Solving Multi-Agent Pathfinding with Matching using A*+ID+OD

Ivar de Bruin[1] ,
Supervisors: Jesse Mulderij[1] , **Mathijs de Weerdt**[1]
[1]TU Delft

Abstract

This paper extends the Multi-Agent Pathfinding (MAPF) algorithm, A*+ID+OD, to be able to solve problems with matching. This extension still keeps the optimal and completeness properties of the original algorithm. Matching is added to the algorithm in both an exhaustive and heuristic manner. Exhaustive matching is further improved by adding a new layer of Independence Detection (ID) to reduce the number of matchings. Besides this, the pruning efficiency is increased by sorting the matchings based on the initial heuristic. The exhaustive matching method has been found to perform better than the heuristic matching method.

The exhaustive version of A*+ID+OD is finally compared to other extended MAPF algorithms which shows that on small maps, Conflict Based Min-Cost Flow (CBM) performs best as it is the only algorithm that does not use exhaustive matching. A*+ID+OD and Enhanced Partial Expansion A* (EPEA*) also perform well on open maps with multiple teams when compared to other exhaustive matching algorithms due to the addition of matching ID.

1 Introduction

The Dutch Railways (NS) is tasked with the maintenance and servicing of most of the trains in the Netherlands. During the night these trains have to be cleaned and serviced. The related scheduling and routing problems are collectively called the Train Unit Shunting and Servicing (TUSS) problem and it is NP-Hard [1].

Recently, efforts have been made to establish an upper bound for this problem [1]. One way of doing this is by creating a relaxation of TUSS and solving this relaxation with a complete and optimal algorithm. A first step in creating such a relaxation is a Multi-Agent Pathfinding (MAPF) problem which is extended with matching to a Multi-Agent Pathfinding with Matching (MAPFM) problem [1]. To be able to use this relaxation, the algorithm used for solving it must be complete and optimal.

MAPFM is also useful for other MAPF applications where it does not matter which specific agent goes to a goal, but rather which type of agent goes to a goal. This is for example the case for warehouse robots [2] but could also be used in other logistics applications where it only matters what kind of transport is used, not which specific ship or truck is used.

For solving MAPF, many algorithms have been developed such as A*+ID+OD [3], Enhanced Partial Expansion A* (EPEA*) [4], M* [5], Branch-and-Cut-and-Price (BCP) [6], Increasing Cost Tree Search (ICTS) [7] and Conflict Based Search (CBS) [8].

When it comes to extending MAPF problems with matching, comparatively little research has been done. One of the few works is an extension from CBS to Conflict Based Min-Cost Flow (CBM) done by Ma and Koenig which uses maximum flow to solve a Task Assignment and Pathfinding (TAPF) problem [9]. TAPF is a term that describes the same problem as MAPFM, however, Ma and Koenig optimize the makespan instead of the Sum of Individual Costs (SIC).

It is quite clear that BCP and CBS are the top-performing algorithms for solving MAPF problems [6], [10]. It is however not yet known what the best algorithms are when they are extended to solve MAPFM.

This paper will focus on extending A*+ID+OD with matching. A*+ID+OD utilizes Independence Detection (ID) and Operator Decomposition (OD) to reduce the branching factor of A* to make it viable to solve MAPF problems. Although it is not the fastest MAPF solver, it is a good candidate for MAPFM as it is complete and optimal and its utilization of heuristics allows for simple adaption to include matching.

The matching is implemented in both a heuristic and exhaustive manner. The exhaustive matching method is further improved by solving matchings independently when possible, as well as sorting the matchings based on the heuristic to improve pruning performance. The exhaustive and heuristic methods are then compared to find the best method. After this, the extended A*+ID+OD is compared to extensions of some of the other algorithms mentioned before to find the algorithm most suited for MAPFM.

This paper will first give a formal definition for the extension from MAPF to MAPFM. After this, the necessary extensions to A*+ID+OD will be explained. The different versions of the algorithm will then first be compared to each other and the best of these versions will be compared to other MAPFM algorithms. Finally, the reproducibility of this paper will be discussed.

2 Problem Description

This section will give a formal problem definition for both Multi-Agent Pathfinding (MAPF) and its extension to Multi-Agent Pathfinding with Matching (MAPFM).

2.1 Basic Multi-Agent Pathfinding

MAPF is a very general problem with many different applications and definitions. For this paper, the definitions proposed by Stern et al. in [11] will be used.

Input The problem input can be described as a MAPF problem with k agents as $\langle G, s, g \rangle$ where:

- $G = \langle V, E \rangle$ is an undirected graph of vertices V and edges E. [1]
- s is a list of k starting vertices such that $s_i \in V$ denotes the starting vertex of agent a_i
- g is a list of k goal vertices such that $g_i \in V$ denotes the goal vertex of agent a_i

Solution A solution to this problem describes a set of paths π such that a path π_i exists for each agent a_i. This path defines the location of agent a_i at a moment t as $\pi_i[t]$, starts at s_i, and ends at g_i. These paths should not conflict.

Conflicts Stern et al. define multiple conflicts, of these, two will be used in this paper [11].

- Vertex conflicts: A vertex conflict occurs when two agents are at the same vertex at the same time.
- Edge conflicts: An edge conflict occurs when two agents traverse the same edge at the same time in the opposite direction.

Goal behaviour When an agent reaches its goal, two things can happen: Either they disappear or they stay (also known as stay-at-target [11]). This paper uses stay-at-target. This means that even though the cost function will no longer increase when an agent has reached its goal, these agents can still cause vertex conflicts. If at any point an agent moves of its goal node, the time they spent on the goal will be added to the cost.

Objective There are two main objective functions that can be used to optimize the cost:

- Makespan: Makespan optimizes the difference between starting time and the point where all agents are finished. When all agents start at the same time, the makespan tries to keep the longest path as short as possible. This means that path lengths of 5 and 6 are better than path lengths of 1 and 7 as a maximum length of 6 is better than a maximum length of 7.
- Sum of Individual Costs (SIC): SIC optimizes the total sum of all the individual costs. This means that path lengths of 1 and 7 are better than path lengths of 5 and 6 as a total of 8 is better than a total of 11.

For this research, SIC is used as it is the most common objective function for search-based MAPF [11]. It also most closely resembles the real-life objective where minimizing the total time and fuel is often the most important.

[1] A 4-connected grid will be used for ease of implementation and comparison.

2.2 Extending Multi-Agent Pathfinding with matching

Matching in the context of MAPF means agents no longer have a specific goal to go to. Instead, both agents and goals belong to a specific team. Now, each agent has to go to a goal node that belongs to the same team as that agent. These teams are denoted by labelling the start and goal positions with a colour. This means our previous MAPF definition can now be extended with two arrays sc and gc resulting in the definition of a MAPFM problem as $\langle G, s, g, sc, gc \rangle$ where:

- sc is a list of k colours such that sc_i is the colour of s_i (and as such of agent a_i)
- gc is a list of k colours such that gc_i is the colour of g_i

With this definition, every colour should occur an equal number of times in both sc and gc. A valid solution to a MAPFM problem is a set of paths where each agent a_i ends at a unique goal node g_j such that $sc_i = gc_j$.

3 A*+ID+OD with Matching

This section will explain the developed algorithm. First, an explanation of the base A*+ID+OD algorithm will be provided, followed by an explanation of the two matching possibilities, heuristic and exhaustive. Finally, two improvements for exhaustive matching are introduced.

3.1 A*+ID+OD

A*+ID+OD is a MAPF algorithm defined by Standley in [3]. The algorithm is an extension of A* [12], a common heuristic-based pathfinding algorithm. An advantage of A* is that as long as the used heuristic is admissible, the algorithm will be optimal and complete. The extension of A* to A*+ID+OD keeps this property [3].

A big disadvantage of base A* is that its branching factor increases exponentially with regards to the number of agents when it is adapted to MAPF [3]. This is mitigated by extending the algorithm with Operator Decomposition (OD) and Independence Detection (ID) which are both described in more detail in [3].

Operator Decomposition In MAPF, A* states are expanded by moving all agents at the same time. This causes the exponential branching factor, as all combinations of moves for all agents are calculated. OD reduces this branching factor by moving agents one at a time. The branching factor under OD is the number of possible moves for an agent, which is constant. Moving agents one at a time does however increase the path depth to a solution by a factor k where k is the number of agents. Combined, these two performance differences result in a net performance increase.

Because OD moves agents one at a time, there are two types of nodes. Nodes where all agents have yet to move are called standard nodes, while nodes where some agents have moved while others have not are called intermediate nodes. Only standard nodes have to be put in the CLOSED list as overlapping intermediary nodes can only be generated by the same standard node, because agents always move in the same order.

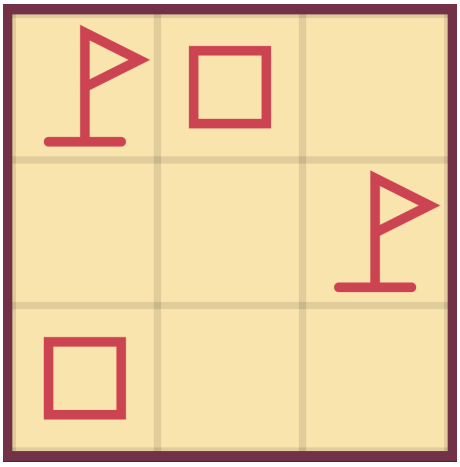

Figure 1: Benchmark showing heuristic matching will cause unnecessary conflicts. Squares are starting positions and flags are goal positions

Independence Detection In many MAPF problems, groups of agents never get in each other's vicinity. In these cases, it is a waste of performance to compute these agents at the same time as their pathfinding is completely independent. ID as such tries to find solutions for individual agents while ignoring the other agents. It then tries to resolve any conflicts between agents by giving priority to one and trying to find a solution with equal cost for the other. If this fails for both agents, it will merge them and solve them at the same time. This process repeats until no more conflicts are found at which time a valid solution has been found.

Conflicts are further avoided by adding a Collision Avoidance Table (CAT) which keeps track of the paths of all agents. This can then be used to keep track of any conflicts a new path will cause with other groups. This number of conflicts is an effective tiebreaker for node expansion in the A* solver [3].

3.2 A*+ID+OD with Heuristic matching

The first way of implementing matching is to change the heuristic used by the basic A* solver. The normal heuristic used in A* for MAPF is the distance for each agent to their goal as calculated by a breadth-first search [3]. This heuristic is quick to compute and a good relaxation from a multi-agent problem to a single-agent problem.

The heuristic can be altered to instead give the distance to the closest goal of a certain team by initializing the breadth-first search with all goals belonging to that team instead of just one goal node. The changed heuristic is still admissible and the algorithm remains complete and optimal. The heuristic has however become worse as it no longer keeps in mind that agents need to go to unique goals. This can be shown using Figure 1. Heuristic matching will send both agents to the top left goal, causing a conflict and as such causing ID to combine the agents into one group after trying to solve the conflict. If the agents are instead sent to unique goals, no conflicts will occur and ID will never have to combine the agents into one group.

3.3 A*+ID+OD with Exhaustive matching

The second matching option, exhaustive matching, tries to reduce the matching problem back to a normal MAPF problem by solving the problem for all possible matchings, as can be seen in Algorithm 1. This is done by first calculating all possible goal assignments, possibly in a lazy manner, and then running the basic MAPF A*+ID+OD solver on each of the goal assignments and choosing the best solution.

Algorithm 1 Exhaustive matching (ID adapted from [3])

```
1:  function EXHAUSTIVE_MATCHING
2:      matchings ← list of all possible valid matchings
3:      best_solution ← null
4:      best_cost ← ∞
5:      for all matchings do
6:          solution, cost ← solve matching with IDSolver
            with best_cost as maximum
7:          if cost < best_cost then
8:              best_cost ← cost
9:              best_solution ← solution
10:         end if
11:     end for
12:     return best_solution
13: end function
14:
15: function IDSOLVER(matching, max_cost)
16:     assign each agent to a group
17:     plan a path for each group
18:     fill CAT with every path
19:     repeat
20:         Find a conflicts between two groups G1 and G2
21:         if G1 and G2 have not conflicted before then
22:             fill illegal move table with the G2 paths
23:             find a set of paths with the same cost for G1
24:             if failed to find such a set then
25:                 fill illegal move table with the G1 paths
26:                 find a set of paths with the same cost for
                    G2
27:             end if
28:         end if
29:         if failed to find an alternative set of paths for G1
            and G2 then
30:             merge G1 and G2 into a single group G3
31:             maximum ← max_cost - sum of the path
                costs for all agents not in G3
32:             cooperatively plan G3 with maximum
33:         end if
34:         update CAT with new paths
35:     until no conflicts occur
36:     solution ← paths of all groups combined
37:     return solution
38: end function
```

A downside to this approach is that running the algorithm fully for each matching is incredibly slow. To make this matching method useful, some way of pruning the matchings needs to be added. This can be done by keeping track of the cost of the best solution so far, and then discarding

any A* nodes with a higher f cost than this value, where $f = cost + heuristic$.

The presence of ID in the algorithm makes this more complicated. Because ID initially runs A* on a subset of agents, the cost for the solution to this sub-problem will be significantly lower than the cost for the full solution. To still be able to prune in these sub-problems, the maximum cost for a sub-problem needs to be calculated. This is done by subtracting the cost of the path for each agent that is not present in the sub-problem from the maximum cost. (See line 31, Algorithm 1) If no such path has been found yet, the initial heuristic will be used instead. Any solution with a cost higher than this maximum cost will cause the total solution to go over the original maximum cost and as such, the matching can be pruned immediately without affecting the optimality of the algorithm. If a sub-problem now has no solution, the full problem can also not be solved within the given maximum cost and the matching can be pruned.

Exhaustive matching with pruning is still limited by the exponential growth of the matchings as it still needs to go through a part of each matchings computation. As such this method of matching will be limited to a certain number of agents unless a way is found to prune large numbers of matchings without at least partially computing each one. This method remains optimal and complete as all matchings are attempted and only pruned if it is impossible for that matching to be better than the current best matching.

3.4 Matching ID extension

To improve the performance of exhaustive matching, a version of ID can be implemented on top of matching as can be seen in Algorithm 2. The idea behind it is the same as the idea behind the A* implementation of ID (Section 3.1): Solving multiple sub-problems is faster than solving the whole problem at once. This also applies to generating matchings. Initially, all teams are solved individually, with teams only being combined when they conflict with each other. Because there are now fewer agents being solved simultaneously, there will also be fewer matchings to evaluate.

To decrease the chances of conflicts occurring, a Collision Avoidance Table (CAT) is added to this version of ID. This means that the actual A* solver can get multiple tables, one for each layer of ID in the algorithm.

Another optimization is to combine all teams with only one agent immediately, as solving these groups together does not create more matchings and the A* ID already attempts to solve the agents individually.

This extension should increase the performance on maps with few obstacles and multiple teams. On obstacle dense maps such as mazes, it will however struggle to avoid conflicts and as such it will most likely just do unnecessary work, similar to standard ID (experimentally tested in Section 4.3).

This algorithm remains limited by the exponential growth of the number of matchings, similar to normal exhaustive matching. However, this algorithm should be able to solve problems with more agents and teams than normal exhaustive matching because the number of matchings can be reduced on certain maps.

Algorithm 2 Exhaustive matching ID

```
1: create a group for each team
2: combine all groups of size one
3: create empty CAT
4: for all groups do
5:     solve the group with Exhaustive Matching passing along the CAT
6:     update the CAT with the found solution
7: end for
8: repeat
9:     simulate execution of all paths until a conflict between two groups G_1 and G_2 occurs
10:    combine G_1 and G_2 into G_3
11:    solve G_3 with Exhaustive Matching passing along the CAT
12:    update the CAT with the new solution
13: until no conflicts remain
14: solution ← paths of all groups combined
15: return solution
```

3.5 Sorting extension

The efficiency of the current exhaustive matching implementation relies for a large part on the efficiency of the pruning of matchings. This pruning efficiency can be improved by sorting the matchings based on the initial heuristic [13]. This can be implemented by first putting the possible matchings through a priority queue. Once the best matching in the queue is worse than the best solution so far, all remaining matchings in the queue can be discarded. This extension should increase pruning efficiency, especially when the initial heuristics are spread out (Experimentally tested in Section 4.3).

4 Experiment Results

This section will first describe the experimental setup as well as an inherent problem with MAPFM comparisons. It will then compare the different exhaustive matching implementations of A*+ID+OD and compare these to heuristic matching. Finally, A*+ID+OD will be compared to other extended MAPF algorithms.

4.1 Experimental Setup

All experiments were run on the same virtual computer with a 12 core 12 thread Intel Xeon E5-2683 running at 2 GHz with 8 GB of RAM and all benchmarks are proven solvable [13].

The algorithms are compared using two map types which test different aspects of the algorithms:

- Maze maps: Maze maps are maps with small corridors that are fully connected. These maps focus more on conflict resolution than conflict avoidance and matching as agents cannot easily pass each other. They are representative of places such as shunting yards where trains also cannot move around each other.
- Obstacle maps: Obstacle maps have an obstacle density of around 10% These maps have a substantial amount of space for agents to manoeuvre around each other and as such focus more on matching and conflict avoidance.

All maps are 20 by 20 and were generated in sets of 200 for each set of parameters. The solver has two minutes to solve a map. This timeout was chosen as it is long enough to be able to compare runtimes, while still making sure that the total benchmark time remains reasonable. The benchmarks were then run on an increasing number of agents that were either in a single team or evenly spread over three teams.

In addition to these, Berlin_1_256 [11] was also used. For this map, all 25 even scenarios given on https://movingai.com/benchmarks/mapf/ were run with a uniform team assignment. This map was used as it is larger and as such will hopefully amplify the difference between algorithms.

4.2 Survivorship bias

An inherent problem with benchmarking MAPFM is that the difficulty of a problem only partially depends on the number of agents. Figures 5 and 7 show that for higher numbers of agents the solvers are only able to complete a subset of the problems within the timeout. The average runtime then becomes less representative of the actual parameters as it is only computed over the solved benchmarks, resulting in survivorship bias. This is mitigated by only using runtime plots for lower numbers of agents.

4.3 Exhaustive matching

Three versions of exhaustive matching will be compared to each other to find out which extensions improve performance:

1. Exhaustive matching as described in Section 3.3
2. Exhaustive matching with ID as described in Section 3.4
3. Sorted exhaustive matching with ID as described in Section 3.5

Figure 2 shows the runtime performance of exhaustive matching implementations on maze maps. It can be seen that matching ID has a very minor effect on runtime while sorting consistently improves runtime. This makes sense as it is difficult to solve teams independently on such dense maps, but the initial heuristics will vary quite a bit based on the matching, increasing the performance gain from sorting.

Figure 3 shows the runtime performance of exhaustive matching versions on obstacle maps. For one team, ID has no effect, however, for three teams it has a significant effect for higher agent numbers. Adding sorting on top leads to a further improvement for one team.

Figure 4 shows the clearest difference. This figure compares the exhaustive matching methods on Berlin_1_256. Here, 20 agents were uniformly spread over a varying number of teams. The larger map size and larger maximum team size has amplified the difference between the algorithms. It can be seen that ID improves performance significantly, while sorting improves performance even further. It can also be seen that for smaller teams, sorting adds very little. This is likely because ID already makes the problems significantly smaller which reduces the usefulness of sorting.

Figures 5 and 7 show the percentage of mazes and obstacles that have been solved within the timeout of two minutes for all matching versions. These figures also show that ID increases performance on obstacle maps and that adding sorting increases performance on maze maps.

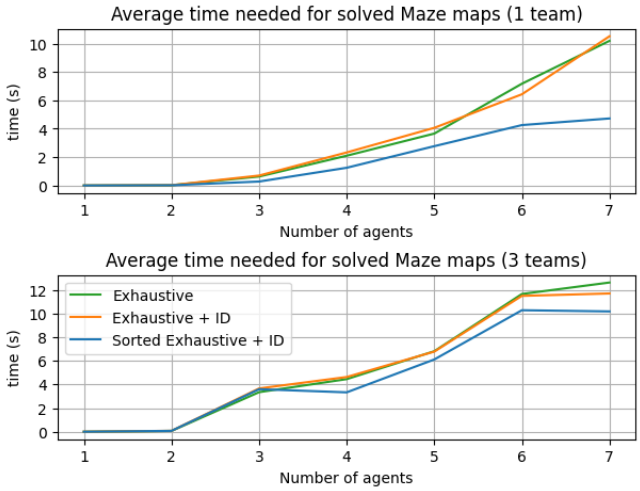

Figure 2: Exhaustive matching performance on Maze maps

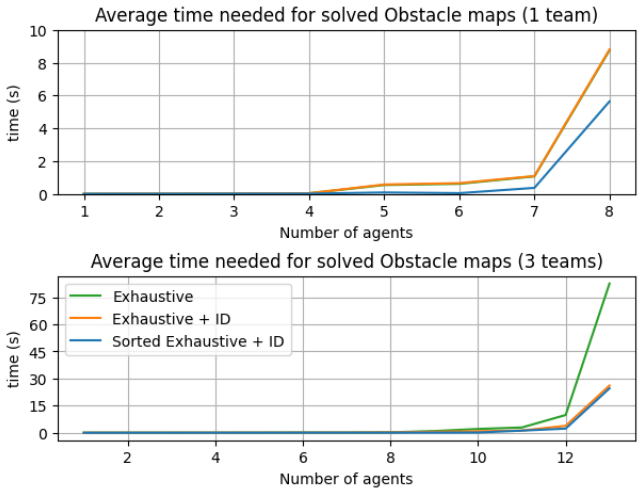

Figure 3: Exhaustive matching performance on Obstacle maps

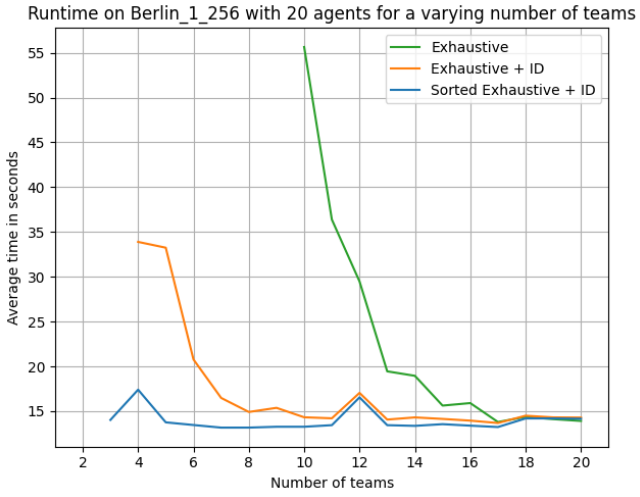

Figure 4: Exhaustive matching performance on Berlin_1_256

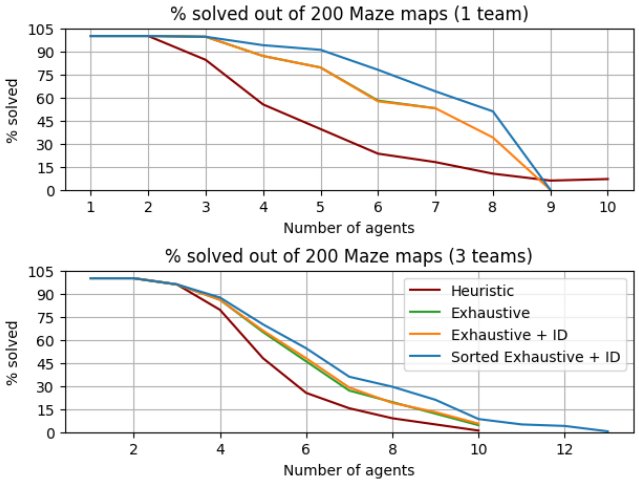

Figure 5: Percentage of solved Maze maps

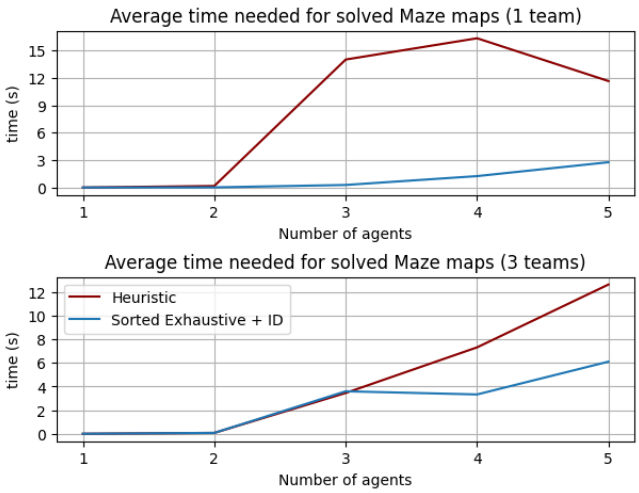

Figure 6: Heuristic vs Exhaustive matching on Maze maps

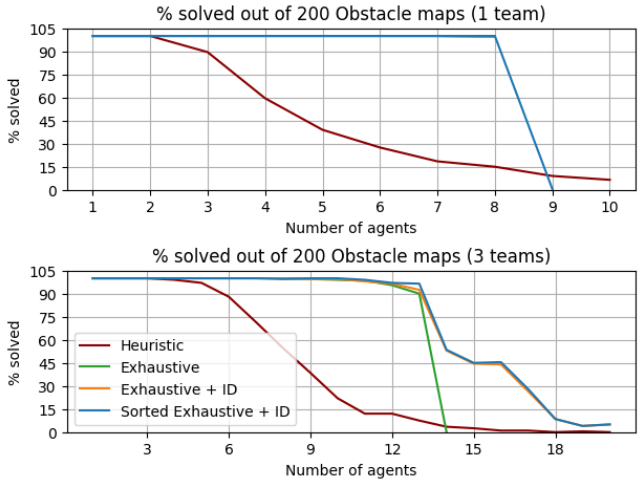

Figure 7: Percentage of solved Obstacle maps

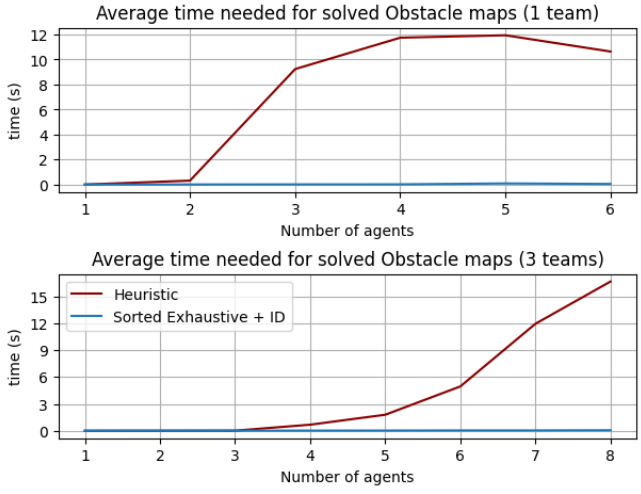

Figure 8: Heuristic vs Exhaustive matching on Obstacle maps

4.4 Exhaustive versus heuristic

Next, heuristic matching will be compared to exhaustive matching. Figures 5 and 7 compare how many maps each algorithm was able to solve. They show that heuristic matching with three teams is always worse than the best exhaustive matching method. However, for one team, it is slightly better for higher agent numbers as heuristic matching does not scale exponentially with the number of matchings. However, heuristic matching can only solve around 10% of those maps and is way worse than exhaustive matching for all maps with fewer agent numbers.

Figures 6 and 8 compare the runtime of sorted exhaustive matching with matching ID to heuristic matching. From these figures, it can be seen that even on the maps that can be solved by heuristic matching, it is slower than exhaustive matching. This makes it clear that the current matching heuristic is worse than exhaustive matching as it is both slower and able to solve fewer maps.

4.5 Comparing A*+ID+OD to other algorithms

This subsection will compare different MAPF algorithms extended for solving MAPFM to find out which one can solve the most maps. The following algorithms will be compared:

- A*+ID+OD with sorted exhaustive matching and matching ID (3.5)
- CBM using flow to solve matching [14]
- EPEA* with sorted exhaustive matching and matching ID [13]
- ICTS with approximate sorted exhaustive matching [15]
- M* with sorted exhaustive matching [16]

Figure 9 compares the algorithms on maze maps. For three teams, the algorithm does not matter as the problem here is often the number of collisions. However, CBM performs very well for one team as it is the only algorithm that does not use exhaustive matching.

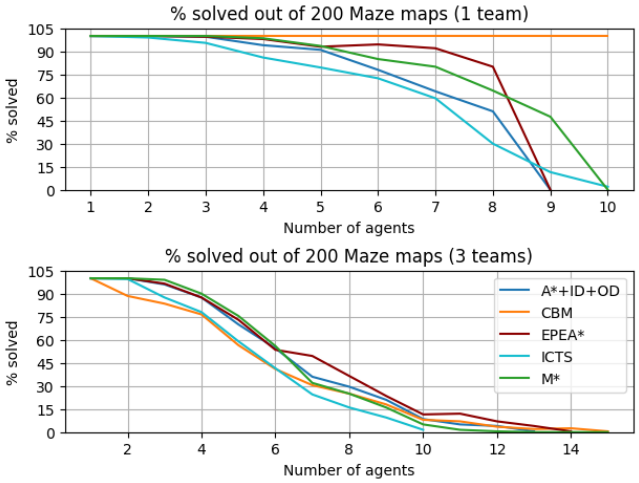

Figure 9: MAPFM algorithm comparison on Maze maps

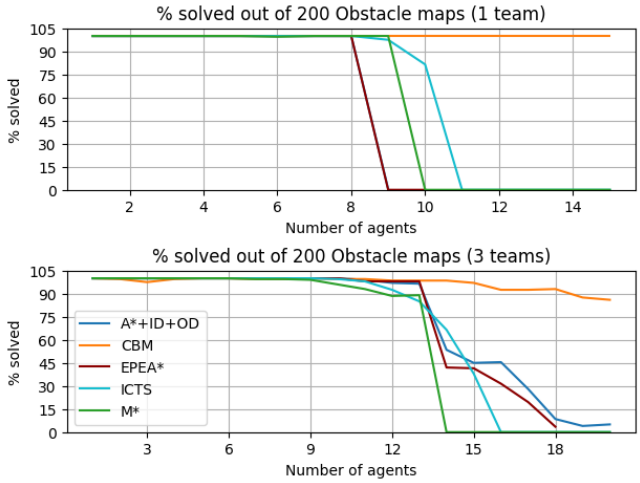

Figure 10: MAPFM algorithm comparison on Obstacle maps

Figure 10 compares the algorithms on obstacle maps. It can be seen that most algorithms perform similarly for obstacle maps with one team. However, CBM outperforms all other algorithms.

CBM is again the best on obstacle maps with three teams, however, A*+ID+OD and EPEA* also perform well, due to the fact that these are the only exhaustive matching algorithms that use matching ID.

5 Reproducibility

This research was done in a way that allows for easy reproducibility. The full code base for A*+ID+OD can be found online.[2] Furthermore, the randomly generated benchmarks were stored in files and are available online, together with the scripts that generated them.

The algorithm is also fully deterministic and as such all results can be reproduced with the only differences occurring due to computer performance differences.

The comparison between algorithms was done in a way that is as fair as possible. All algorithms are implemented in Python 3 with the focus being on optimizing the algorithms, not the codebase. However, Conflict Based Min-Cost Flow (CBM) does use libraries implemented in C++, granting it a performance advantage. It is however not possible to negate this.

6 Conclusion

This paper describes the extension from Multi-Agent Pathfinding (MAPF) to Multi-Agent Pathfinding with Matching (MAPFM). After this, various ways of introducing matching to the MAPF algorithm, A*+ID+OD are shown and compared. From this comparison, it can be concluded that exhaustive matching performs better than heuristic matching.

It was also found that both of the exhaustive matching extensions led to a performance increase. Matching Independence Detection (ID) improved performance significantly on open maps with multiple teams and sorting the matchings led to a further improvement on maps where the initial heuristics vary significantly between matchings, such as mazes.

An important characteristic of the original A*+ID+OD algorithm is that it is optimal and complete. Both of these properties are still present for both exhaustive matching and heuristic matching.

When different algorithms are compared, it is clear that different exhaustive matching algorithms perform about the same. The exception is on open maps with multiple teams. Here A*+ID+OD and EPEA* perform much better compared to the others, due to matching ID.

As CBM is the only algorithm that does not use exhaustive matching, it performed much better with the exception being maze maps with three teams, where all algorithms are equal, as the number of collisions is the problem here.

7 Future work

There are two areas of improvement based on this paper. Firstly, the current implementation of A*+ID+OD can still be improved in a few ways. Secondly, the results from this paper also raise questions that require further study.

7.1 Algorithm improvements

There are some areas in which the implementation of A*+ID+OD can still be improved. First of all, there is the implementation of partial expansion. This extension has already been described for the original A*+ID+OD algorithm in an appendix of [4], which described how partial expansion can be adapted to work with Operator Decomposition. Partial expansion could reduce the high memory usage that occurs when A*+ID+OD runs for a prolonged period of time. Partial expansion only keeps track of nodes that do not increase the f cost. If an expansion creates nodes with a higher f cost than that of the parent, the parent will be put back in the OPEN list with the lowest of these costs. This ensures that partial expansion remains optimal and complete while using less memory. However, as the given description for partial

[2] Online codebase: https://github.com/ivardb/Astar-OD-ID

expansion in combination with OD was minimal, we were unable to implement it successfully for this paper.

Another important improvement could be made by finding a better matching heuristic, which could make the algorithm scale better to higher numbers of agents. Exhaustive matching will always be limited by the number of agents as the number of matching grows exponentially with the number of agents. Heuristic matching on the other hand does not have this limitation. As such, a new matching heuristic that makes heuristic matching viable would make the algorithm scale better.

7.2 Result improvements

There are two questions for further study that follow from the results in this paper. The first problem is that the current results have a standard deviation that is very high when problems are grouped based on the number of agents and the number of teams. This shows that it is not fully understood what makes MAPF difficult. There are clearly other factors besides the number of agents and the number of teams. The question is then what other metrics are good indicators for difficulty and how can benchmarks be reliably generated according to these metrics. With such an improved metric, comparisons between algorithms would also become more accurate as it can more clearly be shown for which problems an algorithm performs well and for which ones it does not.

The other question is related to larger maps. The current results seem to indicate that on large open maps with multiple teams A*+ID+OD and EPEA* should perform the best as CBM scales worse with map size than these algorithms. This is interesting to investigate as many real-world applications take place in larger spaces and algorithms should not only scale with the number of agents but also with map size.

References

[1] J. Mulderij, B. Huisman, D. Tönissen, K. van der Linden, and M. de Weerdt, *Train unit shunting and servicing: A real-life application of multi-agent path finding*, 2020. arXiv: 2006.10422 [cs.MA].

[2] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Magazine*, vol. 29, no. 1, p. 9, 2008. DOI: 10.1609/aimag.v29i1.2082.

[3] T. Standley, "Finding optimal solutions to cooperative pathfinding problems," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 24, 2010, pp. 173–178.

[4] M. Goldenberg, A. Felner, R. Stern, G. Sharon, N. Sturtevant, R. C. Holte, and J. Schaeffer, "Enhanced partial expansion A*," *Journal of Artificial Intelligence Research*, vol. 50, pp. 141–187, 2014, ISSN: 10769757. DOI: 10.1613/jair.4171.

[5] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds," in *2011 IEEE/RSJ international conference on intelligent robots and systems*, IEEE, 2011, pp. 3260–3267.

[6] E. Lam, P. Le Bodic, D. D. Harabor, and P. J. Stuckey, "Branch-and-cut-and-price for multi-agent pathfinding.," in *IJCAI*, 2019, pp. 1289–1296.

[7] G. Sharon, R. Stern, M. Goldenberg, and A. Felner, "The increasing cost tree search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 195, pp. 470–495, 2013.

[8] 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, 2015.

[9] H. Ma and S. Koenig, *Optimal target assignment and path finding for teams of agents*, 2016. arXiv: 1612.05693 [cs.AI].

[10] G. Sharon, R. Stern, A. Felner, and N. Sturtevant, "Meta-agent conflict-based search for optimal multi-agent path finding," in *Fifth Annual Symposium on Combinatorial Search*, 2012, pp. 97–104.

[11] R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, E. Boyarski, and R. Bartak, *Multi-agent pathfinding: Definitions, variants, and benchmarks*, 2019. arXiv: 1906.08291 [cs.AI].

[12] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968, ISSN: 0536-1567. DOI: 10.1109/TSSC.1968.300136.

[13] J. de Jong, "Multi-Agent Pathfinding with Matching using Enhanced Partial Expansion A*," 2021.

[14] R. Baauw, "Adapting CBM to optimize the Sum of Costs," 2021.

[15] T. van der Woude, "Multi-Agent Pathfinding with Matching using Increasing Cost Tree Search," 2021.

[16] J. Dönszelmann, "Matching in Multi-Agent Pathfinding using M*," 2021.

Acronyms

A*+ID+OD A* with ID and OD. 1–4, 6–8

BCP Branch-and-Cut-and-Price. 1

CAT Collision Avoidance Table. 3, 4

CBM Conflict Based Min-Cost Flow. 1, 6–8

CBS Conflict Based Search. 1

EPEA* Enhanced Partial Expansion A*. 1, 6–8

ICTS Increasing Cost Tree Search. 1, 6

ID Independence Detection. 1–8

MAPF Multi-Agent Pathfinding. 1–4, 6–8

MAPFM Multi-Agent Pathfinding with Matching. 1, 2, 4–7

OD Operator Decomposition. 1, 2, 7, 8

SIC Sum of Individual Costs. 1, 2

TAPF Task Assignment and Pathfinding. 1

TUSS Train Unit Shunting and Servicing. 1