A* in Warehouse Robotics

Ryan Smith

Wentworth Institute of Technology

August 15, 2023

INTRODUCTION

Warehouse automation is a large and complex problem which involves many moving parts. Part of this involves robots, often in the form of automated movers with simple tasks such as transporting items from one location to another. Even with these robots with their seemingly simple tasks are complex machines which must take many different factors of their environment into consideration. The biggest challenge for these robots is often pathfinding, as the busy and complex environment of a large warehouse presents many obstacles to navigate while also having many different goals with varying complexities. The solution to this challenge is AI, which given the right algorithm is able to adapt to a changing environment and optimize solutions. Finding the best algorithm to use for this pathfinding challenge is no easy task, and many different algorithms have been used in the past. To narrow down the scope of this problem, this paper will be focusing on the A* algorithm and its feasibility in use with warehouse robots.

A* is a good searching algorithm for finding the most optimal path from an initial state to a goal state in a reasonable amount of time. I wanted to build on the implementation of A* we wrote earlier in the course, and warehouse robotics seemed like a sufficiently large problem to tackle without being impossible to simulate. There are a few simple differences between the sample world problem from before and a warehouse world, but there are also some more complex problems to be considered. A few of the important ones are dynamic obstacles (i.e. people, vehicles, other robots), goal prioritization, and multi-agent planning. For the purpose of this project, multi-agent planning was chosen as the more complex problem to focus on. While the other problems are also important, I decided it is most important in a warehouse environment to ensure that multiple robots can work together without getting in each other's way. As such, the goal of this project was to explore the best way to plan for multiple agents using A*.

METHOD

I. Creating Warehouse World

The first step to implementing A* for the warehouse problem was converting Sample World (from A2 earlier in the semester) to Warehouse World. This required modifying the world txt files to simulate a small warehouse, then modifying the states and nodes used to account for the world changes. The new world files involved a robot and blocked cells just as Sample World did. The difference was with the goals: the samples had become boxes, and there was a new goal added which was shelf locations. The robot now had to move to a box, pick it up, move to an empty shelf, and deposit the box. The following figure shows an example of a small world that was used for

early testing, involving one robot (@), three boxes (*), and 2 columns of shelves, with 3 being empty (|) and 3 being full (#).

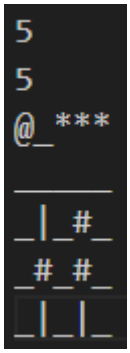

Figure 1: Small World 1 (1R)

Though these changes seem simple, the resulting coding changes meant a much more complex search space and an immediate increase in algorithm runtime. Since each box needs a shelf, the number of goals is essentially doubled. The robot also had an additional action, since "sample" had become "pick up" or "drop". In addition, I chose to store the blocked cells in each state rather than in a fixed variable. It would decrease the space complexity of the algorithm to store blocked cells separately from states, but I felt it made for a more accurate representation of a warehouse which is more important than cutting corners to save on runtime. There are many different ways the warehouse could have been represented, but I felt like the problem I chose was a good balance. For example, boxes could have been given specific locations they must be brought to, but this would restrict the algorithm and I wanted to give the robots more freedom to see which path they would consider optimal.

II. Multi-agent Planning

The next step was to consider planning for multiple robots, since the Sample World problem was only viable for one robot. This would require tracking multiple robots (both their location and box holding status) in each state, as well as adjusting the expansion heuristic functions to account for multiple agents. My goal was to have a functioning algorithm for the following medium sized world:

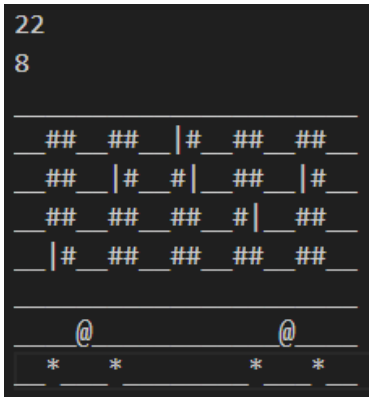

Figure 2: Medium World 1 (2R)

My initial solution was to simply iterate through all robots and generate all possible actions for each robot. Then for determining heuristics, I took the Manhattan distance to the closest goal (a box or shelf depending on if a box was being held) for each robot, and returned the sum of the distances

for the value of h(n). This initially worked, but it was slow. Adding a second robot to the small world in Figure 1 increased the runtime from 0.5 second to 20 seconds, which is a 4000% increase. Adding a third robot increased the runtime to 350 seconds, and the medium world did not complete in a reasonable amount of time. While the solutions presented were optimal, the time was too much for the algorithm to be considered admissible.

III. Multi-action Nodes

The issue with my initial approach was the fact that each node could only handle a single action. If a single node is considered a single time step, then this meant only one robot could move at a time. Not only would this be an issue for a real world application of this algorithm, but it also increased the depth of the goal state. My solution was to use multi-action nodes, meaning each robot could act once per node. The change to the nodes was simple, since all it required was storing a list of actions rather than a single one. The complicated part was the expansion function. As before I generated all possible actions for each robot, but to generate the new nodes I had to recursively generate every possible combination of actions for any number of robots. This is a trade off; while the depth of the solution path is decreased, the expansion algorithm would now take exponentially longer the more robots there were. Regardless of the effect on time complexity, this is a more accurate representation of how the robots should behave

RESULTS

I. Main Results

To evaluate the results of my A* implementation, I looked at the path produced, and the inputs (# of boxes and robots) vs. the time it took the algorithm to run. My primary goal, a functioning algorithm for the medium sized world in Figure 2, was a partial success. The algorithm was able to find a path, but it took far too much time to run.

```
Robot 1 (4, 6):
R R D PU U L U U U U R PL L D D D D D L L L PU U U U PL D R R

Robot 2 (17, 6):
R R D PU U U L L U U U R PL L D D D D D L L PU U U R U U L PL

796228 nodes generated
145842 nodes expanded
Algorithm completed in 1553.5797646045685 seconds
```

Figure 3: Result from medium world 1 with 2 robots

The fact that it was able to produce any path at all is a success compared to the test with sinlg-action nodes, which was not able to produce a result in the time I allowed it to run (30 min). However, the almost 26 minutes it took to find a path for the 2 robots with only 4 boxes is way too much, and the algorithm runtime would need to be significantly reduced to be admissible for a real life robot. The path seems to be optimal, though the first robot continues to move despite there being no boxes left to pick up.

Due to the long runtime on the goal world, the small world from Figure 1 was used to test the algorithm. The first test was on the runtime of the algorithm with 3 boxes and varying number of robots:

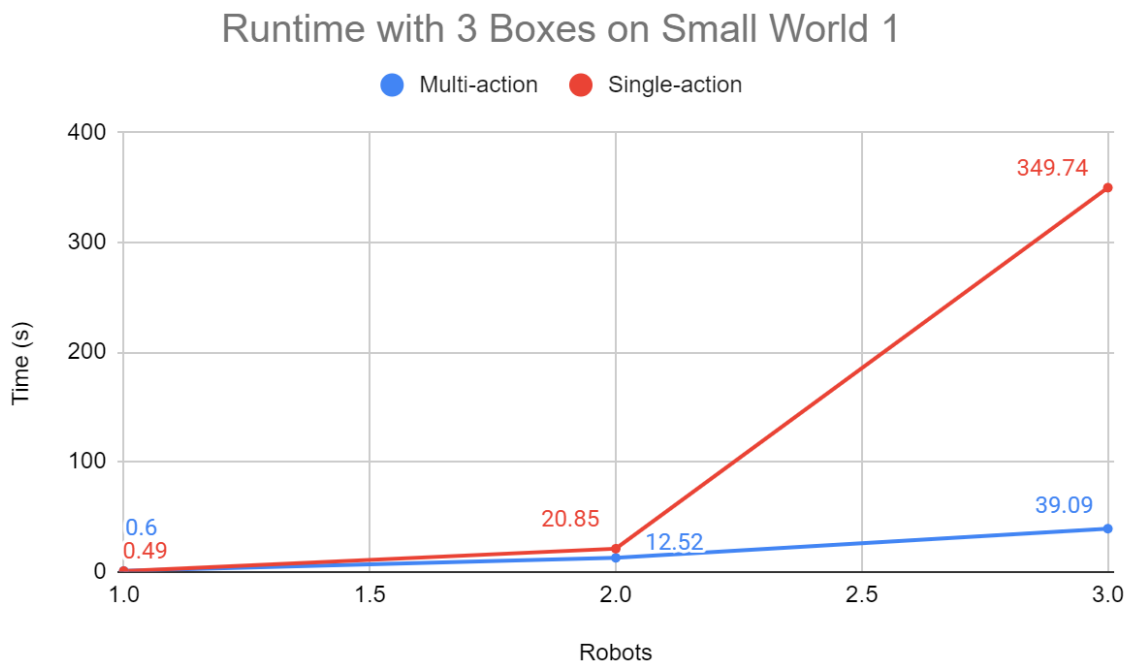

Figure 4: Robots vs. Time for 3 Boxes on small1

This graph shows the major difference between the single-action and multi-action nodes. The increase in time when adding more robots was 20x from 1 to 2, and only 3x from 2 to 3. The next test was on the runtime of the algorithm with 2 robots, and a varying number of boxes:

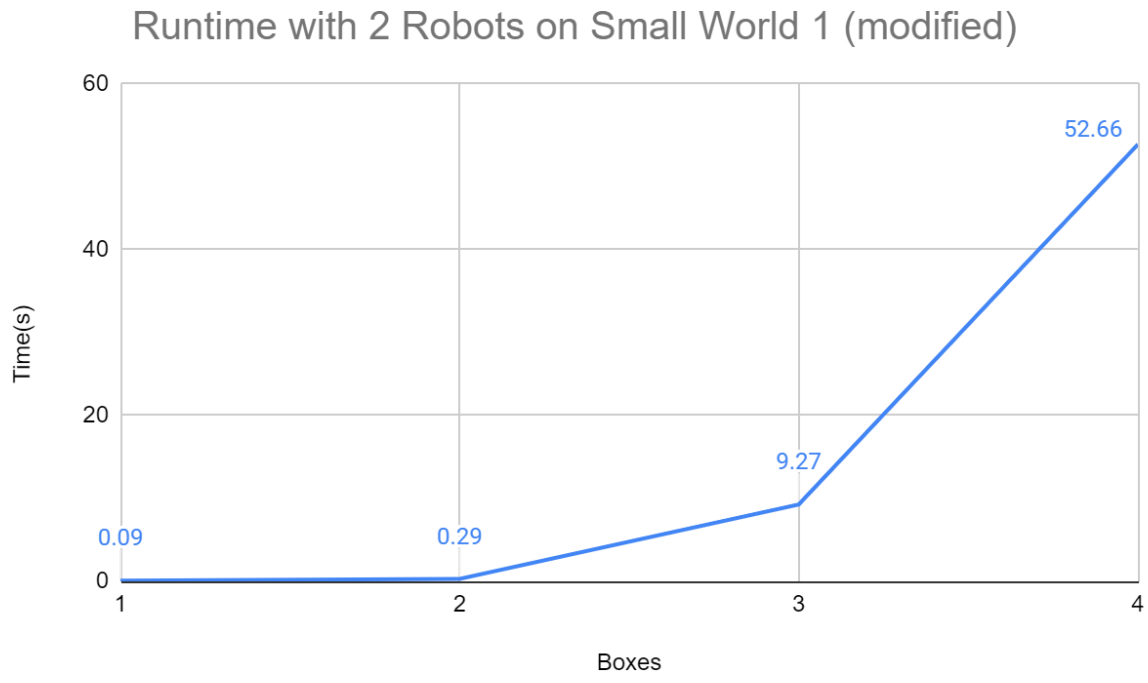

Figure 5: Boxes vs Time for 2 robots on a modified small1

Here we can see that adding more boxes exponentially increased the time taken to run the algorithm. The increases in order were 3x, 32x, 6x. Part of the large increase between 2 and 3

boxes comes from the layout of the world. Due to the need to use 4 boxes, world 1 was modified to have a 4th empty shelf closer to the initial box locations. This meant that the runtime for 2 boxes was shorter than it would have been on an unmodified small world 1, leading to a huge jump in time. The data I was able to obtain was somewhat limited by the quick jump in runtime and necessity of using small worlds, but I can safely say that the number of boxes has a much greater impact on the runtime than the number of robots. Not only do the graphs support this but showing a trend of exponential growth with boxes and logarithmic growth with robots, but this is also logically supported by how the program works. Each new box essentially adds two goals (a box to pick up and a shelf to drop off at), while each new robot reduces the depth that the goal state is found at.

II. Other results

This section covers other results which were either found to be less important or less successful.

A. Heuristics

Heuristics are very influential on the success of A*, but unfortunately I was not able to find any better heuristics than what I had initially used, which was the Manhattan distance. I tried an extension of the Manhattan distance which took into account the number of remaining boxes, but it increased the time the program took to run and was not able to find a result for small world 1 with 3 robots in a reasonable amount of time.

*B. Weighted A**

Adding a weight to A* is good for improving the runtime of an algorithm, but it usually results in suboptimal outputs. When using a weight > 1, the runtime was improved but the paths of the robots quickly became convoluted. They would move back and forth or around in a circle for a few actions before reaching their goal, so weighted A* was thrown out as an option.

CONCLUSION

The results of this project were not entirely conclusive on the feasibility of using A* in warehouse robotics, but they do prove that my implementation is not feasible in its current state due to the algorithm runtime. Any amount of world complexity quickly makes the algorithm take far too long to run, and in the 30 minutes it takes the robots to plan their paths a human could have reality moved the boxes themselves. However, the results were a partial success due to the ability for the algorithm to find an optimal path that always prevents the robots from colliding. If the runtime can be improved or the efficiency of the path can be accepted as less than optimal, this algorithm could potentially work.

The next step I would take given more time is an attempt at simplifying the goals. Adding more boxes significantly increases the runtime, so finding a way to reduce the effect they have on time complexity should be a priority. The next obvious step is attempting to improve the heuristics used. Heuristics are the most important part of any A* implementation, and finding a heuristic that is able to better predict the remaining cost. This could involve taking the number of boxes left into account in a much simpler way than I had previously attempted with my modified Manhattan heuristic. Lastly there are a few small things that could be changed, but ultimately have less effect on the algorithm. One is the fact that a robot will continue to try to move towards a box when there are no more boxes left to pick up. This results in robots waiting for the last boxes to be dropped off to

move aimlessly, so giving them a simple goal like moving back to their starting position could be a good way to take some time off the end of the algorithm.