Analysis of the influence of graph characteristics on MAPFW algorithm performance

Author: Timon Bestebreur

Supervisors: Jesse Mulderij & Mathijs de Weerdt

June 21, 2020

Abstract

The Multi-Agent Path Finding (MAPF) problem is a problem in which a route must be found for multiple agents such that they do not collide. The Multi-Agent PathFinding with Waypoints problem extends this problem by adding waypoints that the agents must visit before travelling to their end location. This paper compares five algorithms for MAPF that have been extended to incorporate waypoints. It also analyzes which influence map characteristics like corridors, chokepoints, overlapping waypoints and the average degree have on the performance of these algorithms. It concludes that EMLA and WM* perform best overall with some variations per characteristic.

Keywords — Multi-Agent Pathfinding, Waypoints, Graph Characteristics

1 Introduction

The Multi-Agent Path Finding (MAPF) problem is a problem in which multiple agents must be assigned a route from some unique starting location to some unique finish location in the least number of steps without colliding with other agents. The MAPF problem is relevant because there are numerous important applications for the MAPF problem such as robotics, self-driving cars and automation in warehouses (Stern et al., 2019).

An extension of MAPF is the Multi-Agent Path Finding with Waypoints problem (MAPFW). The main idea of waypoints is that agents incorporate intermediate locations, called waypoints, in their route. Some examples of MAPFW are scheduling a train that not only travels from start to end location but also stops at a maintenance station (Mulderij, Huisman, Tonissen, van der Linden, & de Weerdt, 2020) and packet delivery services which deliver multiple packets at different destinations while using a route that does not interfere with the route of other delivery vehicles.

In previous work algorithms for MAPF have been compared using custom benchmark instances or a standardized set of benchmark instances proposed by Stern et al. (2019). These have been used in some papers to compare new or improved algorithms to existing algorithms. While these benchmark instances represent a diverse set of test cases, they do not give insight into what influence certain characteristics have on the performance of algorithms.

The main contribution of this research is an analysis of these characteristics and an examination of the influence these have. This gives insight into which algorithm works best in which scenario and allows for a better match between problem instances and the algorithms that solve them.

This paper is organized as follows. Section 2 gives a formal description of the MAPFW problem. Section 3 discusses the different characteristics of graphs that will be examined. Section 4 gives an overview of the algorithms that exist for MAPF and the extensions that were made and discusses the expected performance on each characteristic. Section 5 describes how the results were obtained. Section 6 shows, interprets and discusses the results. Finally, Section 7 discusses the ethical aspects and reproducibility of the research and Section 8 concludes this paper and describes possible directions for future research.

2 Formal description of the MAPFW problem

This section gives a formal description of the input to MAPFW and the problem definition for MAPFW.

First the input to the problem will be defined, second the operations on this input will be described, third some conditions will be defined and finally some final definitions will be given.

Note: The terms node and vertex will be used interchangeably, and they are used to denote the same thing; an element of the set V. The input to a MAPFW problem is a tuple $\langle G = (V, E), A, S, F, X, WP \rangle$. The Graph G consists of vertices and edges. V is a set of l vertices $v_1, v_2, ..., v_l$. $E = \{(v_i, v_j) \mid v_i \neq v_j \in V\}$ is the set of edges.

Each edge is undirected, which means that an edge $(v_i, v_j) \in E$ implies another edge $(v_j, v_i) \in E$.[1] This graph is traversed by agents. $A = a_1, a_2, ..., a_k$ is a set of k Agents. These agents will serve as identifiers to indicate which start location, end location and waypoints belong to which agent. These locations are denoted by the sets S, E and F. $S = \{s_1, s_2, ..., s_k \mid i = 1, 2, .., k : s_i \in V\}$ is the set of start locations, where $s_i \in S$ is the start location for Agent a_i. $F = \{f_1, f_2, ..., f_k \mid i = 1, 2, ..., k : f_i \in V\}$ is the set of finish locations, where $f_i \in F$ is the finish location for Agent a_i. To encode how many waypoints each Agent has to visit the set X is used. $X = x_1, x_2, ..., x_k$ is the set that denotes the number of waypoints for each agent a_i s.t. for $i = 1, 2, ..., k$: agent a_i has x_i waypoints. Then there is a set of sets, denoted by WP, that contains for each agent the set of waypoints that it has to visit. $WP = W_1, W_2, ..., W_k$ s.t. for $i = 1, 2, ..., k : W_i = \{w_1, w_2, ..., w_{x_i} \mid j = 1, 2, ..., x_i : w_j \in V\}$ is the set that contains all the sets of waypoints, one set of waypoints per agent s.t. W_i is the set of waypoints for agent a_i. The path that each agent takes is denoted by P. Let $P_i = \{p_1, p_2, ..., p_{c_i} \mid i = 1, 2, ..., c_i : p_i \in V\}$ be the path for agent a_i of length c_i. The cost of P_i is defined as the length of the path.[2] A note about the labeling of the nodes in a path: even if a $wait$ action is performed the index of the element in P_i is increased. Thus, a path for agent a_i consisting of $\{s_i, s_i, f_i\}$ is labeled as $\{p_0, p_1, p_2\}$.

A path is made by an agent that performs actions. An agent has two possible actions: $wait$ and $move$. These actions both add a node to the current path of that agent. For agent a_i, initially path $P_i = \varnothing$. When the solving of the problem begins, the start location of agent a_i is added to path P_i, thus $P_i = P_i \cup \{s_i\} = \{s_i\}$. At this moment the cost c_i of $P_i = |P_i| = 1$. Then agent a_i can do two actions: $wait$ and $move$. $wait$ is defined as not moving. As a result of this, the node that was most recently added to P_i is added again: $P_i = P_i \cup \{s_i\} = \{s_i, s_i\}$. $move$ is defined as adding a node to P_i s.t. this node is not equal to the node that was most recently added to P_i. A $move$ action from node $v_i \in V$ to node $v_j \in V$ is only allowed if there is an edge $(v_i, v_j) \in E$.

A path is valid if it is complete and has no collisions. A path P_i is a complete path iff $\{s_i\} \cup \{f_i\} \cup \{w_j \mid j = 1, 2, ..., x_i : w_j \in W_i\} \subseteq P_i$. There are two types of collisions; a node collision and an edge collision.[3] A node collision occurs when two agents include the same node in their path at the same time. More formally, a node collision occurs when, for two paths P and Q, the set $C = \{p_i \Leftrightarrow q_i \mid p_i \in P, q_i \in Q, n = min(|P|, |Q|), i = 1, 2, ..., n\} \neq \varnothing$. An edge collision occurs when two agents use the same edge from opposing nodes.[4] More formally: An edge collision occurs when, for two paths P and Q, the set $C = \{p_i \Leftrightarrow q_j \wedge p_j \Leftrightarrow q_i \mid p_i, p_j \in P; q_i, q_j \in Q; n = min(|P|, |Q|); i = 1, 2, ..., n-1; j = n+1\} \neq \varnothing$.

Once an agent a_i has added its finish location f_i to its path P_i, P_i is complete and their are other agents that do not yet have a complete path the agent performs the $wait$ action until all other agents have a complete path. A path is optimal if the path is of minimal cost.

Problem Definition[5]

Name: Multi-Agent Path Finding with Waypoints (Decision problem)
Input: A tuple $\langle G = (V, E), A, S, F, X, WP \rangle$ and an integer k.
Question: Do there exist paths P_a, one for each agent $a \in A$, s.t. all paths are complete, no paths have node collisions, no paths have edge collisions and the total cost $\Sigma_{a \in A}(c_a) \leq k$?

The minimization variant of this problem is finding the minimum value for k for which the question can still be answered affirmatively.

From now on the terms graph and map will be used interchangeably to denote a graph.

3 Analysis of characteristics of Graphs

This section describes the characteristics of graphs that are examined in this paper. This research is focused on the characteristics of the *structure* of the maps and the placement of the waypoints. Thus, the size of the graphs, number of agents and number of waypoints will be chosen in such a way that they facilitate the research.

The four characteristics that are examined in this paper are the number of corridors, the number of choke-points, the number of overlapping waypoints and the average degree of the nodes in the graph. Examples of the characteristics can be seen in Figure 1.

The first characteristic is the number of corridors. A corridor is a series of 2-connected nodes, i.e. multiple nodes that form a narrow passage which only one agent can traverse at a time. Corridors are a source of edge

[1] This can easily be implemented by automatically adding the reverse edge when adding a normal edge. This way you actually have two edges but they act as an undirected edge. You do have to be careful to check for edge collisions if you do this.

[2] The cost is calculated including the first node. Thus, a path consisting of $\{s_1, v_2, v_3, f_1\}$ has cost 4.

[3] The term conflict is also used to denote a collision.

[4] When two agents use the same edge from the same side, there was a node conflict before already, so we will only define the case where agents use the edge from opposing sides.

[5] Structure and word choice of this section is adapted from the document `MAPF_problem_description.pdf` that was provided by Jesse Mulderij on May 5th, 2020.

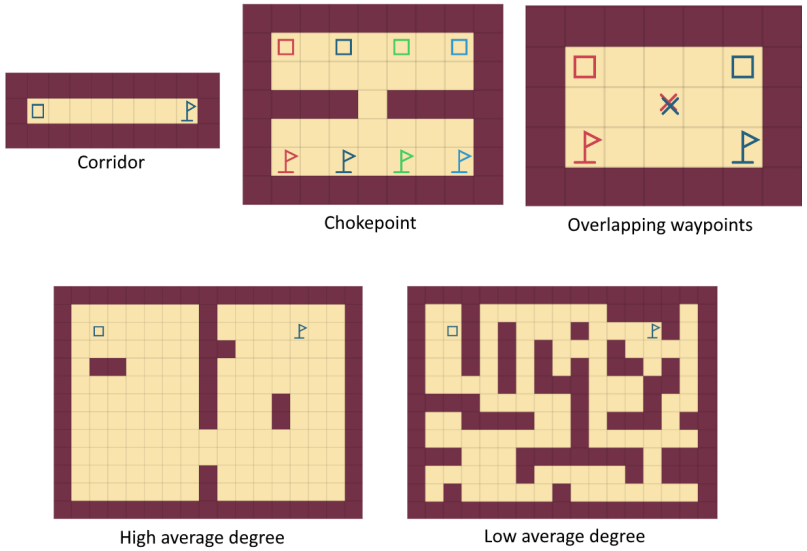


Figure 1: Examples of a corridor, a chokepoint, overlapping waypoints and maps with a high and low average degree. The X in the overlapping waypoints example represents a waypoint that needs to be visited before traversing to the end location.

conflicts. An algorithm that plans a route such that two or more agents use the corridor at the same time will have higher costs since the agents will have to wait for each other or backtrace and go around that corridor.

The second is the number of chokepoints. Chokepoints are narrow openings in a wall. A chokepoint reduces the number of agents that can pass through at the same time. Because of this, chokepoints are a source of node and edge conflicts. Chokepoints become a bigger problem with a higher number of agents, since more agents want to pass the chokepoint at the same time. Algorithms that calculate paths that traverse the chokepoint at the same time have to resolve more conflicts and will thus perform worse.

The third is the number of overlapping waypoints. Waypoints overlap when they are placed at the same node. Overlapping Waypoints are a source of node conflicts, since multiple agents need to access the same location. Thus, the performance of the algorithms in these maps comes down to their ability to avoid conflicts or resolve them efficiently.

The fourth and final characteristic is the average degree of the graph. This is defined as the average number of edges per node.[6] This number will be high on a map with big open spaces and low on a map with lots of small rooms and corridors. A decreasing average degree will cause more tight spaces and thus more chokepoints and corridors. Thus, algorithms that handle a combination of corridors and chokepoints well will perform well when the average degree becomes lower.

4 Overview of algorithms for MAPFW

Five existing MAPF algorithms have been extended to incorporate waypoints. This section briefly discusses the baseline algorithms, how these algorithms have been extended and the expected performance on each characteristic.

The first is the Conflict Based Search (CBS) algorithm, as described by Sharon, Stern, Felner, and Sturtevant (2015). CBS is a two-level algorithm that first performs a high-level search on a Conflict Tree, in which each node contains a certain number of constraints based on the conflicts between agents, and then tries to find a path for each individual agent that satisfies these constraints (Sharon et al., 2015). CBS guarantees an optimal solution. The extended version is called CBSW. The most important extensions Noah Jadoenathmisier made to CBSW is the addition of support for waypoints using a solver for Traveling Salesperson (TSP) and a heuristic that significantly improves performance on corridors of width 1 (?, ?). Because of this heuristic, CBSW will perform well on corridors. Since chokepoints are short corridors, CBSW will also perform well on chokepoints. CBSW will also perform well on maps with overlapping waypoints since CBSW is designed to handle conflicts well (Sharon et al., 2015). The maps with lower average degree will form many corridors and chokepoints. Thus, CBSW will also perform well on these maps. The weakness of CBSW lies in the fact that it requires exponential memory (Sharon et al., 2015). This can hinder performance in larger problem instances.

The second is the M* algorithm, as described by Wagner and Choset (2011). M* first plans for each robot individually and if some robots seem to have paths that are quite similar they are coupled, which minimizes the size of the search space (Wagner & Choset, 2011). M* does not guarantee an optimal solution. The extended

[6] Since edges are undirected, the number of outgoing edges equals the number of incoming edges for each node.

Algorithm	Good performance	Average performance	Bad performance	Extender
CBSW	CD, CP, OW, AD	None	None	Noah Jadoenathmisier
WM*	OW	AD	CD, CP	Jeroen van Dijk
A* + OD + ID	None	None	CD, CP, OW, AD	Stef Siekman
EMLA	OW	AD	CD, CP	Arjen Ferwerda
BCP-MAPFW	None	CD, CP, OW, AD	None	Andor Michels

Table 1: Overview of the expected performance of each algorithm and the names of the people that extended the algorithms. CD stands for Corridors, CP stands for Chokepoints, OW stands for Overlapping Waypoints and AD stands for Average Degree.

version is called WM*. The most important extension that Jeroen van Dijk made to M* is a policy which keeps track of the optimal route to each waypoint and end location. This policy guides the algorithm towards the next waypoint in an ordered list of waypoints that has been created by a subroutine which orders the waypoints using a solver for TSP (Dijk, 2020). The Strength of WM* lies in its optimistic nature which assumes the optimal route can be taken and only diverges from it when there is a collision, which results, in most cases, in a high calculation speed (Dijk, 2020). Thus, WM* will perform well on overlapping waypoints since there will be much room to manoeuvre. The downside of the optimistic approach is that instances with too many collisions cannot be solved within reasonable time (Dijk, 2020). This will result in bad performance on the chokepoints and corridors maps, since these are designed to cause many collisions. WM* will perform average on average degree since its optimistic approach will cause much backtracking due to many collisions while the optimistic approach will result in little recalculation time.

The third is the A* algorithm, with the extension of Operator Decomposition and Independence Detection (A* + OD + ID). A* has been described by Goldenberg, Felner, Stern, Sharon, and Schaeffer (2012). Independence detection groups agents that have an optimal solution that does not conflict with other agents from that group, then solves the problem for the whole group. (Standley, 2012). Operator Decomposition decomposes the task of calculating the next move of all agents from assigning a next state for all agents at the same time to assigning a next state for each agent one at a time in a fixed order. (Standley, 2010). A* + OD + ID guarantees an optimal solution. The most important extension that Stef Siekman made is the addition of conflict avoidance tables and a dynamic solver for TSP which helps ordering the waypoints (Siekman, 2020). Because of the Independence Detection A* + OD + ID is efficient when the number of agents increases while the number of conflicts stays low (Goldenberg et al., 2012). Thus, A* + OD + ID will perform bad on the overlapping waypoints maps, since these will result in many conflicts. The weakness of A* + OD + ID is that there is no clear distinction between high-level and low-level search which makes the algorithm inefficient in graphs with lots of corridors and chokepoints (Siekman, 2020). Thus, A* + OD + ID will perform bad on the chokepoints, corridors and average degree maps.

The fourth is the Multi-Label A* algorithm (MLA*), which extends the classic A* algorithm with the ability to calculate routes with multiple ordered goals and adds a heuristic that assigns tasks to agents over the complete time horizon (Grenouilleau, Hoeve, & Hooker, 2019). MLA* does not guarantee an optimal solution. The extended version is called EMLA. The most important extensions that Arjen Ferwerda made are a reservation table which allows agents to reserve nodes that are critical to their path, the addition of a $wait$ action and a dynamic waypoint choice heuristic that dynamically chooses which waypoint an agent pursues (Ferwerda, 2020). The strength of EMLA lies in the dynamic waypoint choice heuristic which makes the process of selecting the next waypoint for each agent very fast (Ferwerda, 2020). Thus, EMLA will perform well on the overlapping waypoints maps. The weakness of EMLA lies in graphs with many corridors, since these graphs require much effort to calculate a path because there are a lot of walls to walk around (Ferwerda, 2020). Thus, EMLA will perform bad on the corridors and chokepoints maps. Since waypoint selection is so efficient EMLA will perform average on the average degree maps even when its weakness is corridors.

The fifth and final algorithm is the Branch & Cut & Price algorithm (BCP), which combines elements from search-based methods and compilation-based solvers into an optimal algorithm that decomposes the problem for mathematical optimization (Lam, Le Bodic, Harabor, & Stuckey, 2019). BCP guarantees an optimal solution. The extended version is called BCP-MAPFW. The most important extensions that Andor Michels made are the support for waypoints and the efficient ordering of them using a solver for TSP (Michels, 2020). Because BCP-MAPFW is based on mathematical optimizations, its calculation speed is very high (Lam et al., 2019). This will result in average performance on all maps, since BCP-MAPFW is not particularly effective or ineffective on each map.

A note about BCP-MAPFW: BCP-MAPFW is implemented in C++ while all other algorithms are implemented in Python. Because these languages have large differences in runtime (Prechelt, 2000) (Fourment & Gillings, 2008) they cannot be directly compared. Thus, BCP-MAPFW will not be taken into account in the comparison. To get some idea of the relative performance of the implementation against the other algorithms, its results will be included in the graphs, denoted by a dotted line in the figures to make it stand out.

An overview of the information given in this section is shown in Table 1.

Item	Frequency
Number of characteristics	4
Number of maps per characteristic	5
Number of runs per map	10

Table 2: Frequency of each item in the map creation and data collection phase

Map number	Number of waypoints per agent
Map 1	2
Map 2	4
Map 3	6
Map 4	8
Map 5	10

Table 3: Number of waypoints per agent for each map that represents the overlapping waypoints characteristic

5 Method

This section describes the procedure that was followed to verify the expected performance of the algorithms. There have been two types of experiments, one increasing the influence of the characteristics (5.1) and the other increasing the number of agents (5.2). These experiments will be referred to as the Characteristics Experiment and the Progressive Experiment, respectively. For both experiments the start, end and waypoint locations were randomly selected. The algorithms ran on a linux server with a 14-core Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz per core and 8 GB ram. Runs that either exceeded 20 seconds or were unable to solve the benchmark were marked as a failure and were assigned a runtime of 20 seconds.

5.1 Characteristics Experiment

The goal of the first experiment was to keep the number of agents and waypoints the same[7] while increasing the extent to which the characteristic was present in the map. For corridors this meant that the amount and length of the corridors increased in each subsequent map. For chokepoint the amount of chokepoints increased. For overlapping waypoints the waypoints were placed such that all agents had to visit the same waypoint locations and the number of waypoints increased. For average degree the average degree decreased.

Per characteristic five graphs were designed to represent the characteristic in different intensities and varieties. This number of graphs hits the sweet spot between enough data points to cover the full range of intensities of the characteristics while not overflowing the research with data that is very similar because of the small differences. These maps were designed by hand[8] instead of with a random map generator because it is important to make sure the different characteristics are well represented in the graphs.

Every map was solved ten times. The result for each map is the average runtime, calculated over the results of these ten runs. All the frequencies and numbers are summarized in table 2 for convenient reference.

The maps that represent chokepoints, corridors and average degree contained five agents and five waypoints per agent. These numbers were chosen in such a way because five agents and five waypoints gave enough of a challenge to show differences in runtime per algorithm while still yielding reasonable runtimes on the final maps which can be very challenging to solve. It was important to find the balance between too little differences in results per map and too complicated instances that were too hard to solve. Five agents and five waypoints achieved this balance. The maps that represent overlapping waypoints contained five agents and the number of waypoints specified in table 3. The raw data is available online.[9]

5.2 Progressive Experiment

The goal of the second experiment was to keep the influence of the map characteristic constant while increasing the number of agents that have to traverse the map. This gives insight into how well the algorithms are able to handle increasing numbers of agents on a map with certain characteristics. This experiment was set up as follows;

for each of the four characteristics the third map was selected as the benchmark map, because this contains the characteristic in an average amount. For each of the four map characteristics the algorithms were given a set of fifty maps which contained a number of agents. The set of fifty maps was based on the benchmark map that

[7] This does not hold for the maps representing Overlapping Waypoints, since in these maps the structure of the map plays a small role and the main focus is on the fact that the waypoints are at the same location. Thus, in these maps the number of waypoints *does* increase.

[8] These maps were designed using the benchmark editor at `https://mapfw.nl/benchmarks/create`. Since the maps are made by hand, no guarantees can be given about general graphs that include these characteristics.

[9] The maps and data can be viewed here: `https://github.com/UltraTimon/MAPFW_research_data`

Item	Value
Number of algorithms	5
Number of characteristics	4
Benchmark map for each characteristic	Map 3
Timeout per problem	20 seconds
Number of maps per set of problems	50
Number of agents in the first problem set	2
Number of waypoints per agent	5
Number of extra agents in each subsequent problem set	1

Table 4: Overview of the numbers associated with the experiment that increases the number of agents

represented the characteristic. There were five waypoints per agent each run, since five waypoints gave a good balance between a reasonable workload and enough complexity in the routes that had to be traversed.

The challenge for the algorithm was to try to solve each individual problem within a timeout that was set at twenty seconds. This timeout gave a good balance between reasonable runtimes and allowing the algorithm to show what it is capable of. If at least one of the fifty problems was solved within the timeout, a new set of problems was given with one extra agent. This continued until none of the fifty maps could be solved within 20 seconds. Then the number of maps that could be solved was recorded for each number of agents, as well as the cost of each solution. This was used to compare runtime to cost.

The benchmark maps for Corridors, Chokepoints and Average Degree did not have any special requirement on the placement of the waypoints. The benchmark map for Overlapping Waypoints were configured such that for each location that contains a waypoint, all agents have one of their waypoints on that location. This way the waypoints overlap and all agents have to visit the same five waypoint locations. All numbers that have been mentioned in this section are listed in Table 4.

The previously described process resulted in a number that represents the number of maps that the algorithm was able to solve within the timeout for each algorithm for each characteristic for each number of agents. These results and their implications are discussed in the next section.

6 Discussion of results and evaluation of expected performance

This section discusses the results and compares them to the expected performance. The results are discussed per characteristic.

6.1 Chokepoints

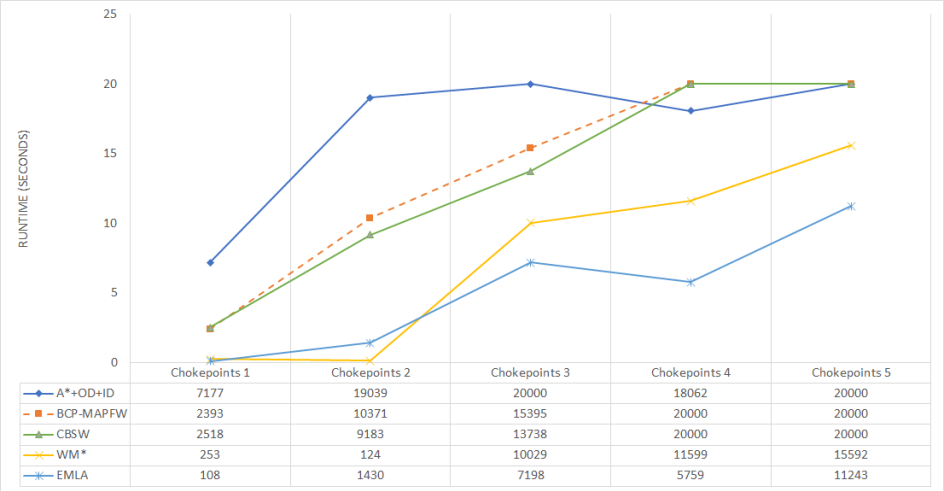
The expectation was that A* + OD + ID and EMLA would perform worst, WM* would perform average and CBSW would be the clear winner.

Both for the characteristics experiment and the progressive experiment the expectation was correct for A* + OD + ID; It performed worst. An explanation for this could be that A* + OD + ID orders the waypoints again every time step instead of once at the beginning (Siekman, 2020). This creates a lot of calculation overhead because the solver for TSP that orders the waypoints takes quite some time to calculate the best order of waypoints. If the number of agents is small this is not that much of a problem, but as the number of agents on the map grows, the number of times that the solver for TSP has to recalculate the optimal order of waypoints grows because every agent has to calculate its own order of waypoints. If the A* + OD + ID algorithm could be modified to precompute the ordering of the waypoints and cache this in memory the performance of the algorithm would probably improve significantly on maps with higher numbers of agents.

EMLA performed best overall on the progressive experiments and relatively well on the characteristics experiment in terms of runtime. The good performance on the progressive experiment is probably due to the dynamic waypoint choice heuristic which makes selecting the next waypoint for each agent very efficient (Ferwerda, 2020). This helped EMLA to keep on solving problems within reasonable time even when the amount of agents becomes quite large. For the characteristics experiment the expectation was that EMLA would perform worst together with A* + OD + ID. The motivation for this expected performance was that EMLA has difficulties handling corridors (Ferwerda, 2020) and chokepoints are very short corridors, so the expectation was that EMLA would perform bad. The results show that these difficulties are either not that severe or that the dynamic waypoint choice heuristic is so powerful that it compensates for the difficulties that corridors pose to EMLA.

Another interesting difference between the expected performance and the results is the bad performance of CBSW. Since CBSW is optimized for corridors (and thus chokepoints) with its heuristic that significantly improves performance on corridors of width 1 (?, ?) the expectation was that CBSW would shine on the chokepoint maps. However, the results show that CBSW performed average on the characteristics experiment. An explanation for this could be that the optimal solution for these maps took more time to be calculated, and once the requirement

of an optimal solution is let go the maps can be solved quicker. This is shown by the good performance of the non-optimal algorithms EMLA and WM*. CBSW performed best of all the optimal algorithms. WM* performed average, as expected. The results are shown in figures 2 and 3.



	Chokepoints 1	Chokepoints 2	Chokepoints 3	Chokepoints 4	Chokepoints 5
A*+OD+ID	7177	19039	20000	18062	20000
BCP-MAPFW	2393	10371	15395	20000	20000
CBSW	2518	9183	13738	20000	20000
WM*	253	124	10029	11599	15592
EMLA	108	1430	7198	5759	11243

Figure 2: Graph representing runtimes for the five algorithms for each map that represents the chokepoints characteristic. Each map gets increasingly complicated. Runtimes are in milliseconds, but the runtime axis is represented in seconds. Lower runtime is better.

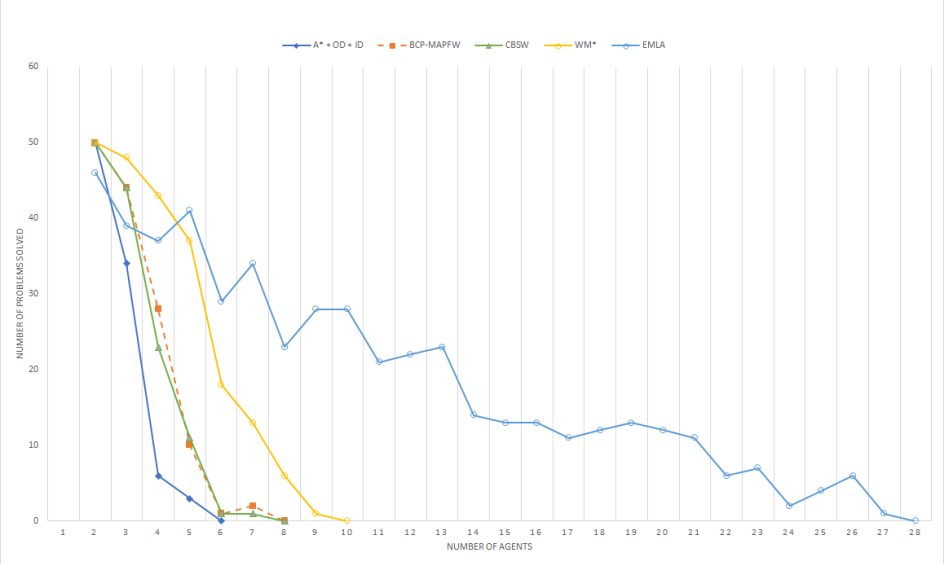


Figure 3: Graph representing the number of randomized problems the algorithms were able to solve, given a number of agents with random start and end locations and five waypoints. The third map that represents chokepoints is used as the layout for the map. Higher number of problems solved is better.

6.2 Corridors

The expectation was that CBSW would perform best, WM* would perform average and A* + OD + ID and EMLA would perform worst.

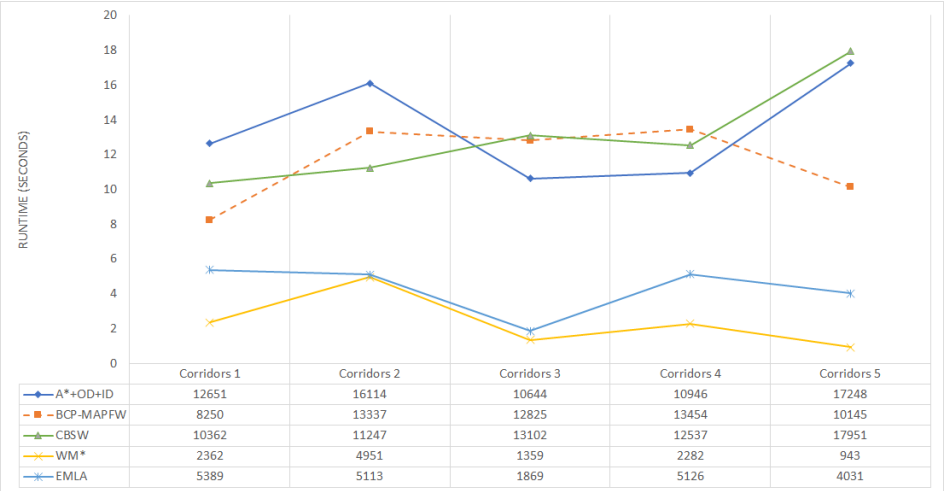
While average performance was expected from WM*, WM* actually performed quite well in terms of runtime. WM* solved each map in the characteristic experiment with the lowest average runtime and came in second in the progressive experiment. Since WM* is designed to be optimistic and assume each agent can take its optimal route it wastes little processing time during execution on checking for and resolving conflicts. If a conflict happens, it backtracks until the conflict is resolved and tries the optimal route again (Dijk, 2020). This could be the reason for the good performance on the corridors maps, which are designed to create lots of conflicts.

A* + OD + ID performed worst, as expected.

CBSW performed quite bad in this progressive experiment, just staying above A* + OD + ID. An explanation for the relatively bad performance is that CBSW requires exponential memory (Sharon et al., 2015), which is a

problem that takes its toll when the number of agents on the map grows. This high memory usage may create processing time overhead when there are more agents to keep track of, thus making it less likely that the problem is solved within 20 seconds. The performance of CBSW on maps with more agents could probably be improved significantly if the amount of memory that is used during execution could be reduced.

The results are visible in Figures 4 and 5.



	Corridors 1	Corridors 2	Corridors 3	Corridors 4	Corridors 5
A*+OD+ID	12651	16114	10644	10946	17248
BCP-MAPFW	8250	13337	12825	13454	10145
CBSW	10362	11247	13102	12537	17951
WM*	2362	4951	1359	2282	943
EMLA	5389	5113	1869	5126	4031

Figure 4: Graph representing runtimes for the five algorithms for each map that represents the corridors characteristic. Each map gets increasingly complicated. Runtimes are in milliseconds, but the runtime axis is represented in seconds. Lower runtime is better.

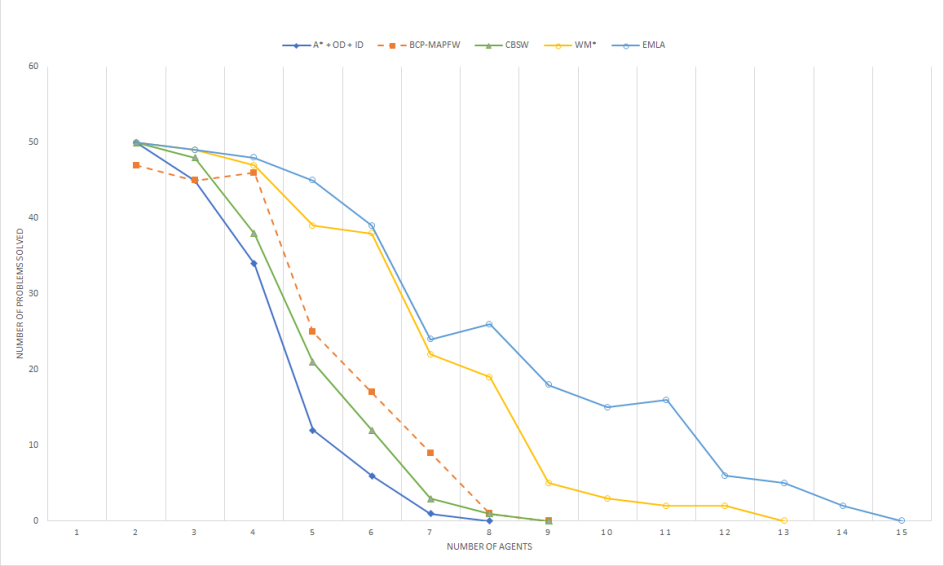


Figure 5: Graph representing the number of randomized problems the algorithms were able to solve, given a number of agents with random start and end locations and five waypoints. The third map that represents corridors is used as the layout for the map. Higher number of problems solved is better.

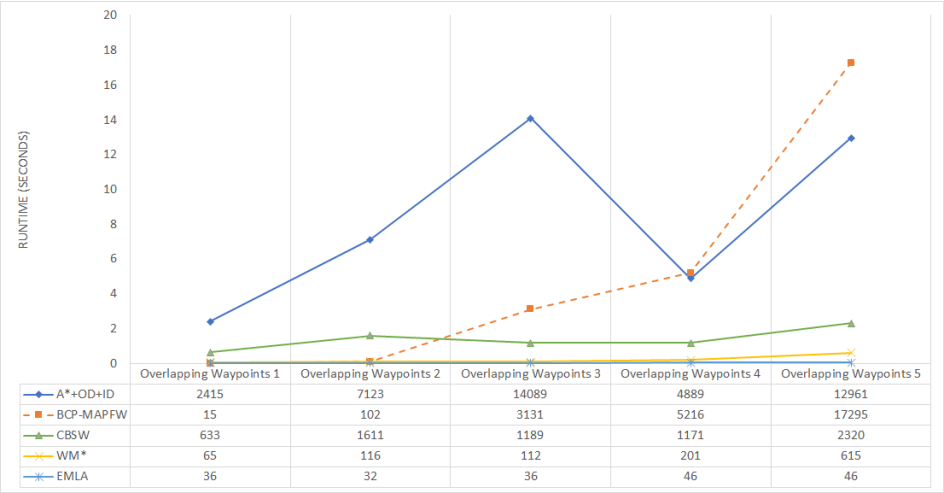
6.3 Overlapping waypoints

The expectation was that CBSW, EMLA and WM* would perform well and A* + OD + ID would perform worse.

A* + OD + ID did perform quite bad on both the characteristic and the progressive experiment. CBSW performed quite bad as well on both experiments. EMLA performed relatively well on the characteristics experiment and exceptionally well on the progressive experiment. The performance of these algorithms corresponds to the previously given explanations. The results are visible in Figures 6 and 7.

Map number	Average Degree
1	3.51
2	3.09
3	2.25
4	1.30
5	0.71

Table 5: Overview of the average degree of each map that represents the average degree characteristic.



	Overlapping Waypoints 1	Overlapping Waypoints 2	Overlapping Waypoints 3	Overlapping Waypoints 4	Overlapping Waypoints 5
A*+OD+ID	2415	7123	14089	4889	12961
BCP-MAPFW	15	102	3131	5216	17295
CBSW	633	1611	1189	1171	2320
WM*	65	116	112	201	615
EMLA	36	32	36	46	46

Figure 6: Graph representing runtimes for the five algorithms for each map that represents the overlapping waypoints characteristic. Each map gets increasingly complicated. Runtimes are in milliseconds, but the runtime axis is represented in seconds. Lower runtime is better.

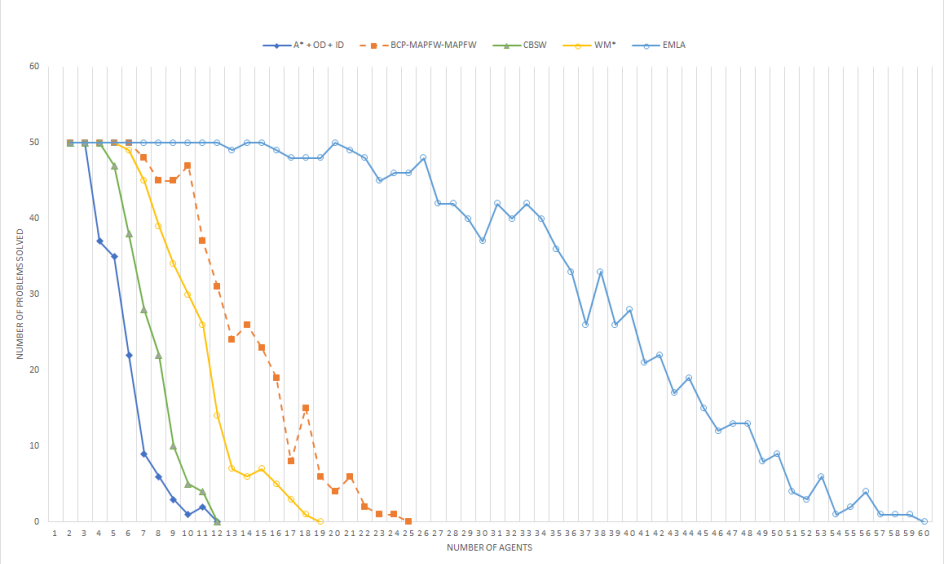


Figure 7: Graph representing the number of randomized problems the algorithms were able to solve, given a number of agents with random start and end locations and five waypoints. The third map that represents overlapping waypoints is used as the layout for the map. Higher number of problems solved is better.

6.4 Average degree

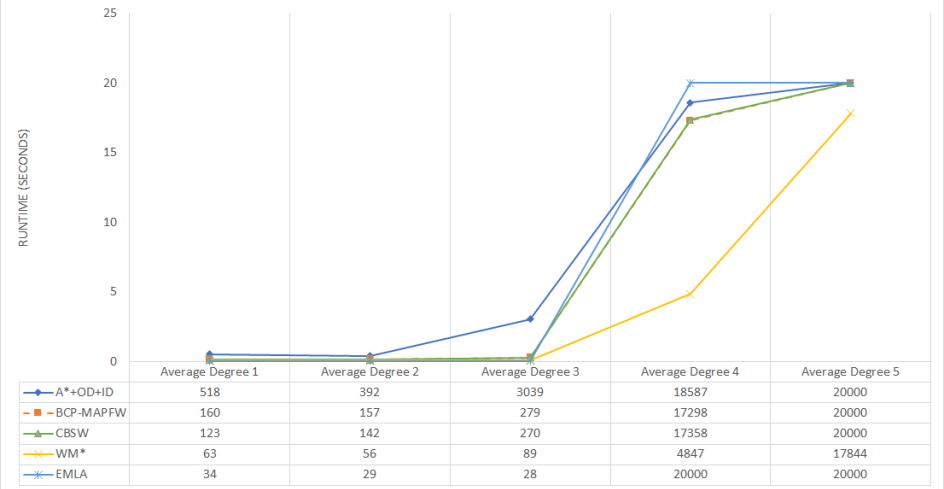
The expectation stated that all algorithms would perform well in the first few maps. Furthermore, CBSW would perform best in the later maps because of its corridor heuristic and WM* would keep on performing quite well in the later maps as well. A* + OD + ID would slack behind.

The runtimes for the characteristic experiment are quite close. The most interesting feature of the data is the significant bump in runtime between map 3 and map 4. To get a more accurate comparison of the algorithms and

how well they compare with an increasing average degree, maps that have an average degree between the values of map 3 (2.25) and map 4 (1.30) would be good test cases. An overview of the average degree of each map for the average degree characteristic is given in Table 5.

In the progressive experiment the expectation is quite off for CBSW and especially EMLA, which performed quite bad and quite good respectively. An explanation for the performance of both these algorithms has been given earlier.

The results are visible in Figures 8 and 9.



	Average Degree 1	Average Degree 2	Average Degree 3	Average Degree 4	Average Degree 5
A*+OD+ID	518	392	3039	18587	20000
BCP-MAPFW	160	157	279	17298	20000
CBSW	123	142	270	17358	20000
WM*	63	56	89	4847	17844
EMLA	34	29	28	20000	20000

Figure 8: Graph representing runtimes for the five algorithms for each map that represents the average degree characteristic. Each map gets increasingly complicated. Runtimes are in milliseconds, but the runtime axis is represented in seconds. Lower runtime is better.

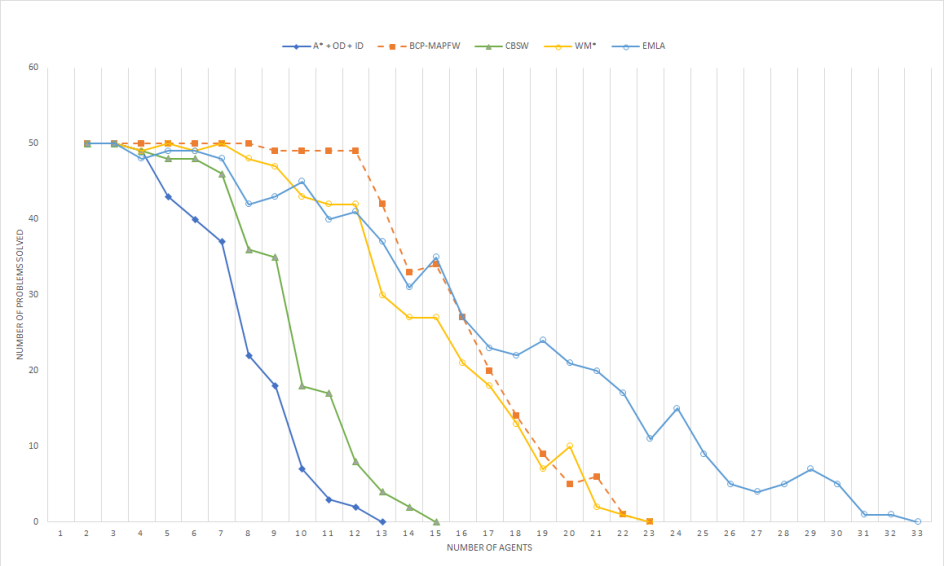


Figure 9: Graph representing the number of randomized problems the algorithms were able to solve, given a number of agents with random start and end locations and five waypoints. The third map that represents average degree is used as the layout for the map. Higher number of problems solved is better.

6.5 Analysis of the cost overhead of the non-optimal algorithms

WM* and EMLA have no guarantee of optimality which can result in higher costs than the optimal solutions. Overall these algorithms perform better in terms of runtime than the optimal algorithms. This section compares the cost of the solutions of WM* and EMLA from the progressive experiment and compares them to the cost of the optimal solution to find out at what cost the faster runtimes were achieved. The results show that WM* on average manages to stay within 1% extra costs on top of the optimal cost, while EMLA produces solutions with on average 19% overhead. Thus, EMLA has good runtimes but pays for it with a high overhead on costs, while WM* manages to achieve relatively good runtimes for relatively little overhead.

6.6 Comparison of the runtime of C++ and Python

Previous work has shown that C++ is about 8-20 times faster than Python in problems in bioinformatics problems (Fourment & Gillings, 2008) and about two times faster in a problem where telephone numbers had to be converted into word strings (Prechelt, 2000).

To get some insight into the difference in runtime between C++ and Python in the MAPFW problem WM* was implemented both in C++ and Python by Jeroen van Dijk. WM* then solved most of the benchmarks[10] at `https://mapfw.nl/benchmarks/` in both the C++ and Python version. Comparing the runtimes of the C++ and the Python implementation shows that the C++ implementation was on average 2.25 times faster than the Python implementation. This shows that the C++ algorithm BCP-MAPFW cannot be directly compared to the other algorithms that were implemented in Python. For the characteristics and progressive experiment the Python implementation of WM* was used.

7 Responsible Research

This section reflects on the ethical aspects of the research and discusses the reproducibility of the methods that were used.

This research has very little ethical implications, since it compares computer algorithms which operate on mathematical structures. This research contributes insights which help the improvement of train planning, which is also ethically sound. There have been conducted no human interviews, human or animal tests and no user data has been collected. Thus, this research is very ethically responsible.

The main issue with this research is that the maps were created by hand, which makes them prone to design flaws and the possibility of creating maps that inadvertently favored one algorithm over the other. Hand creation also decreases the reproducibility of the research, since the maps were not created according to some rigid method. There are two reasons why, with this in mind, the maps were still designed by hand.

First, for the maps representing corridors and chokepoints it was important for the research that these elements would be part of the map. Due to time limitations it was not feasible to develop a random map generator that still incorporates these design elements.

Second, for the maps representing average degree the focus of the research is on the effect of the average degree on the performance, which is exercised by the increased number of conflicts the agents have in the increasingly narrow spaces. With random map generation there is a high chance of creating small rooms in which an agent is stuck on its own, which would result in a trivial path for that agent because there are little conflicts. To make sure that there are no small rooms the maps were designed by hand.

To offset the fact that the maps were made by hand the locations of the start, end and waypoint for each agent and each run were randomly chosen and the results are the average of multiple runs. This way the influence of certain design choices is reduced.

8 Conclusions and Future Work

This paper compares five algorithms for MAPF which have been extended to incorporate waypoints and analyzes what influence certain characteristics of graphs have on the runtime of these algorithms.

Chokepoints, narrow openings in a wall which permit only one agent through at a time, cause conflicts between multiple agents who want to pass the chokepoint at the same time. WM* and especially EMLA handle increasing numbers of chokepoints relatively well while other algorithms have more difficulty.

Corridors, long narrow passages which permit only one agent at a time, cause collisions and require long paths when occupied corridors have to be circumvented. Corridors do not pose much of a challenge for WM* and EMLA, while A* + OD + ID and CBSW have a hard time with it.

Overlapping waypoints cause conflicts between multiple agents which need to access the same locations but cannot do that simultaneously. EMLA is able to handle maps with more than 50 agents and CBSW was able to stay close to the good performance of EMLA and WM*.

The average degree determines the average number of edges per node. A low average degree causes many collisions between agents because of the many narrow passages and chokepoints. Results of the algorithms were quite similar, however EMLA did show that it is capable of handling many agents at a time.

While EMLA and WM* did perform well in terms of runtime, they do not guarantee optimal solutions in terms of cost. The average costs over all runs in the progressive experiment shows that on average EMLA has a cost overhead of 19% while WM* has a cost overhead of 1%.

A comparison of the runtime of a C++ and a Python implementation of the WM* algorithm shows that the C++ implementation is on average 2.5 times faster than the Python implementation, which shows that algorithms that are implemented in C++ and Python cannot be directly compared.

[10] Benchmarks 21, 54 and 58 were not solved because WM* was unable to solve them.

Future research could look into creating (pseudo)random map generators which incorporate map characteristics in such a way that the characteristics do not get lost in the map generation. Furthermore, research could be performed on the subject of the performance of MAPFW algorithms on maps with average degrees that fall between 2.25 and 1.30. This could result in a more accurate analysis of the influence of a decreasing average degree on algorithm performance.

9 Acknowledgements

I would like to express my thanks to Noah Jadoenathmisier and Stef Siekman for creating the website `https://mapfw.nl` which allowed me and my fellow students to quickly compare algorithm performance.

Jesse Mulderij and Mathijs de Weerdt also deserve my thanks. I am grateful for their help and good guidance during this research. Thank you!

References

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

Ferwerda, A. (2020, June). Extending the Multi-Label A* Algorithm for Multi-Agent Pathfinding with Multiple Waypoints. *TU Delft Repository*.

Fourment, M., & Gillings, M. (2008, February). A comparison of common programming languages used in bioinformatics. *BMC bioinformatics*, *9*, 82. doi: 10.1186/1471-2105-9-82

Goldenberg, M., Felner, A., Stern, R., Sharon, G., & Schaeffer, J. (2012, December). A* Variants for Optimal Multi-Agent Pathfinding. , 7.

Grenouilleau, F., Hoeve, W.-J. v., & Hooker, J. N. (2019, July). A Multi-Label A* Algorithm for Multi-Agent Pathfinding. *Proceedings of the International Conference on Automated Planning and Scheduling*, *29*, 181–185. Retrieved 2020-04-23, from `https://www.aaai.org/ojs/index.php/ICAPS/article/view/3474`

Lam, E., Le Bodic, P., Harabor, D. D., & Stuckey, P. J. (2019, August). Branch-and-Cut-and-Price for Multi-Agent Pathfinding. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence* (pp. 1289–1296). Macao, China: International Joint Conferences on Artificial Intelligence Organization. Retrieved 2020-04-23, from `https://www.ijcai.org/proceedings/2019/179` doi: 10.24963/ijcai.2019/179

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

Mulderij, J., Huisman, B., Tonissen, D., van der Linden, K., & de Weerdt, M. (2020, June). Train Unit Shunting and Servicing: a Real-Life Application of Multi-Agent Path Finding. *arXiv:2006.10422 [cs]*. Retrieved 2020-06-21, from `http://arxiv.org/abs/2006.10422` (arXiv: 2006.10422)

Prechelt, L. (2000, October). An empirical comparison of seven programming languages. *Computer*, *33*(10), 23–29. (Conference Name: Computer) doi: 10.1109/2.876288

Sharon, G., Stern, R., Felner, A., & Sturtevant, N. R. (2015, February). Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, *219*, 40–66. Retrieved 2020-04-23, from `http://www.sciencedirect.com/science/article/pii/S0004370214001386` doi: 10.1016/j.artint.2014.11.006

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

Standley, T. S. (2010, July). Finding Optimal Solutions to Cooperative Pathfinding Problems. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*. Retrieved 2020-05-18, from `https://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1926`

Standley, T. S. (2012, July). Independence Detection for Multi-Agent Pathfinding Problems. In *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*. Retrieved 2020-05-18, from `https://www.aaai.org/ocs/index.php/WS/AAAIW12/paper/view/5230`

Stern, R., Sturtevant, N. R., Felner, A., Koenig, S., Ma, H., Walker, T. T., ... Boyarski, E. (2019, July). Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Twelfth Annual Symposium on Combinatorial Search*. Retrieved 2020-04-23, from `https://www.aaai.org/ocs/index.php/SOCS/SOCS19/paper/view/18341`

Wagner, G., & Choset, H. (2011, November). M*: A Complete Multirobot Path Planning Algorithm with Performance Bounds. , 8.