Multi-Agent Pathfinding: Comparison of Prioritized Planning and Conflict-Based Search Algorithms

COMP 3190: AI Research Project

Colin McDonell

ID: 7940861

December 7, 2022

1 Introduction to Multi-Agent Pathfinding

Multi-Agent Pathfinding (MAPF) is a multi-agent planning problem where the task is to calculate paths from start to goal vertices for multiple agents on an undirected graph. The agents must be able to follow their paths concurrently without colliding with each other. [Stern et al. (2019)]

The solution to MAPF problems is the plan with lowest total cost for all agents to get to their goal locations. Finding an optimal solution to any MAPF problem is NP-hard [Yu & LaValle (2013)] because the state-space grows exponentially with the number of agents. Therefore, solving MAPF problems optimally is only feasible for a small number of agents. However, modifications of optimal solving approaches can be very effective for solving problems with many agents and little sacrifice in optimality. [Barer et al. (2014)]

There are many variations of MAPF problems, with different assumptions and requiring different approaches to solving the problem. However, the paper's main focus will be on classical MAPF problems as defined by Stern. [Stern et al. (2019)] In classical MAPF problems, n agents must get from their start to their goal locations on an undirected graph concurrently without colliding with each other. In classical MAPF, three key assumptions pertain to all problems. The first assumption is that time is discrete. An agent's movement and location are not defined for fractional timesteps. Another assumption is that every action is completed in exactly one timestep. Any movement of an agent to a neighbouring vertex takes exactly one timestep. The final assumption is that an agent can only occupy a single vertex per timestep. [Stern et al. (2019)]

1.1 Conflicts in MAPF

A valid MAPF solution to a classical problem requires that all agents can get from their start to goal vertices without any collisions taking place. In order to define a valid solution, we need to define what is considered a collision.

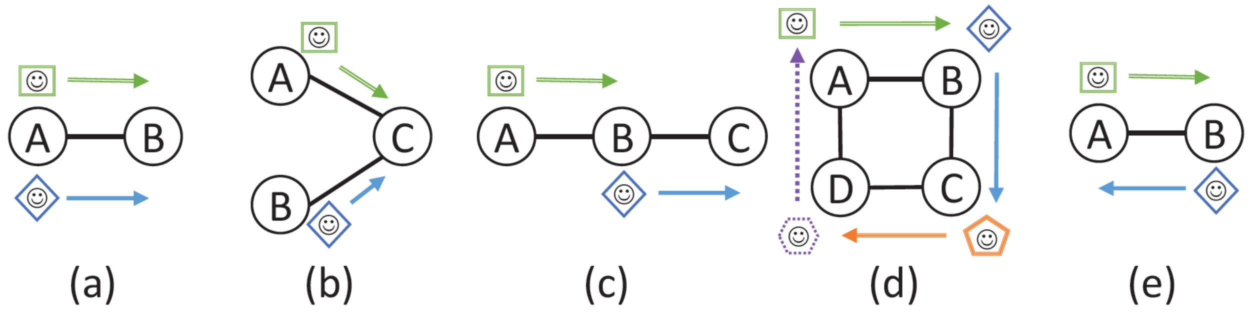


Figure 1: Types of conflicts in MAPF. (a) Edge conflict. (b) Vertex conflict. (c) Following conflict. (d) Cycle conflict. (e) Swapping conflict. [Stern et al. (2019)]

There are five types of conflicts used when solving MAPF problems. [Stern et al. (2019)] Figure 1 shows types of conflict in MAPF problems. An edge conflict (a) takes place when agents move from the same vertex to another identical vertex in the same time step. A vertex conflict (b) takes place when to agents move to the same vertex, both occupying it at the same timestep. A following conflict (c) happens when one agent moves to a vertex that another agent has left on the previous timestep. A cycle conflict (d) is an extension of a following conflict which involves four agents following each other in a circle. Finally, a swapping conflict (e) is when two agents switch vertices between two consecutive timesteps. [Stern (2019)]

According to Stern et. al. (2019), all prior research on classical MAPF considers edge conflicts ((a) and (e)) and vertex conflicts (b) as collisions between agents. Research on search-based MAPF algorithms typically allows following conflicts and cycle conflicts. [Stern et al. (2019)]

1.2 Objective Functions in MAPF

Two objective functions commonly used in classical MAPF problems are makespan and sum-of-costs. [Stern et al. (2019)]

The makespan objective function is the timesteps required for all agents to reach their goal destination. [Stern et al. (2019)]

The sum-of-costs (also known as flowtime) objective function is the total actions required by all agents to reach their targets. An agent action is either moving from one vertex to a neighbouring vertex or staying still for one timestep. Prior research commonly assumes that an agent waiting at its goal location while other agents have not yet reached their destination is not considered in the cost function unless the agent has to move from its goal location at a later timestep. (for example, to avoid a collision between agents) [Stern et al. (2019)]

1.3 Variations of MAPF

Classical MAPF problems are studied extensively. However, the assumptions in classical MAPF problems make MAPF algorithms less applicable to real-world scenarios. Modifications are made to these assumptions to apply MAPF problems to real-world scenarios. The following sections will briefly discuss variations on classical MAPF. Different algorithms are proposed and tested for these variations of MAPF but are out of the scope of this paper.

1.3.1 MAPF on Weighted Graphs

Classical MAPF takes place on 4-connected grids. MAPF on weighted graphs takes place on 2^k-connected grids or in Euclidean space. [Stern et al. (2019)] It is no longer assumed that an agent moves to an adjacent vertex or waits at its current vertex each timestep. Instead, movements to different adjacent vertices can take different amounts of time or incur different costs. (To traverse a longer distance either takes more time or requires more energy to get there in the same amount of time as going a shorter distance). [Stern et al. (2019)]

In 2^k-connected, the allowed move actions for an agent at a cell are all its 2^k neighbouring cells in the grid. (Figure 2) [Stern et al. (2019)]. For MAPF problems in Euclidean space, every vertex in the MAPF graph is a Euclidean point (x,y) and allowed move actions are represented by the edges. [Stern et al. (2019)]

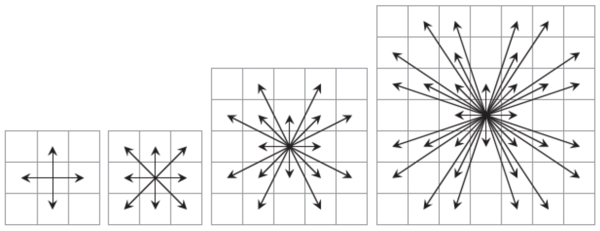


Figure 2: "2^k Neighbourhood movement models for k = 2,3,4 and 5"[Stern (2019)]

1.3.2 MAPF with Additional Solution Feasability Rules

When applying MAPF to real-world scenarios, there is some uncertainty in agents movements. Additional solution requirements can be added to make the solution more likely to succeed. MAPF solutions are robust if the plan for all agents is still conflict free when an agents get delayed up to k timesteps. [Atzmon et al. (2018)] Probabilistic robustness is another form of a robust solution. The probability of agents being able to follow their planned routes is within a probability bound. [Atzmon et al. (2018)]

Another example of an additional MAPF solution requirements is formation rules for the agents. These requirements restrict agents' movements depending on the location of other agents. [Stern et al. (2019)] An example of this requirement is that every agent must have a line of site to at least one other agent at all times to maintain communication between agents. [Stump et al. (2011)]

1.3.3 MAPF with Large Agents

The classical MAPF assumption that agents can only occupy one vertex is not always true when agents have physical dimensions. Examples of such scenarios are quadrotors [Hönig et al. (2018)] and convoys. [Thomas et al. (2015)]

In MAPF with large agents, agents can occupy more than one vertex and can collide with other agents at neighbouring vertices. Agents in the same problem can have different shapes and sizes, and have different movement possibilities. [Li, Surynek, Felner, Ma, Kumar & Koenig (2019)]

With large agents, the classical MAPF constraints are generalized to prevent conflicts between agents. Vertex conflicts are generalized so that two agents may not occupy two vertices at the same time if the agents are close enough to collide with each other. Edge conflicts prevent two agents from moving along an edge at a given time if doing so will result in a collision with either an agent waiting at a nearby vertex or moving along a nearby edge. Figure 3 shows large agents colliding even though they are at different vertices. [Hönig et al. (2018)]

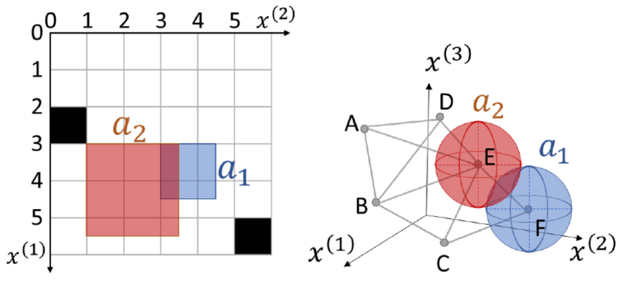


Figure 3: Examples of large agents colliding with each other while at different vertices. [Li, Surynek, Felner, Ma, Kumar & Koenig (2019)]

1.3.4 MAPF with Kinematic Constraints

MAPF with kinematic constraints incorporates real-world kinematic constraints when solving MAPF problems. [Stern et al. (2019)] An agent's movement possibilities depend on their velocity and orientation as well as their location. Some examples of kinematic constraints are different agents can have different maximum velocities, they must rotate before they can go in a different direction [Barták et al. (2018)], or they have non-uniform velocities. [Hönig et al. (2016)] Kinematic constraints can constrain agents' movements in the graph so that some edges can only be used in a specific direction. This changes the undirected graph of the MAPF problem to a directed graph. [Stern et al. (2019)]

1.3.5 MAPF with Deadlines

MAPF with deadlines is a version of MAPF where the maximum agents have to reach their goal vertices within a deadline. [Ma et al. (2018)] Some real-world examples of this variation are robots evacuating from a disaster zone or needing to finish as many tasks as possible before a deadline. [Ma (2022)]

1.3.6 Anonymous MAPF

In Anonymous MAPF, agents are not assigned specific start and goal vertices. Instead, the aim is to find a one-to-one mapping of agents to a set of goal vertices that results in the lowest objective function. [Ma (2022)] Consider a warehouse where packages need to be picked up by a team of robots. It does not matter which robot is assigned to pick up each package as long as all packages are picked up by the team in the most efficient manner possible. Coupling the agent assignment problem with a MAPF problem can achieve greater optimality. [Ma, Koenig, Ayanian, Cohen, Hönig, Kumar, Uras, Xu, Tovey & Sharon (2017)]

Target assignment and pathfinding (TAPF) is a close variant of Anonymous MAPF where agents are assigned to teams, and each team is responsible for assigning their agents to the team's set of goal locations. [Ma, Koenig, Ayanian, Cohen, Hönig, Kumar, Uras, Xu, Tovey & Sharon (2017)]

1.3.7 Online MAPF

In online MAPF, an agent is assigned a new goal location once it reaches its current goal. [Ma (2022)] There are different approaches to incorporating an agent's new goal into the overall MAPF plans. Hang (2021) demonstrated it is better to allow the replanning of other agents when an agent receives a new goal location. [Ma (2021)]

Multi-agent pickup and delivery (MAPD) combines anonymous MAPF and online MAPF. A group of agents are assigned tasks in an online setting. The tasks can enter the system at any time so the planning and replanning of optimal paths must be done during execution time. [Ma, Li, Kumar & Koenig (2017)]

2 MAPF Algorithms

2.1 A* Pathfinding Algorithm

A* is a search algorithm used for pathfinding. [Leigh et al. (2007)] It uses a best-first search algorithm with an admissible heuristic function and maintains a priority queue of vertices called OPEN. (Stern, Multi-agent path finding–an overview, 2019) Starting with the source vertex, vertices with the highest priority are removed from OPEN and expanded. Expanding a vertex means going to its neighbouring vertices and adding them to OPEN. Nodes with the highest priority are removed and expanded until a path to the goal state is found or OPEN is empty. A* is proven to be optimal and complete. [Hart et al. (1968)]

Using A* for pathfinding for multiple agents has several drawbacks. The size of the state-space is exponential in the number of agents. Maintaining the list of expanded states in memory becomes an issue for problems with many agents. [Felner et al. (2017)] The second major drawback is the branching factor. Since an agent can move in either of the four cardinal directions or stay still during a time step, the branching factor is 5 for a single agent. (assuming a 4-connected grid) [Felner et al. (2017)] Generating neighbouring states from a start state of 20 agents on a 4-connected grid would require generating $5^{20} = 9.53\times10^{14}$ nodes which is computationally infeasible. [Felner et al. (2017)] To use A* search beyond a trivial amount of agents, the algorithm must be modified. Prioritized planning uses A* pathfinding but reduces the size of the search-space by relaxing the requirement for an optimal and complete solution. Conflict-based search uses a two-level approach with A* as its low-level pathfinding algorithm and a binary constraint tree as its high-level search-space. Both algorithms will be discussed in detail in later sections. Other effective extensions of A* have been developed to make the computation of optimal paths for all agents much more feasible. They are operator decomposition (OD), independence detection (ID), and the M* algorithm [Stern (2019)] but they are out of the scope of this paper.

2.2 Prioritized Planning

One of the ways to solve MAPF problems quickly is to reduce the task into finding single-agent paths that do not conflict with other agents. A popular approach is Prioritized Planning. In Prioritized Planning, agents are assigned an order 1,...,k. A path is then found for the first agent using a single-agent pathfinding algorithm such as A*. The path for the next agent in the order is planned so that it does not conflict with the first agent. [Stern (2019)] The process contiues until all agents have collision free paths or a solution can not be found. One difference when finding a path for agents in Prioritized Planning vs single-agent pathfinding is agents are allowed to stay still for one or more timesteps. This allowance must be implemented into the algorithm.

Prioritized planning is commonly use for MAPF problems for its simplicity and computational efficiency. [Stern (2019)] When an optimal and complete solution is required, other solving methods should be used.

Prioritized Planning is not optimal because once an agent's path has been planned, it is never modified. An agent whose path is planned later in the order might have to go a much longer route to avoid conflicts when a slightly longer path of the earlier agent would lead to a much shorter path for the later agent and lower objective function. [Stern (2019)]

Prioritized Planning does not always find a solution to a solvable MAPF problem. For example, if an earlier agent's path prevents a later agent from reaching its goal location, Prioritized Planning will not find a solution. Figure 4 shows a simple MAPF problem that a Prioritized Planning algorithm will not be able to solve.

2.3 Conflict-Based Search

Conflict-Based search (CBS) is an optimal and complete MAPF algorithm. [Li, Harabor, Stuckey, Felner, Ma & Koenig (2019)] It is a two-level algorithm. The low-level uses the A*

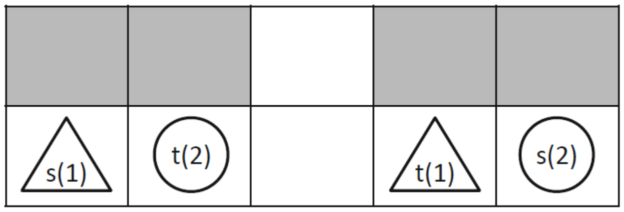


Figure 4: A Prioritized Planning algorithm will not be able to find a solution to the above problem [Stern (2019)]

algorithm [Silver (2005)] to find the best possible paths for each agent and the high-level employs a best-first search on a binary constraint tree (CT). [Li, Harabor, Stuckey, Felner, Ma & Koenig (2019)] The nodes in the CT contain constraints on agents' paths to prevent collisions and the paths of each agent that satisfy the constraints. The heuristic used for the best-first search on the constraint tree is the sum-of-costs of each agent's collision free path. [Li, Harabor, Stuckey, Felner, Ma & Koenig (2019)] A negative constraint prevents an agent from being at a specific location at a specific timestep. A positive constraint requires an agent to be at a specific place at a specific time. [Li, Harabor, Stuckey, Felner, Ma & Koenig (2019)]

When the high-level search expands a CT node, it check if there are any collisions between agents in their path plans. If there are no collisions, then a solution is found. Otherwise, one of the conflicts is resolved by splitting the node into two child nodes. In each child node, a negative constraint is added to prevent one of the conflicting agents from using the vertex that caused the collision in their path. The path of the conflicting agent is replanned to satisfy the constraint by the low-level pathfinding algorithm. [Li, Harabor, Stuckey, Felner, Ma & Koenig (2019)] Optimality is guaranteed with CBS because all ways of resolving each conflict are explored. [Li, Harabor, Stuckey, Felner, Ma & Koenig (2019)]

2.3.1 CBS Improvements

There are a few ways of improving standard CBS while maintaining its optimality. In standard CBS, the order in which conflicts are resolved is random by default. [Li, Harabor, Stuckey, Felner, Ma & Koenig (2019)] Improved CBS chooses to first resolve conflicts in a CT node that produce two child nodes with a larger sum-of-costs than the parent node. [Boyarski et al. (2015)] These types of conflicts are called cardinal conflicts. [Li, Harabor, Stuckey, Felner, Ma & Koenig (2019)] Fewer CT nodes are expanded to find the solution if cardinal conflicts are resolved first. [Li, Harabor, Stuckey, Felner, Ma & Koenig (2019)] Another way to use cardinal conflicts proposed by Felner et. al., is to include them in the heuristic function for the high-level search. This can significantly speed up the search and can improve CBS by up to a factor of 5. [Felner et al. (2018)]

2.3.2 CBS with Disjoint Splitting

In standard CBS, it is possible for some plans to appear in more than one CT nodes. The duplication of plans can lead to increased search effort. [Li, Harabor, Stuckey, Felner, Ma & Koenig (2019)] To eliminate this duplication of plans, CT nodes can be split disjointly. Instead of imposing a negative constraint for one of the conflicting agents in each child node, an agent is negatively constrained in one child node and positively constrained in the other child node. A positive constraint means that the agent must be at a specific location at a specific time. The positive constraint prohibits any other agent from being at the location during that time. By splitting a CT node this way, any plan can only belong to one child CT node, making them disjoint. [Li, Harabor, Stuckey, Felner, Ma & Koenig (2019)]

There are different ways to choose the agent on which the positive and negative constraint will be imposed. Some options explored by Li et. al (2019) is to choose the agent whose path has the most conflicts, most constraints, or least constraints. However, they found that these methods performed similarly to randomly choosing the agent. [Li, Harabor, Stuckey, Felner, Ma & Koenig (2019)]

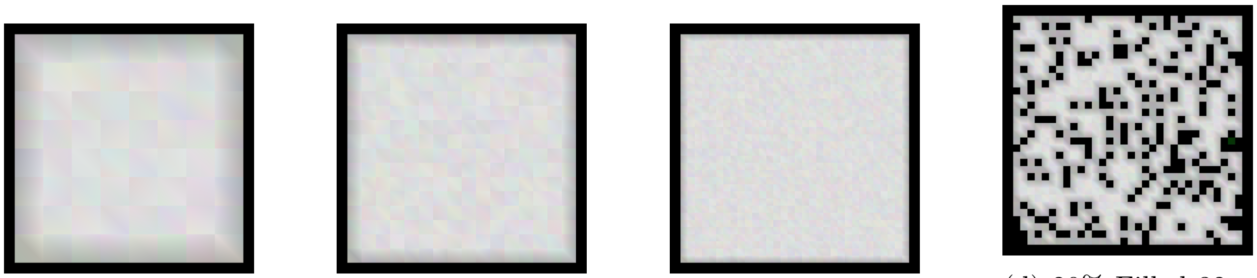
3 Evaluation of Prioritized Planning and CBS Algorithms

Prioritized planning and CBS (both non-disjoint and disjoint) were implemented in python by following a MAPF project outline found at mapf.info under Class Projects [Hönig et al. (2020)] Starter code was provided but the implementation of all MAPF solvers was original work. Algorithms were tested and compared on map instances provided with the code and further benchmarked using a selection of benchmarking scenarios found at movingai.com/benchmarks/mapf/index.html [Sturtevant (2012)]. These benchmarking maps are commonly used to compare MAPF solvers [Stern et al. (2019)] and are cited in over 500 articles according to google scholar. [*Google Scholar* (2022)] A zipped Folder containing all project code can be found at https://github.com/ColinMcD6/AIProjectSubmission.git.

3.1 Maps Selection

Empty maps with heights and widths of 8 by 8, 16 by 16, and 32 by 32 were used. Empty maps were used so that more agents could fit in the maps. The focus of the algorithm evaluation was to determine a given algorithms ability to resolve conflicts between paths of agents. An empty grid with lots of agents produces a lot of initial conflicts between agents' optimal paths.

The final map used was a 32 by 32 grid with 20% of its vertices as wall cells. This map was included with the others to compare the effects of adding wall cells to the algorithms' solving times. Each map's configuration file contained the start and goal locations of a list of agents. Each map type has 50 different agent configurations, 25 where the agents' start and goal locations are spread out evenly across the grid and 25 where they are random.



(a) 8 x 8 map (b) 16 x 16 map (c) 32 x 32 map (d) 20% Filled 32 x 32 map

Figure 5: Maps used for benchmarking. [Sturtevant (2012)]

3.2 Benchmarking

For each map's configuration file, the MAPF algorithms were run starting with 2 agents. The number of agents was incremented and solved until the algorithms could not solve the MAPF instance in the maximum time allowed. The max time allowed ranged from 30 seconds to 2 minutes depending on the map. The 32 by 32 maps were given 2 minutes to solve as the number of agents that could be fit on the map was higher and required more solving time. Figure 5 shows the four different maps that were used for benchmarking the MAPF algorithms.

3.3 Results

The solve rate of the Prioritized Planning, CBS, and CBS with disjoint splitting algorithms are shown below for the four maps used for benchmarking. The algorithms were run on each map with evenly spaced agents and randomly placed agents. Table 1 shows a summary of the benchmarking data from the three algortihms.

3.4 Discussion

The results of running Prioritized Planning, CBS, and CBS with disjoint splitting algorithms on the benchmarking maps showed that incorporating disjoint splitting into CBS makes the algorithm perform much more efficiently. Table 1 shows that CBS without disjoint splitting

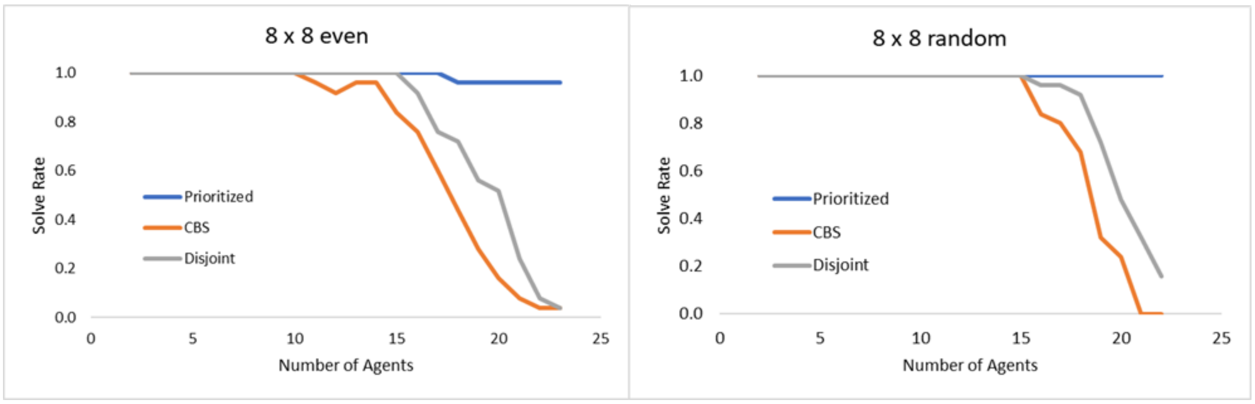


Figure 6: Solve rates for Prioritized Planning, CBS, and CBS with disjoint splitting on an 8 x 8 empty grid.

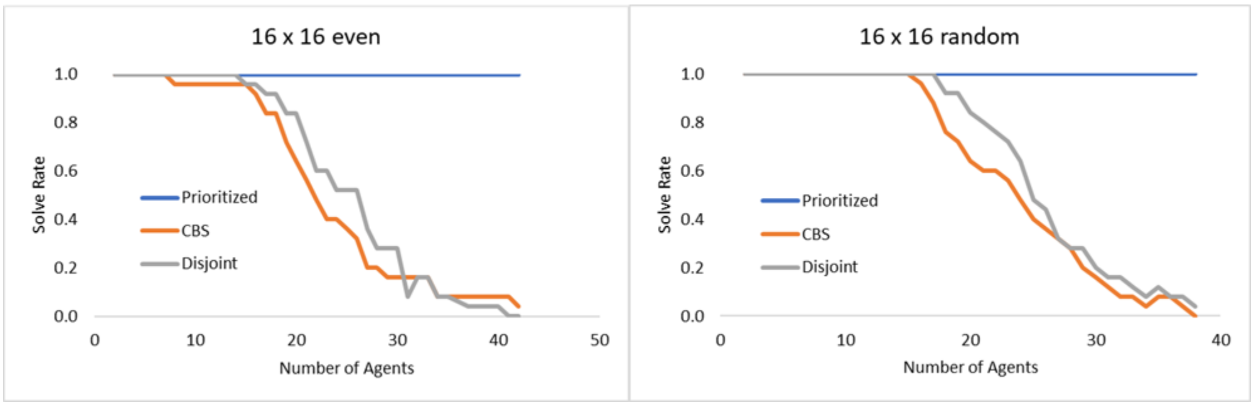


Figure 7: Solve rates for Prioritized Planning, CBS, and CBS with disjoint splitting on a 16 x 16 empty grid.

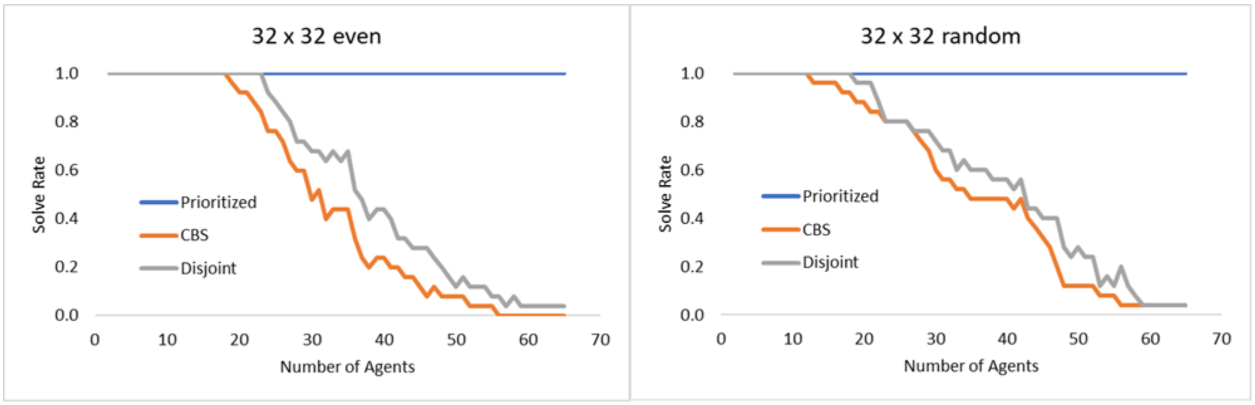


Figure 8: Solve rates for Prioritized Planning, CBS, and CBS with disjoint splitting on a 32 x 32 empty grid.

took on average 135% more time to solve the MAPF problems and generated on average 193% more CT nodes. In all benchmarking maps, CBS with disjoint splitting was able to

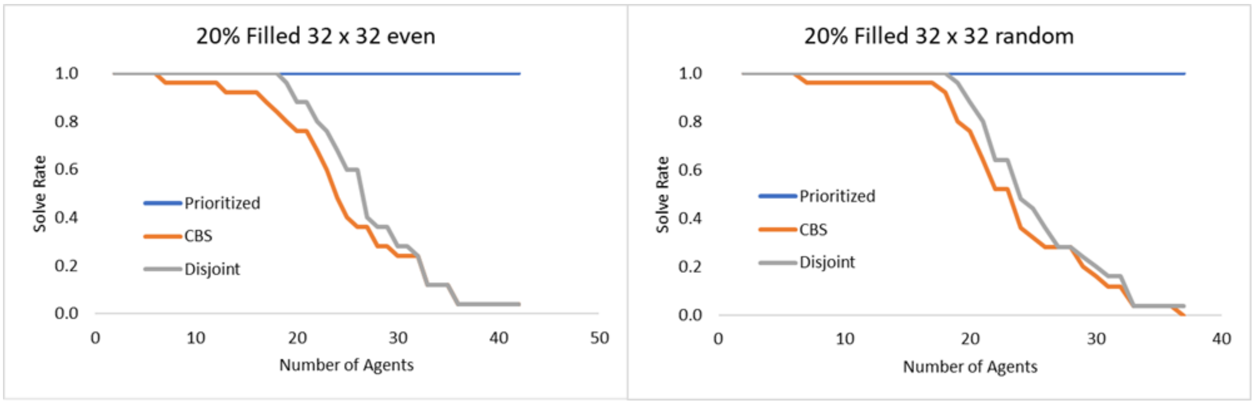


Figure 9: Solve rates for Prioritized Planning, CBS, and CBS with disjoint splitting on a 32 x 32 20% filled grid.

solve more instances in the allotted time. The results also highlight the huge time savings in allowing some suboptimality in the solution. The prioritized planning algorithm on average had 4.5% suboptimality in its final solution cost. However allowing for a suboptimal solution increased solving time by an average factor of 34 times. This huge solving time decrease allowed prioritized planning to solve much more of the instances in the allotted time than the two CBS algorithms did.

One thing to note in Table 1, is that the shorter solving time of Prioritized Planning is less significant over CBS variations on bigger maps. One of the possible reasons for this phenomenon is the single-agent pathfinding algorithm for prioritized planning and low-level search in CBS takes much longer on larger grids. The extra time that CBS takes to search the constraint tree becomes a less significant contributor to the total solving time. More work would need to be done to test this hypothesis.

In any case, it is important to consider the application of the MAPF algorithm before selecting the best one from the three analysed. If the solution does not need to be optimal or complete, then prioritized planning should be considered because of its rapid solving time. If an optimal solution is required at the expense of solving time, then CBS with disjoint splitting is the best of the three analysed.

Table 1: Benchmarking Results

Map	**Prioritized Planning Suboptimality**	**% Extra Nodes Generated by CBS vs Disjoint**	**% Time Increase by CBS vs Disjoint**	**% Time Increase by Disjoint vs Prioritized Planning**
Test Instances	4.8%	300%	177%	7459%
8 x 8 even	6.9%	273%	209%	9277%
8 x 8 random	6.5%	350%	348%	6310%
16 x 16 even	5.1%	113%	90%	1526%
16 x 16 random	3.6%	180%	86%	1322%
32 x 32 even	3.0%	39%	11%	507%
32 x 32 random	4.9%	197%	82%	150%
20% Filled 32 x 32 even	3.5%	130%	96%	662%
20% Filled 32 x 32 random	1.9%	155%	116%	3521%
Average	**4.5%**	**193%**	**135%**	**3415%**

3.5 Future Work

One area of future work is to further improve the CBS algorithm and quantify the performace change of the improvements. One simple improvement for disjoint splitting is for agents' paths to use positive constraints to segment their paths. When an agent needs to replan its path, only replan the affected segment of the path. [Li, Harabor, Stuckey, Felner, Ma & Koenig (2019)] Incorporating this imporovement could decrease the time spent on the low-level search in CBS.

Another improvement is using cardinal conflicts. Using cardinal conflicts to choose the order in which conflicts are resolved in disjoint CBS, and incorporate them into the heuristic function have been shown to improve the CBS algorithm. [Boyarski et al. (2015)] [Felner et al. (2018)]

Given that Prioritized Planning had signifantly faster solving times than the two CBS algorithms, it would be interesting to include another suboptimal algorithm in the benchmarking. Enhanced Conflict-Based Search (ECBS) is a suboptimal version of CBS and has been shown to solve MAPF problems with only a 1% maximum suboptimality for 250 agents on large maps. [Barer et al. (2014)]

Finally, the maps used for the benchmarking were very open. Future work could include more congested maps in the benchmarking. An example of a congested map is the warehouse map in Figure 10. It would be interesting to see how an incomplete solver like Prioritized Planning would perform on a congested map. With less open paths to goal locations, the chances of agents waiting at thier goal locations completely preventing other agents from reaching their targets would increase.

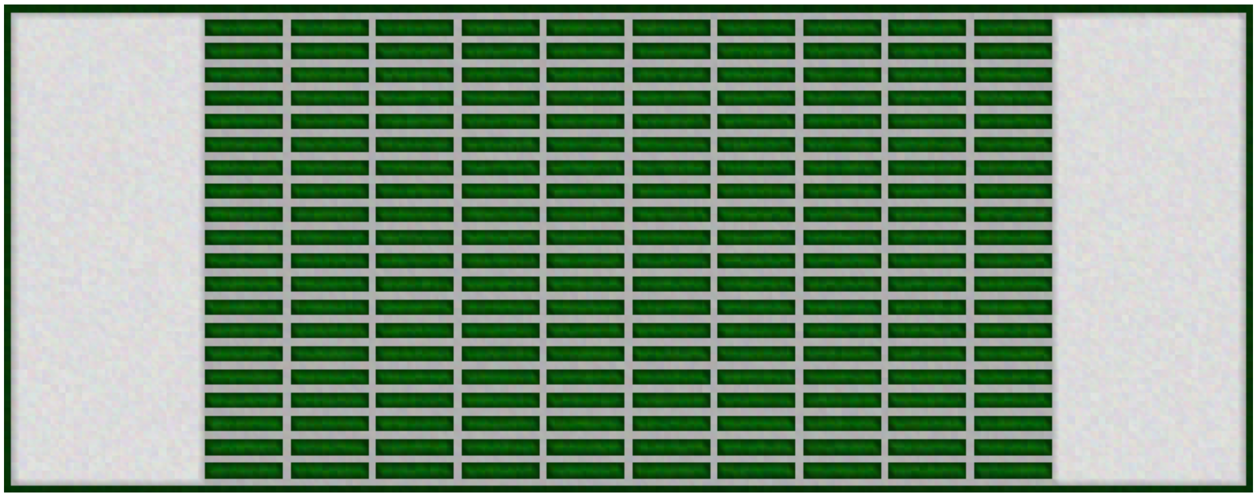


Figure 10: Congested warehouse benchmark map. [Sturtevant (2012)]

References

Atzmon, D., Stern, R., Felner, A., Wagner, G., Barták, R. & Zhou, N.-F. (2018), Robust multi-agent path finding, *in* 'Eleventh Annual Symposium on Combinatorial Search'.

Barer, M., Sharon, G., Stern, R. & Felner, A. (2014), Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem, *in* 'Seventh Annual Symposium on Combinatorial Search'.

Barták, R., Švancara, J., Škopková, V. & Nohejl, D. (2018), Multi-agent path finding on real robots: First experience with ozobots, *in* 'Ibero-American Conference on Artificial Intelligence', Springer, pp. 290–301.

Boyarski, E., Felner, A., Stern, R., Sharon, G., Tolpin, D., Betzalel, O. & Shimony, E. (2015), Icbs: Improved conflict-based search algorithm for multi-agent pathfinding, *in* 'Twenty-Fourth International Joint Conference on Artificial Intelligence'.

Felner, A., Li, J., Boyarski, E., Ma, H., Cohen, L., Kumar, T. S. & Koenig, S. (2018), Adding heuristics to conflict-based search for multi-agent path finding, *in* 'Proceedings of the International Conference on Automated Planning and Scheduling', Vol. 28, pp. 83–87.

Felner, A., Stern, R., Shimony, S., Boyarski, E., Goldenberg, M., Sharon, G., Sturtevant, N., Wagner, G. & Surynek, P. (2017), Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges, *in* 'International Symposium on Combinatorial Search', Vol. 8.

Google Scholar (2022).
URL: *https://scholar.google.com/*

Hart, P. E., Nilsson, N. J. & Raphael, B. (1968), 'A formal basis for the heuristic determination of minimum cost paths', *IEEE transactions on Systems Science and Cybernetics* **4**(2), 100–107.

Hönig, W., Kumar, T. S., Cohen, L., Ma, H., Xu, H., Ayanian, N. & Koenig, S. (2016), Multi-agent path finding with kinematic constraints, *in* 'Twenty-Sixth International Conference on Automated Planning and Scheduling'.

Hönig, W., Preiss, J. A., Kumar, T. S., Sukhatme, G. S. & Ayanian, N. (2018), 'Trajectory planning for quadrotor swarms', *IEEE Transactions on Robotics* **34**(4), 856–869.

Hönig, W., Li, J. & Koenig, S. (2020), 'Model ai assignments 2020: A project on multi-agent path finding (mapf)'.

Leigh, R., Louis, S. J. & Miles, C. (2007), Using a genetic algorithm to explore a*-like pathfinding algorithms, *in* '2007 IEEE Symposium on Computational Intelligence and Games', IEEE, pp. 72–79.

Li, J., Harabor, D., Stuckey, P. J., Felner, A., Ma, H. & Koenig, S. (2019), Disjoint splitting for multi-agent path finding with conflict-based search, *in* 'Proceedings of the International Conference on Automated Planning and Scheduling', Vol. 29, pp. 279–283.

Li, J., Surynek, P., Felner, A., Ma, H., Kumar, T. S. & Koenig, S. (2019), Multi-agent path finding for large agents, *in* 'Proceedings of the AAAI Conference on Artificial Intelligence', Vol. 33, pp. 7627–7634.

Ma, H. (2021), A competitive analysis of online multi-agent path finding, *in* 'Proceedings of the International Conference on Automated Planning and Scheduling', Vol. 31, pp. 234–242.

Ma, H. (2022), 'Graph-based multi-robot path finding and planning', *Current Robotics Reports* pp. 1–8.

Ma, H., Koenig, S., Ayanian, N., Cohen, L., Hönig, W., Kumar, T., Uras, T., Xu, H., Tovey, C. & Sharon, G. (2017), 'Overview: Generalizations of multi-agent path finding to real-world scenarios', *arXiv preprint arXiv:1702.05515* .

Ma, H., Li, J., Kumar, T. & Koenig, S. (2017), 'Lifelong multi-agent path finding for online pickup and delivery tasks', *arXiv preprint arXiv:1705.10868* .

Ma, H., Wagner, G., Felner, A., Li, J., Kumar, T. & Koenig, S. (2018), 'Multi-agent path finding with deadlines', *arXiv preprint arXiv:1806.04216* .

Silver, D. (2005), Cooperative pathfinding, *in* 'Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment', Vol. 1, pp. 117–122.

Stern, R. (2019), 'Multi-agent path finding–an overview', *Artificial Intelligence* pp. 96–115.

Stern, R., Sturtevant, N. R., Felner, A., Koenig, S., Ma, H., Walker, T. T., Li, J., Atzmon, D., Cohen, L., Kumar, T. S. et al. (2019), Multi-agent pathfinding: Definitions, variants, and benchmarks, *in* 'Twelfth Annual Symposium on Combinatorial Search'.

Stump, E., Michael, N., Kumar, V. & Isler, V. (2011), Visibility-based deployment of robot formations for communication maintenance, *in* '2011 IEEE international conference on robotics and automation', IEEE, pp. 4498–4505.

Sturtevant, N. R. (2012), 'Benchmarks for grid-based pathfinding', *IEEE Transactions on Computational Intelligence and AI in Games* **4**(2), 144–148.

Thomas, S., Deodhare, D. & Murty, M. N. (2015), 'Extended conflict-based search for the convoy movement problem', *IEEE Intelligent Systems* **30**(6), 60–70.

Yu, J. & LaValle, S. M. (2013), Structure and intractability of optimal multi-robot path planning on graphs, *in* 'Twenty-Seventh AAAI Conference on Artificial Intelligence'.