A Comparative Study of Prioritized Planning and Conflict-based Search for Planning in Known, Fully Observable Multi-Agent Enviornments

Muhammad Saud Ul Hassan

August 25, 2020

Contents

Abstract

In the multi-agent path finding problem (MAPF) the objective is to plan a path for each agent in the environment while avoiding agent-obstacle and agent-agent collisions. Various algorithms have been proposed in literature to solve the problem, some guaranteeing neither completeness nor optimality and others guaranteeing both at the expense of a combinatorial explosion in the planning complexity. In this project, two of these algorithms, Prioritized Planning [1], [2] and Conflict-Based Search [4], have been implemented in Python, compared on various toy MAPF instances, and demonstrated on the use case of small-scale automated warehousing.

1 Introduction

By its very nature, path planning in multi-agent environments is a significantly more complex problem than single-agent planning. Having to look out for collisions both in cell space and time for several agents makes MAPF a NP-Hard problem to solve optimally [3].

Described more formally, the problem is to plan a path on a N-vertex graph for n robots. In the most basic formulation of the problem, called the Pebble Motion Problem [3], only one robot is allowed to move in a time-step. However, agents can be capable of communicating and coordinating with each other, which might allow them to execute a chain of synchronous moves in a single time step. This problem formulation, called *Multi-robot Path Planning on Graphs with Parallel Moves and Rotations* (or MPP_{pr} for short) by Yu et. al. [3], is the subject of study in this project.

2 Optimal MAPF

2.1 Problem Formulation

Let $G = (V, E, W)$ be a graph with vertices V, edges $E = \{(v_i, v_j) : v_i, v_j \in V\}$, and edge-weights $W = \{w_{v_i,v_j} = 1 : (v_i, v_j) \in E\}$. Let $R = \{r_i : i = 1, 2, ..., n, n \leq |V|\}$ be the robots on G's vertices. A move consists of a set of robots $R' \subset R$ at vertices $V_t \subset V$ at timestep $t \in Z_{\geq 0}$ moving to vertices $V_{t+1} \subset V$, where, for any robot $r_i \in R'$, the vertex $v_j \in V_t$ that it is at timestep t and the vertex $v_k \in V_{t+1}$ that it moves to are such that $(v_j, v_k) \in E \quad \forall j \neq k$. A feasible path $p_i \in P_i : Z_{\geq 0} \rightarrow V$ for an agent $r_i \in R$ is a set of vertices such that $p_i(0) = x_I(r_i)$, $p_i(t_i) = x_G(r_i)$, and $(p_i(t), p_i(t+1)) \in E$ or $p_i(t) = p_i(t+1)$ for $0 \leq t < t_i$ and $p_i(t) = p_i(t+1)$ for $t \geq t_i$. Here $x_I, x_G : R \rightarrow V$ map each $r_i \in R$ to its initial and goal vertex $v_i \in V$, respectively, and $t_i \in Z_{\geq 0}$ is such that $p_i(t) = x_G(r_i)$ for any $t \geq t_i$. The word *path* has been loosely used in this document to mean *feasbile path.*

The problem description above has been adapted from Yu et. al. [3]

2.2 Solution and Optimality Objectives

The solution to a MAPF problem corresponds to a set of *non-colliding paths* that minimizes (or maximizes) a certain objective. A path $p_i \in P_i$ is said to be in collision with a path $p_j \in P_j$ if $\exists k \in Z_{\geq 0}$ such that $p_i(k) = p_j(k)$ (*vertex collision*) or $(p_i(k), p_i(k+1)) = (p_j(k+1), p_j(k))$ (*edge collision*).

Three of the most common optimality objectives in MAPF are:

1. **Minimum Total Distance Objective:** Find a non-colliding path $p_i \in P$ for each robot r_i such that the total distance travelled by the robots is minimized, where the total distance is calculated as

$$\sum_{i=1}^{|R|} len(p_i) \tag{1}$$

where

$$len(p_i) = sum\left(w_{p_i(k),p_i(k+1)}\ 1_{p_i(k)\neq p_i(k+1)}\right), \forall k \in Z_{\geq 0} \tag{2}$$

2. **Minimum Total Travel Time Objective:** Compute a set of non-colliding paths that minimizes $\sum_{i=1}^{|R|} t_i$
3. **Minimum Makespan Objective:** Compute a set of non-colliding paths that that minimizes $\max_{0\leq i\leq |R|} t_i$

In this project, the algorithms are compared only on the total distance objective, though it is easy to adopt the code to minimize any other objective through simple modifications. It is important, however, to mention that Yu et. al. [3] have shown that in general, optimizing for all three of the above objectives is impossible for a arbitrary MPP_{pr} instance.

2.3 Intractability of Optimal MAPF

Optimal MAPF has no polynomial time solution. The proof of this is based on the fact that an arbitrary MAPF instance can be reduced to a SAT instance [3], and SAT is known to be a NP-hard problem, and, therefore, optimal MAPF is also NP-hard. With that out of the way, the next section presents two different algorithms for multi-agent path finding: Prioritized Planning, and Conflict-based Search

3 Algorithms

One approach to solving MAPF problems is to construct a space-time configuration space that represents the degrees of freedom of all the agents in the world as well as their positions in time, and plan a path through the space-time configuration space instead of the real space. This approach, though reasonable in single-agent problems, can quickly become computationally intractable if there are too many agents in the environment, since the configuration space will then be very high dimensional. This can be thought of as planning on a graph whose vertices correspond to tuples of cell positions. In this way, there are as many vertices in the graph as there are ways to combine the cell positions in groups of $|R|$, where $|R|$ is the number of agents. Search through such a graph would be computationally prohibitive.

3.1 Prioritized Planning

The alternative to Centralized Planning suggested by Erdmann et. al. [1] is *Prioritized Planning*. In Prioritized Planning, each agent is assigned a priority and planning is done one agent at a time, which is significantly easier to do than to plan for all agents concurrently. It is up to the agents lower in priority to make sure that their paths do not collide with the already planned paths, i-e, the paths of the agents that are higher in priority. This decomposes the problem into several single-agent planning problems where the already-planned paths appear to the agents yet to be planned for as obstacles moving in time.

In my implementation of Prioritized Planning, I have used space-time A* to solve the single-agent planning problem. In terms of the above problem formulation, this can be stated as finding a path p_i for an agent r_i such that

$$p_i = argmin_{p \in P_i'} \, len(p) \tag{3}$$

where P_i' is the set of all paths for robot r_i that do not collide with any robot r_j that has already been planned for.

The advantage of Prioritized Planning is that it is very fast. However, that comes at the expense of the completeness and optimality. There is no guarantee that Prioritized Planning would find a solution even if one exists, and if it does find one, there is no guarantee that it is the optimal solution.

3.2 Conflict-based Search (CBS)

CBS [4] provides the guarantees that Prioritized Planning so dearly lacks; it is complete, and it is optimal. However, to be able to provide these guarantees, CBS has to search through a much larger part of the solution space than does Prioritized Planning, and that means it might be far slower. However, to its credit, it manages to find the optimal solution without brute-forcing through the solution space. Rather, it searches through the solution space in a systematic and intelligent manner, leading it to find the solution (given that it exists) considerably faster than exhaustive methods.

CBS has two levels of search. In *low-level* search, CBS does single-agent planning in space-time space without caring for agent-agent collisions. Once all the agents have been planned for, it checks if any pair of paths is in collision. If a collision is detected, CBS tries to resolve it by imposing constraints on the colliding agents. For example, suppose agents r_i and r_j are both at vertex v_k at timestep t, resulting in what is called a *vertex collision*. CBS would try to resolve this collision by creating two new problem instances, one where r_i is constrained from being at vertex v_k at timestep t, and another where r_j is constrained from being at v_k at t. Then, it would plan for the individual agents in each of the new problem instances (*low-level* search) and get a new set of paths where the agents r_i and r_j would no longer be in collision at vertex v_k at time t. It would then do a *high-level* search (which is Best-first Search in my implementation) and greedily choose from the solutions the one with the minimum total path cost. If that solution has no colliding paths, it is returned. However, if there is still a collision, CBS would generate yet another pair of constrained problem instances, and repeat the whole cycle of low-level single-agent planning and high-level search through the binary constraint-tree. The search would continue until either a solution has found or it has been established that no solution exists.

4 Discussion

This section is devoted solely to some of the points mentioned in the project instructions on Canvas. The section after this will briefly go over the experiments I carried out on various MAPF instances and the results I obtained.

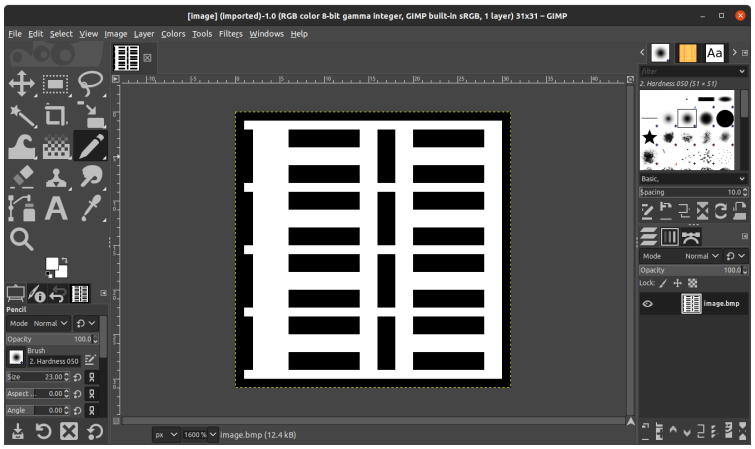

Figure 1: The warehouse map used in Fig. 10 being created in GNU-IMP

4.1 Language, software and API used?

The project was done entirely in Python. No AI-specific libraries were used, and I implemented the algorithms myself. As for the other software, I used GNU IMP (GNU Image Manipulation Program) to draw some of the grid-worlds I used for testing the solvers. Fig. 1 shows a sample grid-world used to test the MAPF solvers on the Automated Fulfilment Center Logistics problem. The grid-world images are converted into a .txt file (by a script I wrote in Python) to which the robots' initial and goal locations are added before sending to a MAPF solver.

I also used Matplotlib (a Python package that provides a MATLAB-like API for drawing plots) for the animations, but the code that does the animation was not written by me. (There are also other parts of the code here and there that I did not write myself)

4.2 What could be done better given more time?

There are some modifications to the algorithms that I would have liked to try if I had more time. For example, I could try improving Prioritized Planning by coupling it with Random Restarts. This could make the algorithm complete and optimal; or so I believe, but I have not studied it thoroughly enough to be sure. But the algorithm would not be particularly fast, I believe. In worst-case, one might have to do Prioritized Planning for each of the $n!$ different possible priority orderings of the robots.

I would also have liked to investigate Prioritized Planning with Intelligent Restarts, where the algorithm intelligently picks among the priority orderings for restarts instead of going about it randomly.

There are also ways to improve Conflict-based Search that I might have explored had there been more time. I was especially interested in trying Disjoint Splitting in CBS, where instead of imposing negative constraints that prohibit a particular agent from being in a particular cell at a particular timestep, as we do in vanilla CBS , positive constraints are imposed that make the agent *be* in

a particular cell at a particular timestep. There is also another variant of CBS, called CBS with Highways [5], that would have been yet another interesting algorithm to explore.

4.3 What did I learn?

In the class, cooperative multi-agent planning was only mentioned in the passing, and I had to study quite a bit of literature on the problem on my own in order to undertake this project. But I got to learn a great deal about path planning and also got to implement the algorithms by hand.

This project also gave me the chance to see first-hand how the fundamental algorithms we studied in class – like A*, Best-first Search, etc. – can be made small modifications to and combined to make something very powerful.

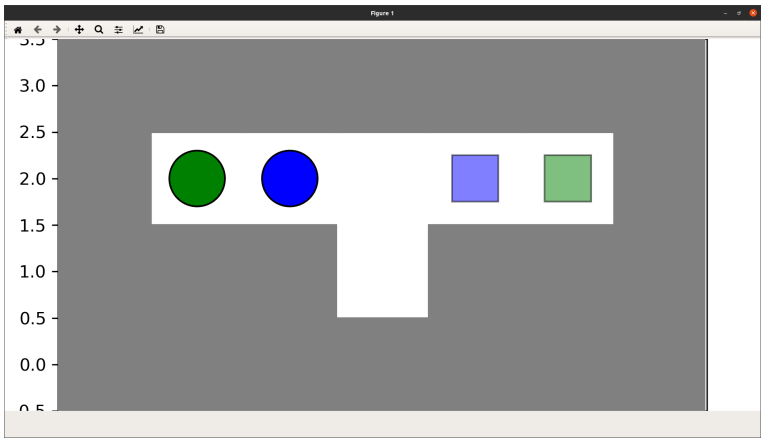

Figure 2: Initial Configuration. The circles represent the robot and the squares of matching colors represent the robots' goals

5 Experimental Results and Conclusions

The toy problem in Fig. 2 is a relatively simple problem, with only two agents, but even in this problem many algorithms fail. For example, simple A* search fails on this problem, as shown in Fig. 3, as it simply takes each agent in turn and plans a minimum-cost path to its goal without ever worrying about agent-agent collisions.

Prioritized planning, on the other hand, succeeds if the green agent has a higher priority (Fig. 4). The blue agent does not go directly to its goal in this case, because if it does, it would block the green agent's path and the green agent has the higher priority. So, the blue agent gives way to the green agent (Fig. 5) and advances to its goal only when it is sure that it would not be blocking the green agent's path.

If, on the other hand, the blue agent is given a higher priority then it goes straight to its goal and blocks the path to the green agent's goal. This has been demonstrated in Fig 6, where the goals have been switched instead of switching

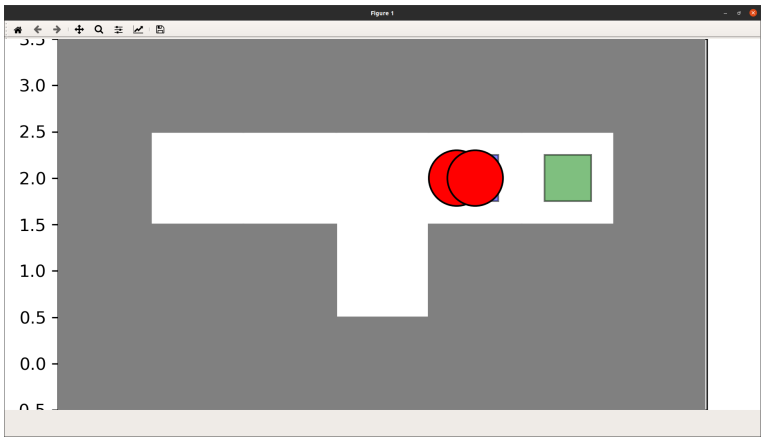

Figure 3: A* search plans a path that leads to the agents colliding with each other

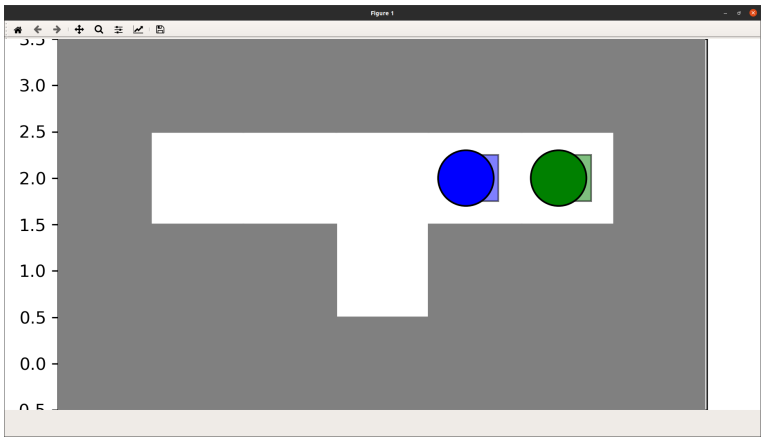

Figure 4: Prioritized Planning solves the problem if the green agent is planned for first

the agent priorities, but the end result is the same – the other agent has no way of getting to its goal.

This clearly demonstrates how Prioritized search is incomplete. Even though there exists a solution to the problem, prioritized search is unable to find it because of its sensitivity to agent priorities. CBS, on the other hand, is guaranteed to find the solution. Since there is only one solution, CBS gives the same paths as Prioritized Planning gave in Fig. 5.

There are cases where both Prioritized Planning and CBS are able to solve the problem, but the solution obtained through Prioritized Planning is not optimal. I compared the two algorithms on 50 different toy MAPF problems, in 48 of which Prioritized Planning was able to find a solution and in only 18 of those 48 did Prioritized Planning give the total-minimum-cost solution. The average total-minimum-cost in the problems was about 36.35 steps and the average total cost of the paths returned by Prioritized Search was around 38.58 steps, which

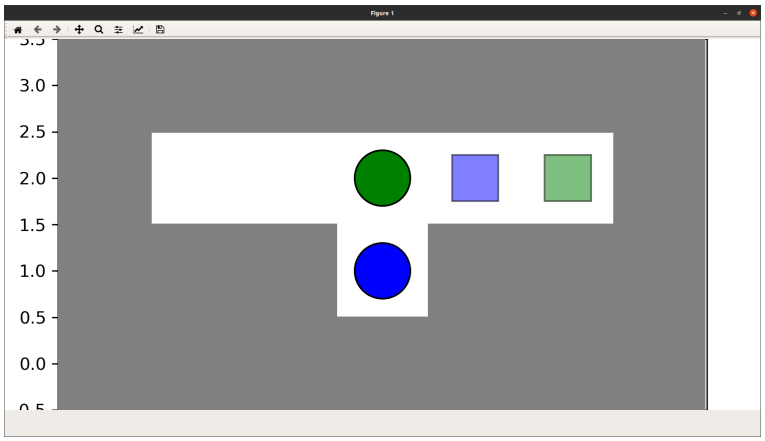

Figure 5: Since the green agent has a higher priority, the blue agent gives way to it

amounts to about 6% more steps on average. A sample problem instance from the set has been shown in Fig. 7.

Some other problems that the algorithms were tried on include path finding for multiple players in a maze (Fig. 8), where the narrow pathways made the chances of collision much higher. Fig. 9 shows another map, taken from the video game *Dragon Age: Origins*, which the agents navigated autonomously. Finally, the algorithms were applied to the problem of path planning in a warehouse (Fig. 10), which not included narrow corridors, but also a number of obstacles and a total of 31 agents, each with its own goal. Prioritized Planning returned a solution in 20 milliseconds, and for CBS I waited about an hour before I finally killed the search.

The results on these toy problems empirically show how Prioritized Search is fast but lacks completeness and optimality guarantees, and how CBS provides those guarantees but can be prohibitively slow in many cases.

References

[1] M. Erdmann and T. Lozano-Perez. *On multiple moving objects*. in 1986 IEEE International Conference on Robotics and Automation Proceedings, Apr. 1986, vol. 3, pp. 1419–1424, doi: 10.1109/ROBOT.1986.1087401.

[2] M. Čáp, P. Novák, A. Kleiner, and M. Selecký *Prioritized Planning Algorithms for Trajectory Coordination of Multiple Mobile Robots*. IEEE Transactions on Automation Science and Engineering, vol. 12, no. 3, pp. 835–849, Jul. 2015, doi: 10.1109/TASE.2015.2445780.

[3] J. Yu and S. M. LaValle, *Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs*. presented at the Twenty-Seventh AAAI Conference on Artificial Intelligence, Jun. 2013, Accessed: Jul. 27, 2020. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/view/6111.

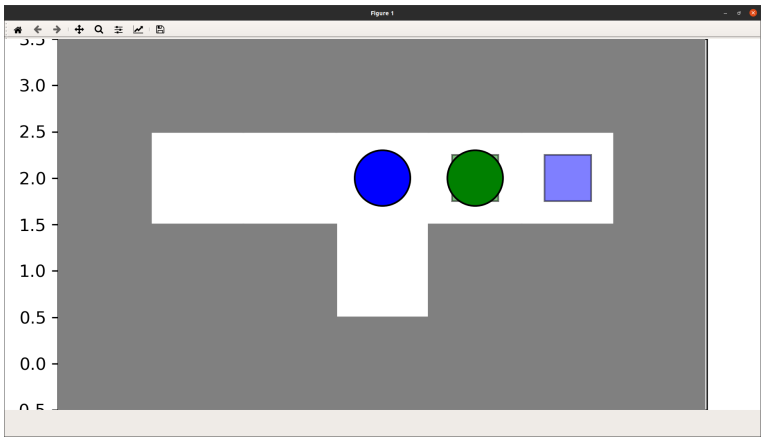

Figure 6: The green agent has a higher priority as in Fig. 4, however, the goals have been switched. The green agent goes straight to its goal and the blue agent is stuck with its path being blocked by the higher-priority green agent

[4] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, *Conflict-based search for optimal multi-agent pathfinding.* Artificial Intelligence, vol. 219, pp. 40–66, Feb. 2015, doi: 10.1016/j.artint.2014.11.006.

[5] L. Cohen, T. Uras, and S. Koenig, *Feasibility Study: Using Highways for Bounded-Suboptimal Multi-Agent Path Finding.*

Table 1: Prioritized Planning and CBS Compared on 48 different problem instances. This shows that the solution proposed by Prioritized Planning is about 2.3 steps longer on average compared to the solution that CBS proposes, where the average minimum path length is 36.35

Instance	**CBS (steps)**	**Prioritized Search (steps)**	**Difference (steps)**
instances/test_1.txt	41	49	8
instances/test_10.txt	19	21	2
instances/test_11.txt	35	37	2
instances/test_12.txt	36	40	4
instances/test_13.txt	36	39	3
instances/test_14.txt	24	24	0
instances/test_15.txt	50	52	2
instances/test_16.txt	51	51	0
instances/test_17.txt	39	39	0
instances/test_18.txt	32	32	0
instances/test_19.txt	47	49	2
instances/test_2.txt	18	18	0
instances/test_20.txt	28	33	5
instances/test_21.txt	46	46	0
instances/test_22.txt	51	52	1
instances/test_23.txt	32	32	0
instances/test_24.txt	47	48	1
instances/test_26.txt	42	42	0
instances/test_27.txt	40	47	7
instances/test_28.txt	41	41	0
instances/test_29.txt	48	54	6
instances/test_3.txt	28	28	0
instances/test_30.txt	43	50	7
instances/test_31.txt	39	39	0
instances/test_32.txt	30	30	0
instances/test_33.txt	28	31	3
instances/test_34.txt	33	34	1
instances/test_35.txt	30	32	2
instances/test_36.txt	23	23	0
instances/test_37.txt	38	38	0
instances/test_38.txt	28	28	0
instances/test_39.txt	35	39	4
instances/test_4.txt	32	33	1
instances/test_40.txt	24	28	4
instances/test_41.txt	45	45	0
instances/test_42.txt	57	63	6
instances/test_43.txt	43	48	5
instances/test_44.txt	33	36	3
instances/test_45.txt	24	24	0
instances/test_46.txt	57	59	2
instances/test_48.txt	36	38	2
instances/test_49.txt	42	46	4
instances/test_5.txt	26	27	1
instances/test_50.txt	48	58	10
instances/test_6.txt	24	26	2
instances/test_7.txt	34	35	1
instances/test_8.txt	38	44	6
instances/test_9.txt	24	24	0
Average Path Cost	36.35	38.58	

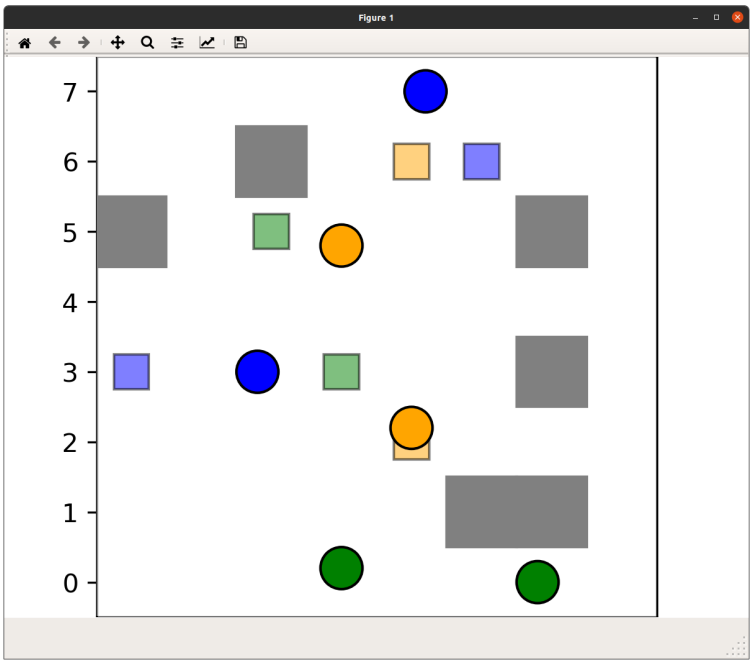

Figure 7: Prioritized Planning and CBS were compared on 48 of such problem instances. The circles here ar the robots and the squares are the goals. The grey areas represent obstacles

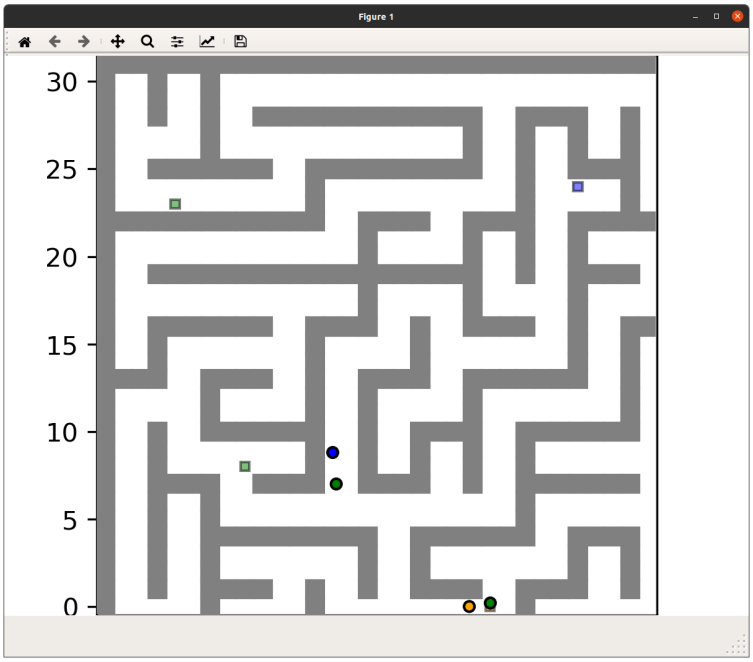

Figure 8: Robots navigating a maze. Even with just four robots, it can be seen how likely the collisions are. Also, observe the yellow and green robots at the bottom. The green robot is at yellow's goal, but if green gets aside for yellow to get to its goal, then yellow would block green's path and green would never be able to get to *its* goal. So, the yellow robot has to back up, let the green robot go, and then advance to its goal. This example here was solved using Prioritized Planning. CBS would have detected such a situation in advance and would have planned the paths so as ascertain that the yellow robot does not have to backup and cover more distance than it has to in doing so.

Figure 9: Agents navigating a map taken from the popular video game *Dragon Age: Origins*

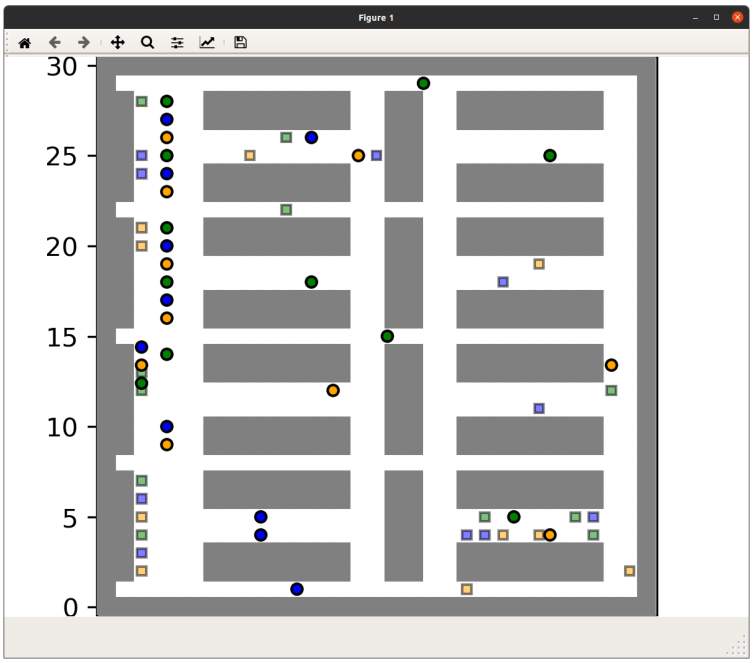

Figure 10: Agents navigating a warehouse. On the left of the warehouse are drop-off stations where a robot would arrive with a basket and a human would take off the items they need from the basket attached to the robot. The robot would then drive back to the appropriate storage shelf and place the basket back. Amazon has employed automated warehousing robots like these to great success in its order-fulfillment centers.