# ARA* on iteratively higher resolution lattices

Tito Anammah (ooanamma), Mayur Paralkar (mpparalk)

December 2020

## 1 Problem Statement

The objective of this project was to implement an anytime planner for a Roomba robot to find a path to its docking station. The Roomba operates in a dynamic environment wherein obstacles could appear at any point in time and may obstruct the Roomba's planned path. This dynamic environment necessitates the need for a fast anytime planner that will enable the Roomba to quickly replan a path to the docking station upon the discovery of a new obstacle.

### 1.1 Planning Algorithm

Our planner builds off of the ARA* (Anytime Repairing A*) algorithm which tries to find a potentially sub-optimal plan as quickly as possible before iteratively improving upon the solution while time remains. ARA* executes a series of weighted A* searches with decreasing weights and efficiently reuses previous search efforts. In order to speed up the time taken to find an initial solution (and subsequent solutions) our anytime planner incorporates multi-resolution lattices by iterating from lower to higher resolution lattices during the search. This way, by starting the search using a low-resolution lattice graph, we cut down the search branching factor and potentially speed up the search process. However, in order to achieve a provably optimal solution (time permitting), we ensure that we iterate all the way through the highest resolution lattice.
We have two variants of this algorithm. In the first variant when switching resolutions we reset all the data structures including the open set. In the second variant however, we don't reset the open list and instead carry it over across resolutions. While this causes the algorithm to lose its optimality guarantees, It does provides a viable alternative to the first variant as it reuses previous search efforts and speeds up the subsequent searches. Also our preliminary experiments show that the solutions returned for the higher epsilon iterations of subsequent resolutions tend to be less costly. This again is as a result of reusing previous search efforts. If the planning problem is especially time constrained then the second variant of the algorithm might be preferable. The pseudocode for both variants of the algorithm are outlined in algorithm 1 and 2.

**Algorithm 1** Multi-resolution ARA*

```
1: procedure PLANNER
2:     for resolution in [res1, res2, ....  resn] do
3:         initialize ARA* data structures
4:         set epsilon to a high value
5:         computePathWithReuse()
6:         publish epsilon suboptimal path
7:         while epsilon >= 1 do                    ▷ Keep improving solution
8:             decrease epsilon
9:             computePathWithReuse()
10:            publish epsilon suboptimal path
11:        end while
12:    end for
13:    return optimal solution
14: end procedure
```

**Algorithm 2** Multi-resolution ARA* V2 (without reset)

```
1: procedure PLANNER
2:     initialize ARA* data structures
3:     set resolution to lowest resolution
4:     set epsilon to a high value
5:     computePathWithReuse()
6:     publish epsilon suboptimal path
7:     while (epsilon >= 1) and (resolution ≤ resn) do    ▷ Keep improving
   solution
8:         decrease epsilon
9:         computePathWithReuse()
10:        publish epsilon suboptimal path
11:        if epsilon == 1 then    ▷ Move to next resolution without resetting
   open set
12:            set epsilon to a high value
13:            increase resolution
14:        end if
15:    end while
16:    return optimal solution
17: end procedure
```

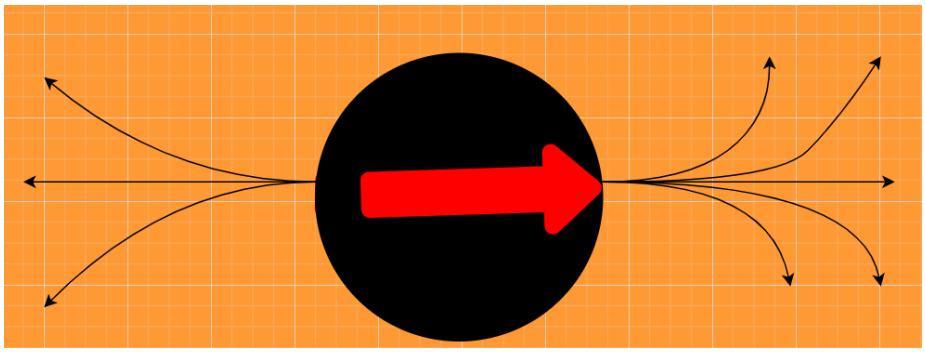

Figure 1: robot motion primitives

### 1.2 State/search representation

A state of the Roomba during search is represented by its x and y location along with its orientation. The state is therefore represented by a tuple $(x, y, \theta)$. The Roomba will be initialized to a specific start location and orientation $(x_{start}, y_{start}, \theta_{start})$, and the goal state representing the docking station will be a specific goal location and orientation $(x_{goal}, y_{goal}, \theta_{goal})$. The map will be represented by a 2D grid indicating free and obstructed locations. The planner is to return a sequence of valid steps $(dx, dy, d\theta)$ from the start state to the goal state.

### 1.3 Motion primitives

In order to make the Roomba trajectories more realistic, we defined 8 motion primitives that define valid transitions. The 8 actions are Forward, Forward Short, Forward Left Slight, Forward Left Sharp, Forward Right Slight, Forward Right Sharp, Backward, Backward Left Slight, Backward Right Slight. Refer to figure 1 for a depiction of the motion primitives

### 1.4 Lattice Resolutions

For this project we defined 2 lattice resolutions 'low' and 'high'. The high resolution includes all 8 motion primitives described earlier while the low resolution only has 4 of the motion primitives: Forward, Forward Left Slight, Forward Right Slight, and Backward. The low resolution therefore has about half the set of primitives present in the high resolution. During the low-resolution search only the low-resolution primitives are used for the most part; however, there is a "heat zone" surrounding the start and goal states within which all motion primitives are used even during the low resolution phase of the search. The heat zone is represented by a radius around the start and goal locations. This modification is done to ensure that the robot has full maneuverability when

leaving the start state and approaching the goal state as those tasks tend to be the most complex.

### 1.5 ARA* search

When running the ARA* algorithm we initialize epsilon to 5 and decrease epsilon to 3 after the first iteration, then to 1.2, and finally to 1. During the open set update phase, we prune away states whose unweighted f-values (cost to the state plus the heuristic estimate of the cost to the goal state) are greater than the cost of the best path to the goal found so far. This small modification was mentioned in the ARA* paper and could potentially speed up the search as the complexity of functions interacting with the priority queue (used for the open set) scales with its size. To acquire heuristic values for states, we ran a 2d backwards Dijkstra's search from the goal position (assuming full connectivity) and also tested with a simple euclidean distance estimate.

## 2 Results and Discussion

For our experiments, we generated 30 $256 \times 256$ unit maps with randomly generated start and goal states. Start states were located within the top 15 rows/columns while goal states were located within the last 15 rows/columns. Obstacles, represented by random 2x2 blocks, were also randomly generated. Any given unit in the grid had a 25% chance of being part of an obstacle. All 30 maps were run with two different heuristic functions.

| Table 1. Euclidean Heuristic | | | | |
|---|---|---|---|---|
| Average Cost | | | | |
| **Resolution** | $\epsilon = 5$ | $\epsilon = 3$ | $\epsilon = 1.2$ | $\epsilon = 1$ |
| Low | 446.8 | 434.8 | 385.5 | 385.5 |
| High | 385.5 | 385.5 | 385.5 | 385.5 |
| High (w/ reset) | 411.2 | 410.1 | 383.0 | 383.0 |
| Average Time To Plan (ms) | | | | |
| Low | 55.5 | 44.4 | 25820.9 | 5677.1 |
| High | 114.6 | 0.1 | 0 | 0 |
| High (w/ reset) | 141.8 | 69.8 | 57844.1 | 9121.9 |

Table 1: Average cost of and average time to generate plans found using different resolutions of lattices across different weights ($\epsilon$) using euclidean distance to goal as the heuristic function. "With reset" corresponds to resetting the open list when switching from low to high resolution. Note that this is the same as running the ARA* algorithm so the table implicitly compares the performance of our planner with ARA*

Table 1 represents the average cost of plans generated at each resolution and the average time taken to generate them. For this experiment we used euclidean

distance as our heuristic function since we wanted to assume that we had minimal time to plan a path - not enough to run a backwards A* or Dijkstra's search. As expected, the low resolution plans had worse costs than corresponding plans from high resolution lattices. Differences at $\epsilon = 1$ were small most likely because our maps were not extremely large or complex. Nevertheless, the high resolution outperformed the low resolution lattice. Finally, the high resolution outperformed the high resolution with reset for many epsilon values because it was able to return the same solution as that found by the optimal low resolution search. However, the data shows that without resetting the open state, the high resolution planning did not ever generate a plan that was better than that of the optimal low resolution plan. This explains why the high resolution planning without reset took almost no time to return the plan.

Planning times were low during $\epsilon = 5$ which was the first planning step because of the heavy weighting on the heuristic values. The $\epsilon = 3$ iteration had the lowest planning times across all iterations. Meanwhile, peaks planning times were experienced during the third iteration. Interestingly, the optimal plan was found during this iteration in both low and high (with reset) resolutions.

| Table 2. 2D Backwards Search Heuristic | | | | |
|---|---|---|---|---|
| Average Cost | | | | |
| **Resolution** | $\epsilon = 5$ | $\epsilon = 3$ | $\epsilon = 1.2$ | $\epsilon = 1$ |
| Low | 492.3 | 492.1 | 397.3 | 385.5 |
| High | 385.5 | 385.5 | 385.5 | 384.6 |
| High (w/ reset) | 422.7 | 422.6 | 400.1 | 383.0 |
| Average Time To Plan (ms) | | | | |
| Low | 47.7 | 6.1 | 118.1 | 4953.9 |
| High | 113.0 | 0.7 | 0.6 | 17302.0 |
| High (w/ reset) | 115.8 | 16.0 | 481.4 | 17749.0 |

Table 2: Average cost of and average time to generate plans found using different resolutions of lattices across different weights ($\epsilon$) using a 2D backwards Dijkstra's search as the heuristic function. "With reset" corresponds to resetting the open list when switching from low to high resolution.

Table 2 shows average planning times and average costs for plans generated using a 2D backwards search heuristic. Running the backwards search took between 4-5 seconds.

Many similar patterns can be found between the first and second tables. Planning in higher resolutions gave lower cost plans than planning in low resolution spaces. However, the average time to generate the plans increased. Planning in high resolution with reset doubled the amount of time necessary to generate a first plan and more than tripled the amount of time necessary to generate an optimal plan. Unlike with euclidean heuristics, the high resolution without reset was able to generate a better plan than the low resolution lattice at $\epsilon = 1$. Furthermore, average planning times were generally less than that using the euclidean distance heuristic.

When using euclidean distance as a heuristic metric, we found that transitioning from the low resolution to the high resolution lattice without resetting our data structures had almost no benefit. Initial plans could be quickly generated, but decreasing the weight $\epsilon$ multiplied the amount of time required to create plans. It would be best to use these heuristics and this algorithm in a rapidly changing environment which strongly limits the amount of planning time. On the other hand, the 2D backwards Dijkstra's search greatly reduced the amount of time needed to generate near-optimal plans. If either optimal plans are needed or the environment is mostly static, using this heuristic is a good alternative. Finally, not resetting our data structures did improve the optimal plan between low and high resolutions using the backwards search heuristic. Although this iteration took almost the same amount of time as planning with reset data structures, it may be worth investigating more.

# 3 Conclusion

In this paper, we ran ARA* on lattices of different resolutions with different heuristics and methods. We discovered a speedup in planning time over ARA* when running our multi-resolution planner. Incorporating low resolution lattice graphs cut down the search branch factor and sped up the search process. By judicously pruning primitives to include within the low resolution primitive set, the planner can achieve near optimal solutions in less time than it would have taken in a single resolution search.

This planner can be versatile when prioritizing either for speed or optimality. For example, if the quality of the solution is prioritized, then when searching in the low resolution phase, one may decide to initialize epsilon to a lower value. This way you could focus on more optimal solutions without a significant increase in time as the low resolution search is expedient. The ability to run the algorithm with or without resetting the open set when transitioning between the low resolution and high resolution search also provides more versatility to the planner. If the problem is especially time constrained, using the second variant of the algorithm might be preferable. Retention of the optimal low-resolution plan accelerates the second variant because it can reuse search efforts to avoid exploring costlier alternatives. The loss of optimality guarantees in the variant may not preclude good results in practice.

# 4 Github Link

https://github.com/tanammah/AutonomousRoombaPlanner/blob/main/Roomba.py