Extending the Multi-Label A* Algorithm for Multi-Agent Pathfinding with Multiple Waypoints

Arjen Ferwerda
a.ferwerda@student.tudelft.nl
Supervised by: Jesse Mulderij, Mathijs de Weerdt
Delft University of Technology

21/06/2020

Abstract

Multi-Agent Pathfinding (MAPF) is a problem in which the goal is to plan paths for distinct agents while avoiding collisions between agents. We consider a new variation of MAPF, namely MAPF with multiple waypoints (MAPFW), where agents are required to visit a set of intermediary locations before visiting their end goal. MAPFW may have interesting applications, such as in the field of train scheduling and routing. To solve MAPFW problems we present the new algorithm Extended Multi-Label A* (EMLA*), which is based of the existing MLA* algorithm. Experimental evaluation shows that Heuristic-Based EMLA* (HB-EMLA*) for unordered MAPFW outperforms other algorithms when it comes to the number of agents and waypoints it is able to handle. However, HB-EMLA* struggles to find solutions to instances which are *not well-formed*, as resting agents may block planning agents preventing a valid plan from being found. HB-EMLA* generally outperforms other algorithms when it comes to run time of larger instances. This comes at the cost of solution quality, where the solutions provided by EMLA* are 30% worse than the best solution of the compared algorithms. Lastly, set benchmarks show that a simple *Nearest Waypoint* heuristic generally outperforms other tested heuristics for HB-EMLA*.

1 Introduction

Multi-Agent Pathfinding (MAPF) is a classical problem in which the goal is to plan paths for distinct agents while avoiding collisions between agents. Multi-Agent Pickup and Delivery (MAPD) is a variant of MAPF where agents must select tasks where they must travel to a pickup-location followed by a drop-off-location [Grenouilleau et al., 2019]. Finding an optimal solution to MAPF problems, and by extension MAPD problems, is NP-hard [Nebel, 2019]. In fact, the complete composite search-space, which considers all possible positions of all agents at each time step, grows exponentially with the number of agents [Ryan, 2008]. As such, it is essential to find algorithms which can as-fast-as-possible compute paths for a large number of agents. Similarly, it is essential to find fast algorithms for real-time applications, such as automated warehouses [Wurman et al., 2008] and autonomous vehicles [Stern et al., 2019].

There are numerous optimal algorithms for the MAPF problem [Lam et al., 2019, Sharon et al., 2015, Yu and LaValle, 2013], and there are several algorithms for the MAPD problem as well [Grenouilleau et al., 2019, Ma et al., 2017]; however, these algorithms are not always applicable to other variants of the MAPF problem. Therefore, one aim of this paper is to explore an additional variant with useful applications. One variant, which might be useful for train shunting and service, is MAPF with multiple waypoints (MAPFW) [Mulderij et al., 2020]. This is a variant where agents are assigned a set of intermediate locations (waypoints) which they must visit before reaching their final goal. Since there are no existing algorithms for MAPFW, the goal of this paper is to explore MAPFW, and to develop new algorithms to effectively solve it.

One way to solve MAPD problems is with the The MLA* algorithm [Grenouilleau et al., 2019]. In MLA* agents are assigned tasks, and then they perform a search for their pickup location and end goal in a combined search. Since MAPD is very similar to MAPFW, but with a single waypoint, we believe that MLA* may be suitable to be extended so that agents visit multiple waypoints before reaching their goal.

The main contributions of this paper are as follows. First, we present an Extended Multi Label A* (EMLA*) algorithm to solve MAPFW. Using set benchmarks we demonstrate that EMLA* is able to solve a large number of instances. We show that it performs well with a large number of agents and waypoints, provided instances are *well-formed*. However, EMLA* struggles when instances are *not well-formed*, as resting agents can block agents being planned, which prevents EMLA* from finding valid plans. Using set benchmarks we show that EMLA* generally performs faster than other algorithms, with the performance being relatively the fastest when instances are *well-formed*, and when there are many conflicts between agents. However, the average solution quality of EMLA* is 30% worse than an optimal algorithm, although this is highly dependent on the features of the graph. Lastly, using set benchmarks we demonstrate that a simple *Nearest Waypoint* heuristic generally outperforms other tested heuristics for solving unordered MAPFW with Heuristic-Based EMLA*.

2 Multi-Agent Pathfinding Problem

We formulate the MAPF problem as follows. The input to the problem with k agents is a tuple $\langle G, s, e\rangle$, where $G = (V, E)$ is a connected undirected graph, $s : [1, 2, ..., k] \to V$ maps agents to their start location, and $e : [1, 2, ..., k] \to V$ maps agents to their end locations. Time is discretized starting at 0, and on each time step an agent is situated on one of the graph vertices. Between time steps the agent may move to an adjacent vertex using an edge in E, or the agent may remain at their current vertex [Stern et al., 2019].

A plan, or path p of length $n \in \mathbb{Z}_+$ for agent i is a sequence $(v_0, v_1, ..., v_{n-1})$, where $p_i[t] = v_t$ denotes that agent i is at vertex v_t at time step t. The plan starts at the agent's starting position and ends with the agent's ending position. Formally, $p_i[0] = s(i)$, and $p_i[n-1] = e(i)$. For all time steps $t \geq n$, agent i is assumed to remain at $e(i)$ [Mulderij, 2020, Stern et al., 2019].

The goal of MAPF is to find a plan for each agent such that each plan is valid. For a plan to be valid, all adjacent nodes in the plan must be adjacent in the graph (connected by an edge), or the nodes must be the same, e.g. for an agent i and associated plan p_i, $\{p_i[m-1], p_i[m]\} \in E$ or $p_i[m-1] = p_i[m]$ for all $1 \leq m \leq n$, where n is the length of plan p_i. Furthermore, a valid plan has no collisions, which occur when agents attempt to cross the same edge or vertex simultaneously.

Formally, we recognize three types of conflicts following the definitions from [Stern et al., 2019], namely edge, vertex, and swap conflicts. We define these as follows, considering a pair of agent plans p_i and p_j:

- **Edge Conflict.** An *edge conflict* occurs iff any two agents are planned to cross an edge in the same direction at the same time i.e. there is an edge conflict between i and j iff $p_i[m] = p_j[m]$ and $p_i[m+1] = p_j[m+1]$ for some time step m.
- **Vertex Conflict.** A *vertex conflict* occurs iff any two agents are planned to converge to the same vertex on a given time step. Formally, there is a swapping conflict between i and j when $p_i[m] = p_j[m]$ for some time step m.
- **Swap Conflict.** A *swap conflict* occurs iff any two agents are planned to swap places with each other by crossing the same vertex in opposite directions i.e. there is a swap conflict between i and j iff $p_i[m] = p_j[m+1]$ and $p_i[m+1] = p_j[m]$ for some time step m.

The minimal solution to a MAPF instance is a set P consisting of valid plans $p_i \in P$, where p_i is the valid plan of agent i, such that the sum of the lengths of all plans is minimal. We define this sum to be $\Sigma_{1 \leq i \leq k} |p_i|$ for a MAPF instance with k agents. We use *solution quality* to refer to this sum.

2.1 Multi-Agent Pathfinding with Unordered Waypoints

Multi-Agent Pathfinding with Unordered Waypoints, or unordered MAPFW, is a variant of MAPF where agents are assigned a set of intermediate locations (waypoints) which they must visit before reaching their final goal.

The input to this problem is the same as classical MAPF, with the addition of a set of waypoints for each agent $w : [1, 2, ..., k] \rightarrow \wp(V)$ or a mapping of the k input agents to a subset of the vertices of the input graph.

Agents must visit all of their assigned waypoints, meaning the plan p_i of agent i must contain all the assigned waypoints. Formally, $\forall x \in w(i)$, $x \in p_i$ where p_i is the plan of i. Other constraints on the plans from the classical problem still apply, such as $p_i[0] = s(i)$, $p_i[n-1] = e(i)$, etc.

2.2 Multi-Agent Pathfinding with Ordered Waypoints

We also consider a variant of MAPF with multiple ordered waypoints (Ordered MAPFW), where agents are required to visit a set of intermediary waypoints in a given order before reaching the end goal. The input to ordered MAPFW is the same as classical MAPF, with the addition of a linearly ordered set of waypoints for each agent $w : [1, 2, ..., k] \rightarrow \wp(V)$. We denote $\leq_i$ to be the total order of $w(i)$ for agent i, i.e. $a \leq_i b$ iff i must visit waypoint a before waypoint b. We require that each agent visits each node in the set of waypoints in the order they are given. Agents may traverse waypoints in any order i.e. an agent may traverse node x before node y even if $y \leq x$, however for a plan to be valid each waypoint in the agents plan must be preceded by all previous waypoints based on the total order $\leq_i$ for agent i. We formalize this as:

$\forall x, y \in w(i)$, $x \leq_i y \Longrightarrow \exists a, b((p_i[a] = x) \wedge (p_i[b] = y) \wedge (a \leq b))$

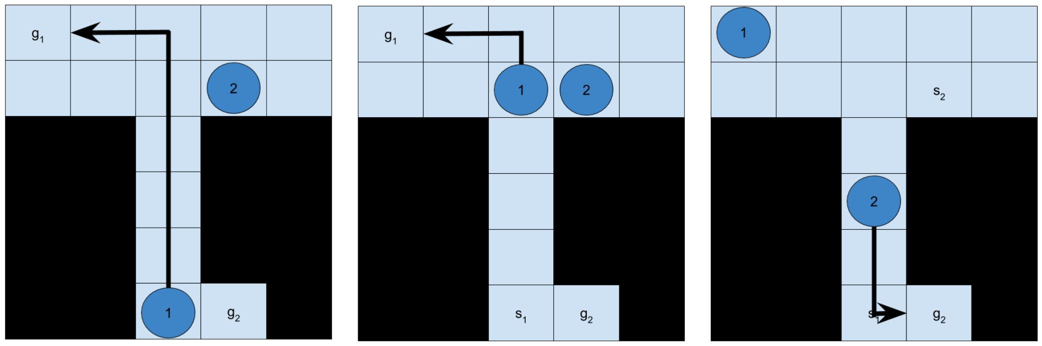


Figure 1: Example of a grid-based graph where the path of one agent blocks the path of another. Agent 2 utilizes *wait-at-start* to wait for agent 1 to pass. We use s_n and g_n to denote the starting and goal location of the nth agent respectively. Black squares are impassable.

3 Summary of MLA*

Recall that in MAPD, agents must visit pickup locations before visiting their goal locations (of their chosen task). In MLA*, each agent is tasked with finding a combined path, visiting the pickup and drop off location, using a singular A*-based search. To do this the authors use a binary label for each search node of the A* search. If the label is 1 the agent is searching for the pickup location, and if the label is 2 the agent is searching for the drop off location. The search commences with the initial node having label 1, which is then switched to label 2 once a path to the pickup location has been found. Presumably, conflicts with other agents are avoided with use of a token, which is similar to a global reservation table, although it is not specified. When an agent finds a valid path, the path is added to the reservation table, and the reserved paths are considered blocked by subsequently planned agents.

It is not clear how agents can consider wait actions in MLA*, so we assume agents will only wait when they are not able to find a path to any task.

The approach of MLA* is similar to a *decoupled* approach, which provides a non-optimal solution. This is contrasted with a *centralized* approach, where a search is performed on (a part of) the complete abstract state space, and which provides an optimal solution [Stern et al., 2019]. Since MLA* does not provide optimal solutions, our algorithm also does not provide optimal solutions.

4 Extended Multi-Label A*

We introduce Extended Multi-Label A* (EMLA*) as a generalized and extended version of MLA* intended to solve Ordered MAPFW. MLA* associates a label with the pickup location and one with the drop off location. We generalize this idea by associating a label with each waypoint, and one label with the goal location. In practice, EMLA* performs a combined A*-based search for all waypoints and the end goal, where agents search for each consecutive waypoint, before finally searching for the goal location. As in MLA*, agents are planned sequentially, although EMLA* uses an arbitrary ordering of agents.

EMLA* follows the outline of the A* algorithm [Hart et al., 1968]. To conduct a search of the graph, we define for each search node n the variables g_n, l_n, and p_n, which denote respectively the nodes g value (the number of time steps elapsed), the nodes label, and the nodes position. The label specifies which waypoint the agent is searching for, or whether the agent is searching for the end goal (after having visited all waypoints). For an agent with k waypoints, $l_n = 1$ indicates that the agent is searching for the first waypoint, $l_n = 2$ indicates that the agent is searching for the second waypoint, etc. $l_n = k + 1$ indicates that the agent is searching for the end goal. Nodes also have an f-value, and an h-value, where $f_n = g_n + h_n$. The f-value is used to prioritize which search node should be expanded, and nodes with a smaller f-value are expanded first. The h-value is a heuristic for the distance of the search node to the current goal of the agent (be it a waypoint or the end goal).

We compute the h-value as follows. For an agent i we denote the jth waypoint using w^i_j, e.g. w^i_5 is agent i's 5th waypoint. For an agent i with k waypoints:

$$h_n = \begin{cases} h(p_n, w^i_1) + h(w^i_1, w^i_2) + \ldots + h(w^i_k, e(i)) & l_n = 1 \\ h(p_n, w^i_2) + h(w^i_2, w^i_3) + \ldots + h(w^i_k, e(i)) & l_n = 2 \\ \vdots & \\ h(p_n, w^i_k) + h(w^i_k, e(i)) & l_n = k \\ h(p_n, e(i)) & l_n = k+1 \end{cases}$$

and we define $h(\cdot, \cdot)$ as the Manhattan distance between two positions. This is a generalization of the h-value in MLA* [Grenouilleau et al., 2019].

In Algorithm 1 we give the pseudo-code for our EMLA* algorithm, which finds a plan for a single agent i. We run EMLA* for an agent, and we create an initial search node, which we add to a queue. We then iterate as long as the queue is not empty, or until we find a path. The node with the lowest f-value is removed from the queue, and checks are performed to see if the node has the same position as a waypoint, or the end goal, depending on the label of the node. When the node is not at one of the agent's goals, it is expanded, meaning neighbouring nodes are added to the queue.

To compute the plans for all agents, we take an arbitrary ordering of agents and run EMLA* sequentially for each agent. When an agent finds a valid plan, the plan is added to a global reservation table, which is then used by subsequently planned agents.

4.1 Avoiding Conflicts

To avoid conflicts between agents, we introduce *wait-at-start* (WAS), where agents that cannot find a path due to conflicts will wait at their start location. Agents can also wait next to a reserved location while still searching for a path, which we refer to as *neighbouring-wait* (NW). Agents may be blocked by other agents, which results in agents sometimes not being able to reach a waypoint, or the end goal. To resolve such a conflict, we use WAS to force the blocked agent to wait at their starting location for a certain number of time steps. Figure 1 gives an example where one agent must wait for the other agent at the start. s_n and g_n denote the starting and goal location of the nth agent respectively. In this example, agent 2 waits for 4 time steps at its starting location before moving towards its goal.

We define two methods to determine the WAS time:

- **Linear-k WAS** is a method of determining the WAS time by adding an integer k to the WAS time each failed iteration of the algorithm. Agents initially wait 0 time steps,

Algorithm 1 Extended Multi-label A*

```
1: Create initial node $n_0$ with $l_{n_0} = 1$, $g_{n_0} = was\_time$, $h_{n_0} = 0$, $f_{n_0} = 0$, and $p_{n_0}$ = the agents starting location and add $n_0$ to $Q$
2: Determine $l^*$                                          ▷ Label corresponding to end goal
3: while $Q$ is not empty do
4:     Remove the node $n$ from $Q$ which has the smallest $f_n$ and add $n$ to closed
5:     if $p_n \in resting$ or $p_n$ is reserved at time step $g_n$ then
6:         continue (reject this node and go to the top of while loop)
7:     else if $p_n$ = the position of $l^w$th waypoint then
8:         Create $n'$ with $l_{n'} = l_n + 1$, $g_{n'} = g_n$, and $p_{n'} = p_n$
9:         $Q \leftarrow [n']$, $closed \leftarrow \{\}$
10:        continue
11:    else if $l_n == l^*$ and $p_n = e(i)$ then
12:        return the found path
13:    else
14:        for all adjacent vertices $\notin closed$ do
15:            Expand node $n$ if adjacent vertex not reserved at time step $g_n$.
16:            Add an NW node if applicable.
17:        end for
18:    end if
19: end while
20: Increase $was\_time$ and run the algorithm again with $WAS$ enabled until a path is found.
```

but if they cannot find a path to their goal or waypoint, they try the search again after waiting k time steps at their start location. Each failed iteration afterwards adds a further k time steps to the WAS time.

- **Exponential-k WAS** is a method of determining the WAS time by multiply the WAS time by an integer k each failed iteration of the algorithm. Agents initially wait 0 time steps, but if they fail to find a path to their goal or waypoint, they attempt to search again after waiting 1 time step at their start location. Each subsequent failed iteration multiplies this WAS time by k.

The aim with these two approaches is to give an option for both solution quality, and run time. Using linear-k WAS might give better solutions since the WAS time can be increased by small increments depending on k, whereas exponential-k WAS might decrease the run time when agents must wait for a large amount of time.

A reservation table is used to keep track of the various planned paths for each planned agent. For an entry (p, t), we consider the position p to be impassable at time step t by any agent. Let us consider a search node n at position p, and at time step t. Normally, if a vertex at position p_a adjacent to p is reserved in the next time step $t + 1$, that position is considered blocked, and is not added to the search. However, in these situations we use NW to add a search node with position p_a at time step $t + 2$. This search node represents two actions: a wait at position p for 1 time step, followed by a move to position p_a in the next time step. This allows agents to wait next to a reserved location, before moving onto that location in the next time step provided it is no longer reserved. Figure 2 gives an example.

It should be noted that the NW and WAS are not effective at resolving all kinds of conflicts. Agents are planned sequentially, and agents are assumed to rest at their starting

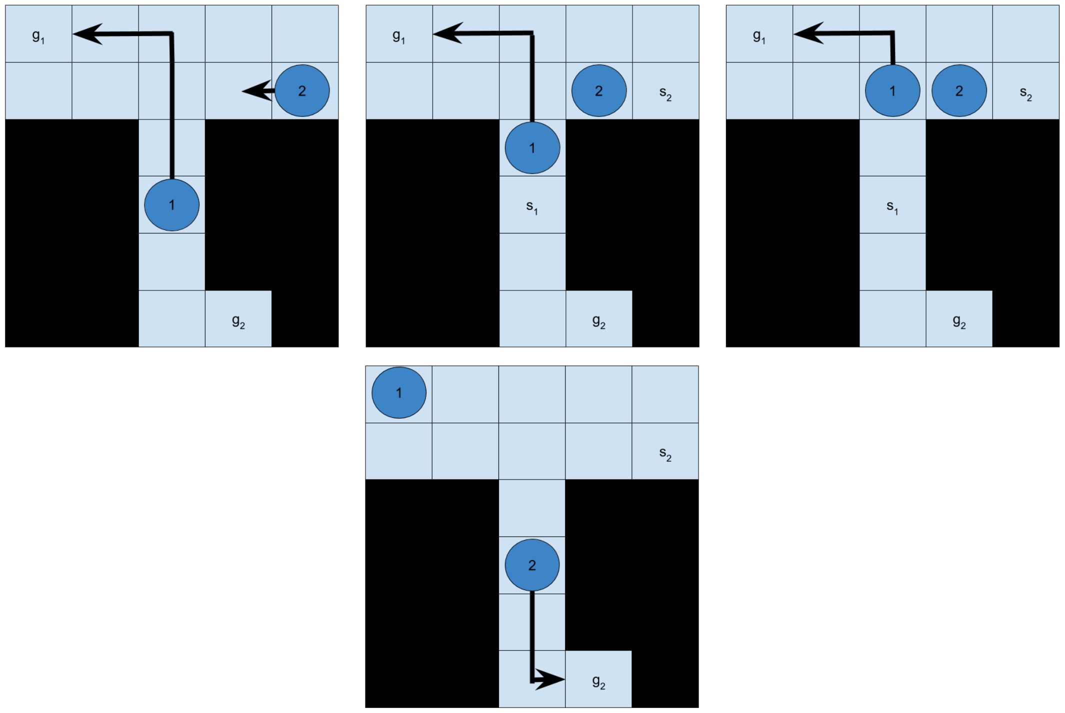


Figure 2: Example of a grid-based graph illustrating *neighbouring-wait*. Agent 2's path to their goal is blocked by agent 1, but agent 2 is able to wait 1 time step at a neighbouring node to wait for agent 1 to pass.

location before being planned. This means that it is possible that some problem instances cannot be solved by our EMLA* algorithm, since a resting agent might be blocking the path of an agent who is at that moment being planned. Consider the example in Figure 3, where the given instance is not solvable using EMLA*, since it requires the simultaneous planning of two agents. In general, we refer to such instances as being *not well-formed*.

Specifically, we define the inverse, a *well-formed* instance, as follows:
A **well-formed** instance is an instance where for each connected component of the input graph, all waypoints and the goal location of an agent are reachable without passing through the start or end point of another agent. More formally, for an instance with k agents, for all agents i, there must exist a path from $s(i)$ to each waypoint, and a path from $s(i)$ to $e(i)$, that does not pass through any starting vertex $s(j)$, or any ending vertex $e(j)$ for any j where $1 \leq j \leq k, j \neq i$.

4.2 Heuristic-Based EMLA*

One limitation of EMLA* is that it can only solve ordered MAPFW instances. To solve unordered MAPFW instances, we introduce Heuristic-Based EMLA* (HB-EMLA*).

Unlike ordered MAPFW, there is no ordering given for the waypoints, so agents are free to choose the order in which they visit their waypoints. A flexible way to determine the order of waypoints is by using heuristics to determine the waypoint the agent will attempt

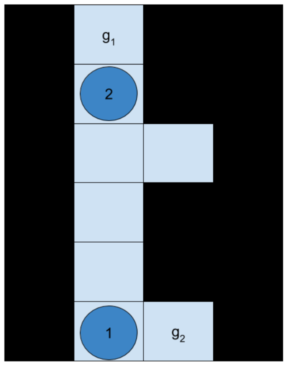


Figure 3: Example of a grid-based graph instance which is *not well-formed*. Agent 1 and 2 are both located at their starting locations, and both agents are blocking the path of the other agent. Regardless of the ordering of the agent planning, neither agent can find a path to their goal.

to reach. Essentially, when the agent is at the starting location, and every time the agent reaches a waypoint, we compute a heuristic for each waypoint to determine which waypoint should be visited by the agent. Once the agent has visited all waypoints, the agent simply proceeds to the end goal, similar to achieving the label l^* in EMLA*. In practice, the algorithm starts by selecting a waypoint for the agent based on the heuristics, and every time the agent reaches the selected waypoint, a new waypoint is selected, and the label is increased.

There are several heuristics available in order to compute which waypoint to select. We give several:

- The **Nearest Waypoint** (NW) heuristic simply computes the distance from the agent to each waypoint, and then selects the closest waypoint. For the distance calculation we use the Manhattan distance, though other distance calculations are possible.
- The **Nearest Waypoint * Nearest Neighbour** (NW*NN) heuristic computes the distance from the agent to each waypoint, and then from each waypoint the distance to the nearest neighbour (the closest other waypoint). The waypoint that is selected by the agent is the waypoint where the product between the agent-waypoint distance and the waypoint-nearest-neighbour distance is minimal.
- The **Nearest Waypoint + Nearest Neighbour** (NW+NN) heuristic is similar to the previous heuristic, but instead of calculating the product, the sum of the agent-waypoint and waypoint-nearest-neighbour distance is used instead.

The goal of these heuristics is to minimize the distance the agent has to travel. Thus, the intention of the NW heuristic is for an agent to simply visit the waypoints closest to it. The idea of the NW*NN and NW+NN heuristics is to minimize the time the agent walks back and forth between waypoints by having the agent prioritize clustered sets of waypoints, rather than distant and isolated waypoints.

5 Experimental Evaluation

The goal of our experimental evaluation is to test the how effective HB-EMLA* is at solving instances with many agents and waypoints. Additionally, we explore the conditions under which HB-EMLA* outperforms other (optimal) algorithms. Lastly, we compare different heuristic approaches for HB-EMLA* to see which is most effective under different conditions.

To help answer our research questions, we perform the following three experiments:

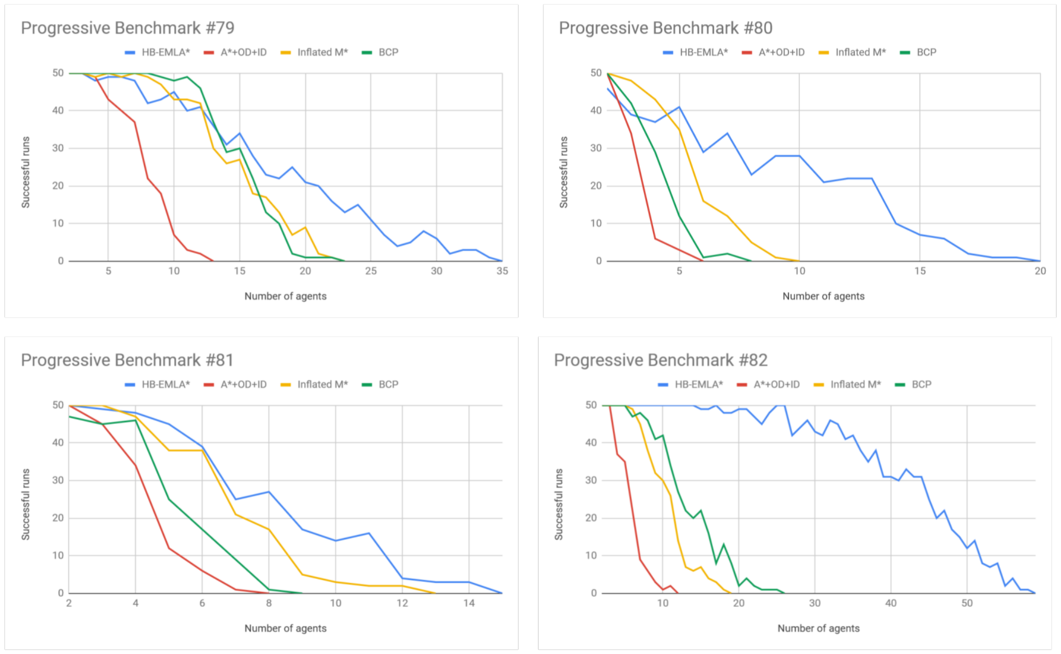
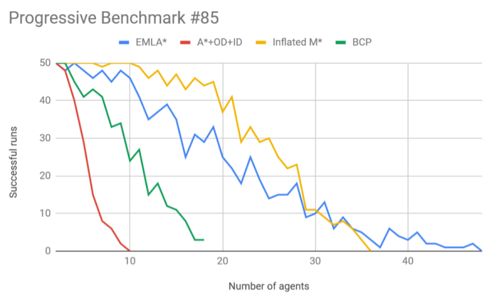


Figure 4: Performance of HB-EMLA* on progressive benchmarks from *mapfw.nl*, as compared to the MAPFW algorithms A*+OD+ID, Inflated M*, and BCP.

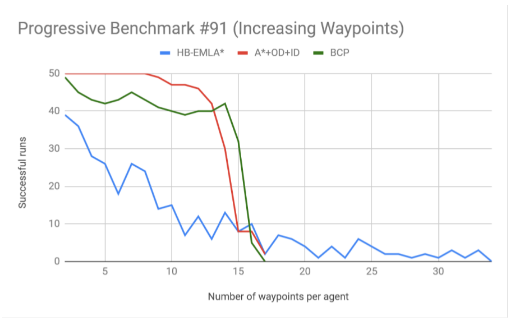
- To generally measure the effectiveness of EMLA* we use progressive benchmarks. These benchmarks use a set graph structure, but randomly place an increasing number of agents, and randomly place a set number of waypoints per agent.
- To compare the performance of our algorithm with that of other (optimal) algorithms we make use of set benchmarks. These benchmarks use both a set graph structure, and set agent and waypoint locations.
- To compare different heuristic approaches for our HB-EMLA* we again use set benchmarks.

To run these benchmarks we use the website *mapfw.nl*, which allows users to run benchmark problems and submit their solutions to the website, allowing evaluation and comparisons of various algorithms [Jadoenathmisier and Siekman, 2020]. We use the *mapfw* python library to obtain the benchmarks from the *mapfw.nl* server, and then run them on our algorithm. All benchmarks, and all solutions can be seen on *mapfw.nl*.

In some of our experiments, we compare our results with those of 4 other algorithms, which were developed in parallel with EMLA*. CBSW is an extension of the Conflict-Based Search algorithm [Sharon et al., 2015, Jadoenathmisier, 2020]. BCP is an extension of the branch-and-cut-and-price for MAPF algorithm [Lam et al., 2019, Michels, 2020]. A*+OD+ID is an extension of an A*-based algorithm which combines operator decomposition and independence detection to refine the search-space [Standley, 2010, Siekman, 2020]. Lastly, Inflated M* is a non-optimal extension of the optimal M* algorithm for MAPF [Wagner and Choset, 2011, Dijk, 2020].



(a) Benchmark with an increasing number of agents.



(b) Benchmark with an increasing number of waypoints per agent.

Figure 5: Additional experiment showing performance of HB-EMLA* on two new benchmarks from *mapfw.nl*.

5.1 Measuring Effectiveness of EMLA*

We ran our HB-EMLA* algorithm on the progressive benchmarks from *mapfw.nl*. For each benchmark, 50 runs with random agent and waypoint locations are started every time the number of agents is increased, and the benchmark ends when all of the 50 runs fail. A run fails when the algorithm is not able to find a valid solution, or when the run takes longer than the timeout limit, which depends on the benchmark. The aim is to increase the number of agents as much as possible before the algorithm fails all 50 runs.

Figure 4 illustrates the results of HB-EMLA*, and 3 comparative algorithms, on the progressive benchmarks, which have a 20 second timeout. The vertical axis shows the number of successful runs, and the horizontal axis shows the number of agents in each run. In this experiment our HB-EMLA* algorithm uses linear-1 WAS for conflict resolution, and uses the Nearest Waypoint heuristic for waypoint selection.

Our algorithm outperforms the other algorithms when it comes to number of agents it can handle. What is not shown in the graphs is the solution quality. It seems that our algorithm generally sacrifices solution quality for speed, which allows it to handle more agents.

It is notable that our algorithm has relatively poor performance on benchmark #81, and exceptional performance on benchmark #82. Benchmark #81 consists of many corridors, where only 1 agent can pass through a corridor at a time. Benchmark #82 is a large graph, with a lot of open space, but where all 5 agent waypoints overlap with the waypoints of the other agents. The result is that, despite causing numerous conflicts, agents in benchmark #82 are usually able to find a path to their waypoints. On the contrary, the many corridors in benchmark #81 limit the number of paths agents are able to take to their waypoints, or end goal. Additionally, it is more likely that resting agents block the path of other agents in benchmark #81, meaning it is sometimes impossible for agents to find a path to their goals.

As a followup experiment, we ran our HB-EMLA* algorithm on two new progressive benchmarks. Benchmark # 85 is similar to benchmarks #79-82, although the graph is much larger, and with fewer choke points. Benchmark #91 is a 16x16 grid, where for each run 20% of the vertices are randomly made impassable. Additionally, in benchmark # 91 the number of waypoints per agent is increased rather than increasing the number of agents. Figure 5 summarizes the results of these experiments.

Benchmark	HB-EMLA*	CBS	A*+OD+ID	Inflated M*	BCP
5	**14 ms**	1.77 s	27.61 s	556 ms	201 ms
8	**71 ms**	1 min, 58 s	1 min, 22s	27.77s	-
10	618 ms	33.49 s	-	**244 ms**	14.18 s
11	8 ms	12 ms	24 ms	**4 ms**	6 ms
12	**12 ms**	299 ms	3.59 s	133 ms	25 ms
19	**510 ms**	-	-	35.91 s	-
24	**129 ms**	6 min, 46 s	-	1 min, 8 s	1.46 s
27	12 ms	63 ms	92 ms	14 ms	**7 ms**
59	**166 ms**	1 min, 38 s	-	1 min, 16 s	5 min, 37 s
64	9 ms	10.92 s	6.21 s	**7 ms**	592 ms

(a) Run time for each algorithm to compute all agent plans.

Benchmark	HB-EMLA*	CBS	A*+OD+ID	Inflated M*	BCP
5	135	**119**	**119**	**119**	**119**
8	1187	**821**	**821**	**821**	-
10	2162	**1590**	-	1598	1590
11	111	**100**	**100**	**100**	**100**
12	123	**115**	**115**	**115**	**115**
19	4066	-	-	**2921**	-
24	1996	**1542**	-	1551	1994
27	173	**169**	**169**	**169**	**169**
59	1312	**824**	-	850	**824**
64	151	**96**	**96**	193	**96**

(b) Solution quality for each algorithm. The value is the sum of length of agent plans.

Table 1: Performance of HB-EMLA* and other algorithms on 10 random benchmarks. The smallest values for each benchmark are in bold.

HB-EMLA* performs well on benchmark # 85, since it lacks small choke points, and due to the size it is often well-formed. However, it seemingly struggles with the large size of the benchmark, as many more runs fail due to a timeout. Comparatively, it performs worse than M*, unless the number of agents is sufficiently large. This may be due to the lack of optimizations in EMLA*, and the large search space which is not sufficiently reduced by our algorithm.

On benchmark #91 our algorithm underperforms compared to BCP and A*+OD+ID, although it is able to handle some of the instances with a large number of waypoints. This may be due to the same limitation as in benchmark #81, where instances with more waypoints increase the likelihood that an instance is not well-formed. When a waypoint for an agent is placed on the start- or end location of another agent, the instance is not well-formed, and thus is often not solvable using EMLA*. Furthermore, due to the random placement of obstacles in benchmark # 91, instances may feature choke points, further increasing the odds that instances are not well-formed.

Based on our experiments, it seems that our algorithm is most effective when instances are open with few obstacles, and when instances are well-formed. The performance suffers when there are many choke points, or when there are a large number of agents or waypoints on a relatively small graph.

5.2 Algorithmic Comparisons

We ran our HB-EMLA* algorithm on 10 randomly selected benchmarks, from a list of 17 manually selected candidate benchmarks. Candidate benchmarks were selected from *mapfw.nl* based on size, and whether or not EMLA* could find a solution. More details on the candidates and these criteria can be found in appendix A.

As in 5.1, our HB-EMLA* algorithm uses linear-1 WAS and the Nearest Waypoint heuristic. Table 1 summarizes the results, showing the benchmark id's, and the results of HB-EMLA* and 4 comparison algorithms. Table 1a shows the run times for each algorithm on each benchmark, and Table 1b shows the solution quality. Values in bold are the smallest values for each benchmark.

It is clear that our algorithm generally under performs when it comes to solution quality. On average, HB-EMLA*'s solution is **30%** worse than the best solution; however, there is a large variation between benchmarks. Our algorithm provides worse solutions on large benchmarks with many waypoints, as the ordering of waypoints has a large impact on the solution quality in those situations.

This is contrasted with the generally good run time performance of HB-EMLA*. There is a large variation between benchmarks, as well as between the algorithms. In the worst case, the performance is comparable with Inflated M*, or BCP. In the best case, it performs 458 times better (benchmark #59).

Benchmark #59 consists of a relatively large graph with many waypoints, and long rows of obstacles with some space between rows. This instance resembles the instances used to simulate warehouses in other studies [Ma et al., 2017]. HB-EMLA* shows similar results to other similar benchmarks (such as benchmark #64 which is a candidate benchmark), demonstrating the effectiveness of EMLA* at solving such instances.

5.3 Heuristic Comparisons

For this experiment, we ran the same benchmarks as in section 5.2, but using variations of our HB-EMLA*. Specifically, we used the three distinct heuristics, NW, NW+NN, and NW*NN. All variants use Linear-1 WAS.

Table 2 shows the performance of HB-EMLA* using different heuristics. Table 2a shows the run times for each variation, and Table 2b shows the solution qualities.

For most benchmarks, Nearest Waypoint outperforms the other heuristics in terms of solution quality. In terms of run time, there is no clear winner, although NW + NN considerably underperforms in benchmark #19. Interestingly, although NW * NN generates the worst solution for benchmark #19, it has the fastest run time. Comparatively, NW + NN generates a better solution, but the run time is around 10 times worse. Benchmark #19 resembles a maze, with large open spaces connected with long corridors acting as choke points. It is possible that the large difference in run time is due to the particular ordering of waypoints, which causes an agent to not be able to find a path to their goals, thus requiring WAS to find a path. Unfortunately, this can be quite costly if an agent must wait for a long time, since the agent will perform a search on the graph each time the WAS time is increased. In situations were an agent must wait for a long time at the start, the run time can be decreased by using a larger k when determining the WAS time. We ran benchmark #19 using the NW + NN heuristic, and with linear-10 WAS, and the run time was reduced to 580 ms, while the solution quality remained 4327. Exponential-k WAS may provide a similar benefit, although it is not clear to what extent.

Benchmark	Nearest wp.	NW + NN	NW * NN
5	**9 ms**	**9 ms**	**9 ms**
8	**52 ms**	55 ms	55 ms
10	510 ms	**202 ms**	232 ms
11	**4 ms**	5 ms	**4 ms**
12	**6 ms**	8 ms	**6 ms**
19	413 ms	3.98 s	**368 ms**
24	114 ms	135 ms	**103 ms**
27	**6 ms**	8 ms	**6 ms**
59	135 ms	**100 ms**	176 ms
64	**4 ms**	7 ms	6 ms

(a) Run time for HB-EMLA to compute all agent plans using different heuristics.

Benchmark	Nearest wp.	NW + NN	NW * NN
5	**135**	**135**	**135**
8	**1187**	1301	1301
10	2162	**2099**	**2099**
11	**111**	**111**	**111**
12	**123**	**123**	**123**
19	**4066**	4327	4414
24	**1996**	2130	2030
27	**173**	193	193
59	**1312**	1402	1383
64	151	182	**150**

(b) Solution quality for HB-EMLA* using different heuristics.

Table 2: Performance of HB-EMLA* using different heuristics for waypoint selection. All variations use Linear-1 WAS.

Since it is hard to predict how the heuristics will perform, it is advisable to use NW since it generally provides satisfactory results. This is the reason why we use NW for our other experiments.

6 Ethicality and Reproducibility

An effort has been made to conduct this research in an ethically responsible way. The results are presented in an honest and straightforward manner, and the implications of the conclusions have been thoroughly considered.

In order to facilitate the reproduction of the results in this paper, the benchmarks which have been used are publicly available at *mapfw.nl* and the source code is available at *github.com/ArjenFerwerda/EMLA*.

This paper was written for academic purposes, as part of the CSE3000 course offered by the Technical University Delft. It was written under supervision of the university, and as such, we believe there are no conflicts of interest.

7 Discussion & Reflection

This research was affected by a few limitations which influenced the results. Due to a limitation in resources and time, most of the data from the set benchmarks were obtained locally, on different machines, rather than from a centralized computer or server. This means that some of the differences in run times and agent numbers between the other algorithms might be due to differences in hardware. Additionally, some data from the progressive benchmarks were obtained locally, though for all algorithms benchmarks #79 through #82 were run on a centralized server to mitigate the potential hardware difference, and to improve consistency.

The choice of programming language may affect the results, as the algorithms in this research were written in Python, while the BCP algorithm was written in C++ [Michels, 2020]. A* + OD + ID, Inflated M*, and CBSW were also written in Python [Siekman, 2020, Bestebreur, 2020, Jadoenathmisier, 2020].

8 Conclusions

This paper attempts to find an effective way to extend the existing MLA* algorithm to solve the MAPFW problem. We have implemented two algorithms, EMLA* for ordered waypoints, and HB-EMLA* for unordered waypoints. We explore under what conditions our algorithms outperform other (optimal) algorithms, and we explore different heuristic approaches to solve unordered MAPFW.

Experimental evaluation shows that our algorithms are able to solve a large number of instances, and it performs well with a large number of agents and waypoints, provided instances are *well-formed*. However, when instances are *not well-formed*, EMLA* may fail to find solutions, since resting agents may block the path of agents being planned. Furthermore, EMLA* performs relatively poorly on random instances with small graphs relative to the number of agents or waypoints, since instances are more likely to be *not well-formed*.

EMLA* generally performs faster than other algorithms, with the performance being relatively the fastest when instances are *well-formed*, and when there are many conflicts between agents. In these conditions, HB-EMLA* is often able to find alternate paths to agents' goals when conflicts arise, or WAS and NW can be used to avoid conflict. When instances feature many choke points, the performance of EMLA* suffers, though this can be alleviated by using linear-k or exponential-k WAS with a large k-value. This comes at the cost of solution quality, as alternate paths tend to be longer than the optimal paths, and WAS can introduce a large amount of unnecessary wait time. When problem instances become sufficiently large, the run time of EMLA* also suffers. This may be due to a lack of optimizations, and the larger search space.

Using benchmarks, we have found that using a simple nearest-waypoint heuristic is an effective way to order an agents' waypoints, when compared to other heuristics. However, since it is a heuristic approach, good performance is not always guaranteed, and the ordering is often non-optimal. This also results in a drop in solution quality when compared to optimal algorithms on instances with many waypoints, or with maze-like instances.

9 Future Work

While our experiments focused on using a non-optimal algorithm for unordered problems, future work may also focus more on solving ordered MAPFW using optimal or non-optimal algorithms.

Future work should also aim to make results more comparable by running different algorithms on the same machine. To facilitate this, it is essential that future researchers aim to make their algorithms as reproducible as possible, possibly by publishing source-code.

Although linear-k WAS, exponential-k WAS, and NW help agent planning when conflicts arise, future work should explore further options for conflict resolution. Dynamic agent planning may re-plan agents to wait when they cause conflicts for future agents. Similarly, resting agents may move to nearby vertices devoid of other agents or waypoints when they block other agents, which is similar to moving agents to "free endpoints" in the HBH algorithm [Grenouilleau et al., 2019].

A major limitation of EMLA* is its inability to solve most instances which are *not well-formed*. Future work should focus on possible solutions to this problem, such as by adding dynamic agent ordering. Agents can be planned as necessary to move out of the way of other agents, or to re-plan agents who cause conflicts with subsequently planned agents.

HB-EMLA* can perform well on large instances, but more optimizations can help improve performance when instances become extremely large. An option to improve the A* search on is to use better heuristics, such as the abstract heuristic calculated by the Reverse Resumable A* algorithm [Silver, 2005]. Alternatively, for an instance with static graph structure, the abstract path length (the length of the path between two vertices without agents) can be computed for every vertex with polynomial overhead.

While the *Nearest Waypoint* heuristic has satisfactory results for most instances, future work should focus on testing different approaches in determining waypoint ordering. One approach may focus on using (optimal) Travelling Salesperson Problem solvers to determine waypoint ordering, which is an approach already taken by the MAPFW CBS algorithm [Jadoenathmisier, 2020].

10 Acknowledgements

We would like to thank Jesse Mulderij and Mathijs de Weerdt for their guidance and supervision. We would like to thank Jeroen van Dijk, Noah Jadoenathmisier, Andor Michels, Timon Bestebreur, and Stef Siekman for their cooperation and help with this research.

References

[Bestebreur, 2020] Bestebreur, T. (2020). Analysis of the influence of graph characteristics on MAPFW algorithm performance. *TU Delft Repository.*

[Dijk, 2020] Dijk, J. v. (2020). Solving the multi-agent path finding with waypoints problem using subdimensional expansion. *TU Delft Repository.*

[Grenouilleau et al., 2019] Grenouilleau, F., van Hoeve, W.-J., and Hooker, J. N. (2019). A multi-label a* algorithm for multi-agent pathfinding. In *ICAPS*, pages 180–185.

[Hart et al., 1968] Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, 4(2):100–107.

[Jadoenathmisier, 2020] Jadoenathmisier, N. (2020). Extending CBS to efficiently solve MAPFW. *TU Delft Repository.*

[Jadoenathmisier and Siekman, 2020] Jadoenathmisier, N. and Siekman, S. (2020). *MAPFW Benchmarks*. Accessed 31th of May, 2020. https://mapfw.nl/.

[Lam et al., 2019] Lam, E., Le Bodic, P., Harabor, D. D., and Stuckey, P. J. (2019). Branch-and-cut-and-price for multi-agent pathfinding. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 1289–1296. International Joint Conferences on Artificial Intelligence Organization.

[Ma et al., 2017] Ma, H., Li, J., Kumar, T., and Koenig, S. (2017). Lifelong multi-agent path finding for online pickup and delivery tasks. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 837–845.

[Michels, 2020] Michels, A. C. (2020). Multi-agent pathfinding with waypoints using Branch-Price-and-Cut. *TU Delft Repository.*

[Mulderij, 2020] Mulderij, J. (2020). MAPF problem description. Personal communication.

[Mulderij et al., 2020] Mulderij, J., Huisman, B., Tönissen, D., van der Linden, K., and de Weerdt, M. (2020). Train unit shunting and servicing: a real-life application of multi-agent path finding.

[Nebel, 2019] Nebel, B. (2019). On the computational complexity of multi-agent pathfinding on directed graphs. arXiv preprint arXiv:1911.04871.

[Ryan, 2008] Ryan, M. (2008). Exploiting subgraph structure in multi-robot path planning. *Journal of Artificial Intelligence Research*, 31:497–542.

[Sharon et al., 2015] Sharon, G., Stern, R., Felner, A., and Sturtevant, N. R. (2015). Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40 – 66.

[Siekman, 2020] Siekman, S. (2020). Extending A* to solve multi-agent pathfinding problems with waypoints. *TU Delft Repository.*

[Silver, 2005] Silver, D. (2005). Cooperative pathfinding. In *Proceedings of the 1st Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2005*, pages 117–122.

[Standley, 2010] Standley, T. S. (2010). Finding optimal solutions to cooperative pathfinding problems. In *AAAI.*

[Stern et al., 2019] Stern, R., Sturtevant, N. R., Felner, A., Koenig, S., Ma, H., Walker, T., Li, J., Atzmon, D., Cohen, L., Kumar, T. K. S., Boyarski, E., and Barták, R. (2019). Multi-agent pathfinding: Definitions, variants, and benchmarks. In *SOCS.*

[Wagner and Choset, 2011] Wagner, G. and Choset, H. (2011). M*: A complete multirobot path planning algorithm with performance bounds. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3260–3267.

[Wurman et al., 2008] Wurman, P., D'Andrea, R., and Mountz, M. (2008). Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 29:9–20.

[Yu and LaValle, 2013] Yu, J. and LaValle, S. M. (2013). Planning optimal paths for multiple robots on graphs. In *2013 IEEE International Conference on Robotics and Automation*, pages 3612–3617.

A Set Benchmark Selection

To select the set benchmarks for our experimental evaluation, we used two criteria:

- **Size:** part of the intention of this research is to find the conditions in which it is better to use a non-optimal MAPFW solver, and in our case EMLA*, as opposed to an optimal solver. Generally EMLA* either performs similarly to an optimal algorithm, or worse, when instances are sufficiently small. For this reason the benchmarks we select from *mapfw.nl* must have at least 100 nodes, or at least as large as a 10x10 grid.
- **Solvable using EMLA*:** in order to facilitate comparison, it is necessary to use benchmarks which are solveable using our algorithm.

The full list of candidates is: 1, 5, 7, 9, 10, 11, 12, 19, 21, 22, 23, 24, 27, 33, 59, 61, 64.