of the cost from the current state to the goal. To apply the weighted A* in practice, for instance, in ARA* [3], the algorithm needs to initially run Weighted A* with a large value of ε to find an initial solution quickly and continue the search with incrementally smaller values of ε to improve the solution and reduce its suboptimality bound. Therefore, two parameters need to be provided before the algorithm starts: the initial ε and the ε step size, and these two parameters need to be tuned to improve the performance of anytime A* algorithms.[2]

This motivated the development of an anytime A* algorithm that does not require ad-hoc parameters. Jur van den Berg et al[5] has proposed ab Anytime Non-parametric A* (ANA*) algorithm that, instead of expanding nodes based on $f(s)$ value, the ANA* algorithm expands the open state s with a maximal value of $e(s)$, in which $e(s)$ is calculated by the following equation:

$$e(s) = \frac{G - g(s)}{h(s)} \tag{2}$$

where G is the cost of the current best solution, initially set to infinite or an arbitrarily large value. The value of e(s) estimates the possibility that this node will constitute a more optimal path. If $e(s)$ is greater than 1, which means that $g(s) + h(s) < G$, then this node will possibly lead to a more optimal solution. On the contrary, if $e(s)$ is smaller than 1, indicating $g(s) + h(s) < G$, this node is never going to improve the solution. As a result, the ANA* will continue expanding the node s with maximal $e(s)$ until the optimal solution is reached.

As ANA* trades computation time for optimality, one open question arises: to what extent ANA*'s performance will be affected by the quality of heuristic functions. This final project will investigate the effect of different heuristic accuracy on ANA* and A*'s performances. In particular, a custom metric and Euclidean distance will be utilized as two types of heuristic functions, while the Euclidean distance will be defined as the true cost function between poses. The following sections discuss the effect of different heuristic accuracy on ANA* and A*'s performance. In particular, a custom metric and Euclidean distance will be utilized as two types of heuristic functions, while the Euclidean distance will be defined as the true cost function between poses. The performance of two algorithms will be tested under three types of 2D navigation environments, and the performance will be evaluated in time to find the optimal solution.

2. Implementation

2.1. The Anytime Non-parametric A* algorithm

Algorithm 1 $ImproveSolution()$

```
while $OPEN \neq \emptyset$ do
    $s \leftarrow \arg\max_{s \in OPEN}\{e(s)\}$
    $OPEN \leftarrow OPEN \setminus \{s\}$
    if $e(s) < E$ then
        $E \leftarrow e(s)$
    end if
    if $IsGoal(s)$ then
        $G \leftarrow g(s)$
        return
    end if
    for each successor $s'$ of $s$ do
        if $g(s) + c(s, s') < g(s')$ then
            $g(s') \leftarrow g(s) + c(s, s')$
            pred $(s') \leftarrow s$
            if $g(s') + h(s') < G$ then
                Inset or update $s'$ in $OPEN$ with $e(s')$
            end if
        end if
```

Algorithm 2 $ANA*()$

```
$G \leftarrow \infty; E \leftarrow \infty; OPEN \leftarrow \emptyset;$
$\forall s : g(s) \leftarrow \infty; g(s_{start}) \leftarrow 0$
Insert $s_{start}$ into $OPEN$ with key $e(s_{start})$
while $OPEN \neq \emptyset$ do
    $ImproveSolution()$
    Report current $E$-suboptimal solution
    Update $e(s)$ in $OPEN$ and prune if $g(s)+h(s) \geq G$
```

Above is the pseudo-code for the ANA* algorithm [5]. When the algorithm starts, the current best cost G and current optimality E are initialized to an arbitrarily large number, and the starting state will be inserted into the priority queue $OPEN$. After entering the while loop, the improveSolution() function will be repeatedly invoked to find or improve the current solution. After improveSolution() returns, the main function will note the current best cost and optimality and check if the solution can be further optimized. If $E > 1$ and the $OPEN$ is not empty, an optimization is still possible, and the $OPEN$ priority queue will be reconstructed so that only nodes with its $(g(s)+h(s) \geq G)$ will stay in the updated priority queue and passed back to a new iteration of improveSolution().

This project has optimized the implementation performance of the algorithm without interfering with the optimality of the algorithm by two measures. Since checking for collision accounts for most of the algorithm computation time, a list of nodes is kept to store all the verified collision nodes. Before checking the collision of new successors, the program will first check the collision lists to determine if these nodes are previously marked collided or not. More than that, before adding a new neighbor to the $OPEN$ priority queue, the software implementation will check the $g(s)$ and $e(s)$ of the new neighbor. If the new neighbor has a $g(s)$ that is greater than the current best cost or a $e(s)$ smaller than 1, this neighbor will never improve the current solution, and the program will proceed to the next successor.

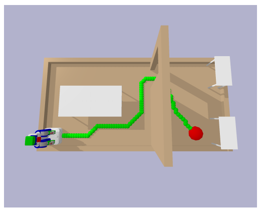
2.2. Testing Environments

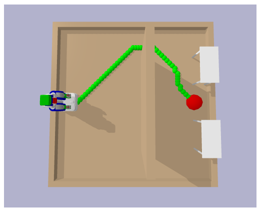
In this project, three 2D navigation environments have been proposed to test the performance of A* and ANA*. These environments are built within Pybullet simulation, and a PR2 robot will be employed as a test robot. The algorithms are implemented in Python3 and executed using a single thread on a laptop equipped with an i7-11800H CPU and 32 GB RAM. [*]

The primary design idea behind these testing environments is that they should contain obstacles that make the search for solutions non-trivial. Figure 1 below demonstrates the three test environments. Inside these figures, the robot is at its starting configuration, while the red markers indicate the goal configuration of the robot. The green series of markets suggests the optimal path from the start configuration to the goal. Both start and goal configurations are defined in the $SE(2)$ space, and the algorithms consider "8-connected" neighbors.

As the priority queues are sorted in descending order of nodes' distances to the goal, the algorithms naturally prefer nodes with a closer direct distance to the goal. Therefore, obstacles have been placed in a position that blocks the straight-line connection between the goal configuration. Because of the obstacles, the search algorithms will have to explore unfavorable nodes with a larger distance to the goal and expose the algorithms' performance under stricter conditions. Meanwhile, the environments will allow multiple feasible sub-optimal paths, allowing the ANA* algorithm to explore and reach sub-optimal solutions before finding the optimal one.

[*]The project's implementation codes can be accessed in this repository.


(a) Environment 1


(b) Environment 2

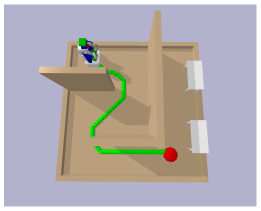

(c) Environment 3

Figure 1: Testing Environments

2.3. Choices of Heuristic function

In $SE(2)$ space, the most commonly used heuristic is the so-called Euclidean heuristic, which calculates euclidean distances between two poses in the $SE(2)$ space. In particular, given two poses $a, b \in SE(2)$, the euclidean distance $h(a, b)$ is defined by equation (3) below :

$$\begin{aligned} h(a,b) = [(a_x - b_x)^2 + (a_y - b_y)^2 \\ + (\min(|a_\theta - b_\theta|, 2\pi - |a_\theta - b_\theta|))^2]^{1/2} \end{aligned} \tag{3}$$

In this project, the euclidean distance will be considered a true cost function between two configurations as it considers movements in all joints.

In practice, as the numerical value of a_θ is often greater than a_x or b_x, a small step in the robot's rotational joint will result in a much larger change in cost function compared to a step in the robot's translation joints. As in real applications, rotations are not as expensive as represented by the behavior of the Euclidean Heuristic. Therefore, this project has proposed an alternate heuristic function that downscales the rotational component inside the Euclidean Heuristic. The alternate heuristic function, which will be denoted as Custom for the rest of this report, is given by equation (4) below:

$$\begin{aligned} h(a,b) = [(a_x - b_x)^2 + (a_y - b_y)^2 \\ + 0.5 * (\min(|a_\theta - b_\theta|, 2\pi - |a_\theta - b_\theta|))^2]^{1/2} \end{aligned} \tag{4}$$

When employing a custom heuristic function to ANA* and A*, one has to guarantee the custom heuristic function's admissibility and consistency.[1] In this case, as the square of the rotational component has been multiplied by a factor of 0.5, it is obvious that this custom heuristic will never be greater than the true cost function, the Euclidean distance. and so that the change in a_θ will play a less significant factor in the calculation of cost. The custom heuristic function is also consistent, as it is impossible to yield a negative cost by extending to any neighbor.

Nevertheless, this custom heuristic will always give a lower cost estimate than the Euclidean distance. When executing, the difference between the custom heuristic function and the true cost function will give a lower estimate to some nodes with higher costs. This situation will lead to inefficient searches by both ANA* and A*, as the two algorithms will visit nodes with inaccurate lower estimates instead of nodes with true lower costs. This situation will then provide a chance to investigate the vulnerability of the two algorithms against an inaccurate heuristic function. The results will be presented and discussed in the next section.

3. Results and Analysis

The results have been demonstrated in Table 1 and Figure 2 below. From Table 1, one can notice that both algorithms have reached the optimal solution in a shorter time when the Euclidean distance is employed as the heuristic function. As the heuristic is always giving the true cost and therefore correctly leading the way for the two algorithms, the two algorithms' efficiencies have been maximized and therefore, an optimal solution is quickly found. One may also notice that when the Euclidean distance is utilized, ANA* reached the optimal solution even faster than A*. This can be explained by the fact that ANA* is actively pruning the nodes that is having a cost that is greater than the current best solution, therefore saving time from not visiting those nodes at all.

When the custom heuristic is employed, however, ANA*'s performance has been affected significantly. Figure 2 has demonstrated that in most cases, the ANA* with a custom heuristic will take a longer time to converge to the optimal solution. When compared to A*, ANA* with a custom heuristic also needs a longer time to reach an optimal solution, regardless of what heuristic the A* has employed. On the other hand, A*'s performance has been comparatively stable. The utilization of custom heuristics does not significantly affect the time needed to find the optimal solution and indeed, the A* is always able to find the optimal solution. One can then make the conclusion that, compared to A*, ANA* is more vulnerable to the inaccuracies of heuristic functions.

The reason behind ANA*'s performance significantly depends on the quality of the heuristic can be explained by the way ANA* works. In principle, ANA* can find a suboptimal solution in a shorter period of time because it has been inflating the weighted value multiplied by the heuristic value. When the heuristic function is giving a consistently lower estimate compared to the true cost, the ANA* will be further misled by the inaccurate heuristic term. More than

Algorithm (Heuristic)	Environment 1	Environment 2	Environment 3
ANA* (Custom)	483.34	134.58	447.45
ANA* (Euclidean)	114.47	107.18	230.53
A* (Custom)	310.65	258.97	336.48
A* (Euclidean)	296.79	245.98	340.27

Table 1: Time taken in seconds for each algorithm to find optimal solution

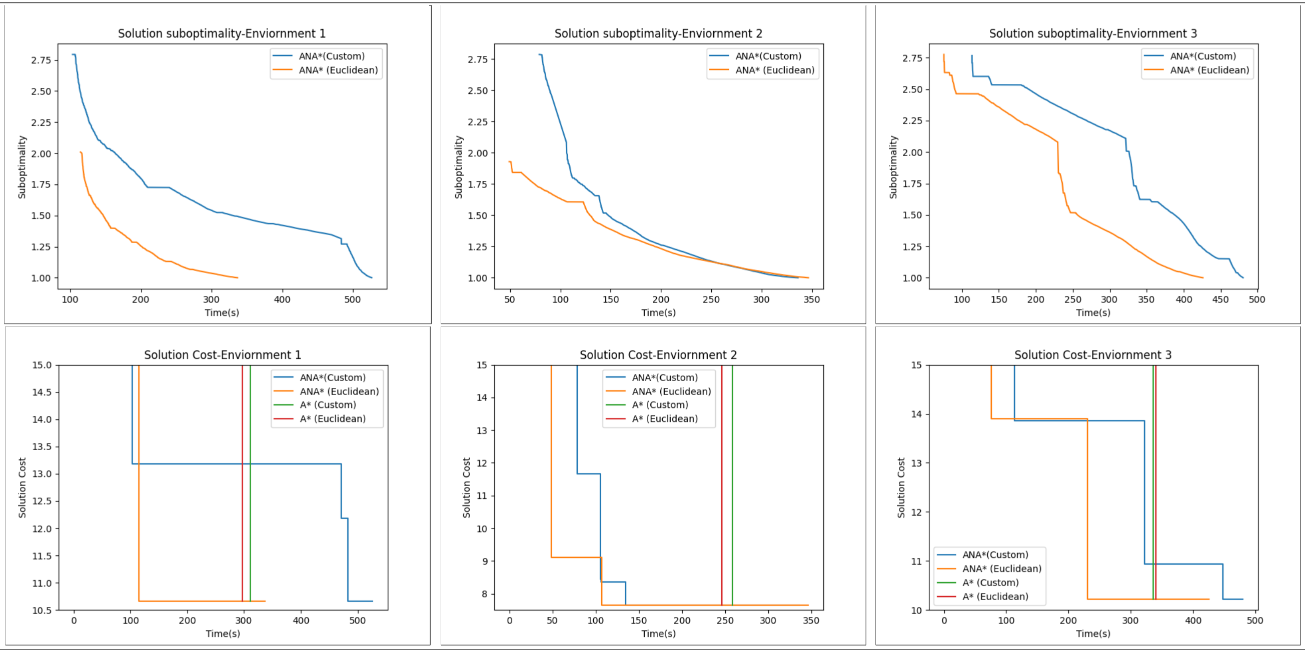


Figure 2: Qualitative results on time taken for the algorithms to reach optimality.

that, an inaccurate heuristic value will reduce the advantage brought by the pruning step of ANA*, which further affects its performance in finding the optimal solution.

4. Conclusions

Overall, given a heuristic function that is close enough to the true cost, the ANA* will outperform A* in terms of computation time to find the optimal solution. However, when the heuristic function provided does not converge to the true cost, even if the provided heuristic is consistent and admissible the ANA* will be significantly affected and requires a larger time to reach the optimal solution compared to A*. On the other hand, A*'s performance on runtime to find the optimal solution has been relatively stable against an inaccurate heuristic.

References

[1] Ariel Felner, Uzi Zahavi, Robert Holte, Jonathan Schaeffer, Nathan Sturtevant, and Zhifu Zhang. Inconsistent heuristics in theory and practice. *Artificial Intelligence*, 175(9-10):1570–1603, June 2011. 3

[2] E. A. Hansen and R. Zhou. Anytime Heuristic Search. *Journal of Artificial Intelligence Research*, 28:267–297, Mar. 2007. 1

[3] Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun. Anytime search in dynamic graphs. *Artificial Intelligence*, 172(14):1613–1643, Sept. 2008. 1

[4] Ira Pohl. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1(3-4):193–204, Jan. 1970. 1

[5] Jur Van den Berg, Rajat Shah, Arthur Huang, and Ken Goldberg. Anytime Nonparametric A*. *Proceedings of the AAAI Conference on Artificial Intelligence*, 25(1):105–111, Aug. 2011. 1, 2