A Guide to Solving Pathfinding Problems with Multiple Agents

Bram van Kooten, Victor Ionescu, Jasper Teunissen, Mike van der Meer and Gijs Paardekooper

***Abstract*—Currently, literature regarding Multi-agent Path Finding (MAPF) does not give a broad enough overview of all the different approaches. Many papers are hard to read and require proper knowledge of MAPF. The goal of this report is to give a global overview of MAPF. To achieve this goal, we provide a detailed explanation of what MAPF problems look like, as well as giving a clear overview of the strength and weaknesses of different solutions. Besides this theoretical analysis, we also analyse and critique benchmarking performed by other researchers. Following all this, we conclude that the field of MAPF lacks agreement on terminology. Furthermore, performance analysis is limited to researchers choice, skewing research in their own favour.**

Keywords: Multi-agent Algorithm, Path Finding, MAPF, Overview, Performance analysis

I. Introduction

The world is more connected than ever before and getting increasingly connected by the day. With that, problems in logistics with regards to airplanes, ships, trains and other means of transport and travel are becoming more and more difficult. With this increased amount of traffic there is also an increased chance of collisions between the paths, e.g. between the paths of two planes, which could have disastrous consequences. So how do we make sure that there are no collisions? Besides that, how do we make sure everything arrives on time? This problem can be modelled as a Multi-Agent Path Finding (MAPF) problem.

MAPF is a type of multi-agent planning problem in which the task is to plan paths for multiple agents. The agents are substitutes for real world entities (trains, planes etc). While it is easy enough to create a path for each agent individually, problems arise when we have to factor in the interaction with other agents. In general MAPF problems, two agents are not allowed to collide with each other. This means that besides finding the optimal path on an individual level, the agents also have to make sure there are no conflicts with other agents. (Stern et al., 2019)

With this report we aim to give a global overview of MAPF and illustrate where it might be a suitable solution for a problem. We believe that the current literature does not give a broad enough overview of the different approaches which have been developed so far. Many papers are difficult to read on their own, and require the reader to first brush up on their knowledge of MAPF in other places. In the same trend, many papers only discuss their own developed algorithm, and compare it with older, but still similar, types of algorithms. This paper aims to give a more top-down approach into the different classes of MAPF algorithms. This allows the reader to make an educated choice whether MAPF is suitable for their problem in the first place and which type of algorithm would best suit their needs.

The first important step in solving any real-world problem involves finding solutions that already exist for similar problems, to prevent "re-inventing the wheel". Consequently, it is important to describe exactly what MAPF problems look like. We also show some example problems where MAPF has already been applied, and how the two problems were related. This is done in chapter II. If MAPF turns out to a problem that suits your needs, the next step would be to determine which solutions would be a good fit (e.g. speed vs efficiency). Many different solutions have been proposed in the literature, but there is not yet a clear overview of their differences in strengths and weaknesses. We attempt to provide such an overview in chapter III. Such an overview is also beneficial for engineers who are not satisfied with the performance of their current implementation. However, theoretical analysis is just that, theoretical. The actual performance of a solution may, in reality, differ from the predictions. It is therefore important to also perform empirical analysis. In chapter IV we provide a summary of the most relevant performance analysis in the literature so far, as well as recommend methods to create your own analysis. Finally, we summarise all of our findings and important knowledge that we have gained while working on the paper in the conclusion.

II. General MAPF Instance

Multi-agent path-finding is not a stand-alone problem, but is designed to model real-world problems into something more simplistic and well-defined. Before we discuss different algorithms and their performances, we must first look at what a MAPF instance may look like. This is because we need to establish basic knowledge about the entities and objectives that are present in this problem. To prevent re-explaining these concepts later on, we define terms that describe these concepts which we will use in the remainder of this paper. The terms and concepts are explained in section II-A. In order to also visualise these

All authors are with the institute Delft University of Technology (TU Delft)

concepts, in section II-B, we describe a few real-world cases where MAPF algorithms have already been applied. Afterwards, we provide a formal definition in terms of graphs and sets in section II-C. We will explain key terms that are used to describe the inner workings of MAPF algorithms. Finally, we look at a a number of variants that have been developed. These variants indicate how the basic model can be extended to better represent the original problem.

A. Informal description

In order to determine the applicability of MAPF to a problem, we need to look how a problem can be reduced to an instance of MAPF. An instance of MAPF consists of a number of agents, a domain/environment in which the agents can move. Agents move through the environment to reach their goal locations.

1) Environment

Many MAPF problems are based on movement of real entities, such as robots, vehicles, or simply humans. To reduce such a problem to a MAPF instance, the real, physical world must somehow be represented by a finite amount of data. This means that possible locations in the real world must be mapped to some digital data structure. A common approach is to use a grid. A grid is a surface (2D) or volume (3D) divided into a fixed number of equally sized cells (squares or cubes). Each cell represents a small part of the original physical area. See figure 1 for an example of a grid with obstructions. Making the cells bigger results in less computation time, but also less precision, and vice-versa. If the agents are assumed to position themselves in the centre of the cells, larger cells could be used to enforce a certain distance constraint between the agents.

Some real-world applications exist for which grids are not a good idea. These are applications where the pathways are already discrete by construction (e.g. motorways, train tracks). This could more easily be done in a (finite) graph. A graph consists of a number of nodes, which represent locations, and a number of edges connecting any 2 nodes, representing a pathway. Since this is all that is required to define a movement (a start point, an endpoint, and a pathway), any domain that allows movement can be reduced to a graph. One algorithm that uses this and was inspired by road networks is FAR, which is discussed in the next chapter.

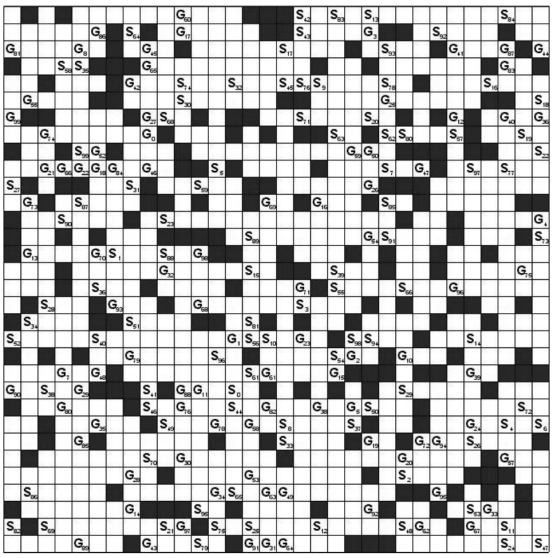


Fig. 1: *Example from (Silver, 2005) of a grid domain. Agent start locations are marked with S_i, goal locations are marked with G_i. Black squares represent obstructions through which the agents cannot move.*

2) Actions

Agents will attempt to reach their goal positions by performing one of two actions: move or wait. The wait action is a natural consequence of the no-collisions constraint, since an agent has to let another agent pass first if they are set to collide. The wait action can also be used to prevent agents blocking other agents, see figure 2.

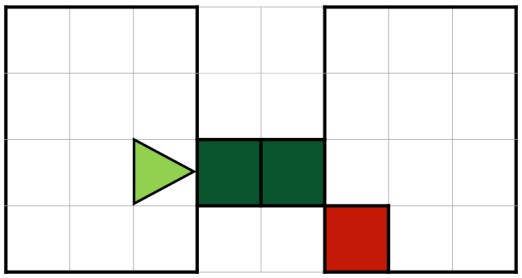


Fig. 2: *Example from (Sigurdson et al., 2018) of grid containing a one-cell-wide corridor. The green triangular agent on the left wants to move to the red square, but is blocked by the green square agents that have already reached their goals. The other 2 agents must therefore wait before they reach the corridor so the triangle can first reach its goal state.*

3) Objectives

As mentioned, the main goal is for the agents to reach their target location at some point in time. In many real-world problems, there exist some constraints on the amount of time available, or the amount of energy available (e.g. robots running on batteries, vehicles running on fuel). Minimising the amount of time until all agents reach their goals is called minimising the makespan (Stern

et al., 2019). Minimising the total amount of energy used can be equated to minimising the total length of all paths combined, or sum-of-costs for short. See figure 3 for an example of makespan and sum-of-costs.

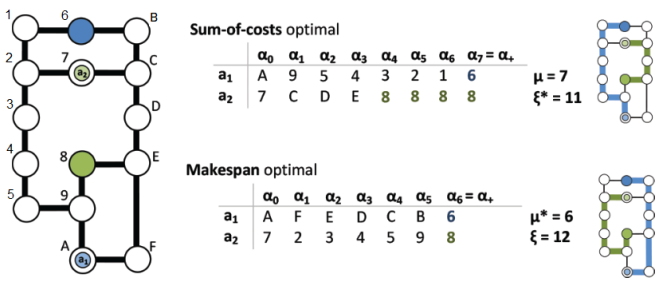


Fig. 3: *Example from (Surynek et al., 2016) of an instance where optimizing makespan and sum-of-costs yields different solutions. μ denotes the makespan, ξ denotes the sum-of-costs. In the top-right solution, ξ is optimised. In the bottom-right solution, μ is optimised*

B. Example Applications

Now that we have an understanding of the different components in a MAPF instance, we will visualise them by using real-world examples. This should illustrate in which cases it can be of use. On top of that it shows the benefits of MAPF when applied correctly. The first example we discuss are video games. In video games, players only constitute a very small percentage of the objects because there are a lot of non player characters (NPCs) walking around. The paths of these NPCs are determined by an MAPF algorithm. Game designers that want to offer their players a smooth gaming experience will benefit from selecting a suitable MAPF algorithm. The second example takes place in warehouses. Because there is a growing amount of shopping taking place, warehouses need to become bigger as well as more efficient. This process can be streamlined with the use of robots. However, these robots need to know where to go without colliding into each other. In the third and final example we discuss the problem of logistics in a harbour. Ships travel between harbours all over the world where they need a place dock. Which ship has to dock where and how the routes to their docking locations are decided is determined by an MAPF algorithm. MAPF could be used to minimise the makespan which could in turn result in less working hours for the personnel. In the following three subsections we will relate the concepts that we have learned in the previous section (e.g. agents, environment) to the real-world entities in the given example.

1) Video game NPCs

Video games often have a lot of objects moving around at the same time. An example of this is (Sigurdson et al., 2018), where multiple algorithms are tested for the game Dragon Age: Origins (DAO), a 3D RPG. The map contains some characters that the player can control, some buildings, walls, and a bunch of computer-controlled characters (NPCs). At any point in time, the player is able to assign or alter goal locations of his characters. These characters cannot move through each other, or through walls. A MAPF instance models these characters as agents. The map is represented as a 8-connected grid (the agents can move in 8 directions). Goal locations are set by the user, and can be changed at any moment. In order to account for this sporadic re-planning, the chosen MAPF algorithm must trade in optimality for speed. More on this trade-off is explained in chapter III.

2) Autonomous Warehouse Robots

With the ever growing amount of online shopping that we do nowadays, distribution centers also have to get bigger and more efficient. One way warehouse processes have been streamlined is with the use of Autonomous Robots. Normally employees have to fetch a product themselves from a certain location if it is required (e.g. for delivery), which takes time. A system called Kiva uses autonomous robots that ride around to fetch these products (Wurman et al., 2008). The storage shelves in this case can be attached to the moving robots. The shelves that hold one of the ordered items are then transported to a central point for collection. In this case the robots are the agents, and the goal location is the central point for collection. The warehouse itself is a grid where robots move around upon. MAPF is used to create the paths of the various robots and to make sure no collisions occur. Using Kiva for a warehouse as is described by Wurman et al. greatly increases efficiency and reduces the expenses on personnel.

3) Vessel traffic

As was described in the introduction, the world is more connected than ever before. Trade between a lot of countries in the world is flourishing and a lot of goods are exchanged. To transport all these products over the world, a large number of ships are needed. All of these ships have to dock in a harbour to load or unload goods, and they continuously get assigned new target locations. These ships may come from many different places and have different destination points. They should, however, never end up in the same place or along the same path, since that would result in a collision. MAPF algorithms plan the routes of these ships in such a way that this will never happen (Teng et al., 2017). In this case the ships are the agents and the harbour is represented by a grid. The goal locations are the places where a ship can dock to load or unload their goods. A difficulty with ships is that unlike cars, ships cannot use brakes and stand still in a few moments. Surprisingly enough, a lot of accidents happen in harbours around the world. MAPF algorithms have been proven to be effective at reducing those collisions in harbours (Kim et al., 2014).

C. Formal definition

In this section, we will define an MAPF instance in more formal notation. We believe that explaining the inner workings in different ways improves understanding. This notation is commonly used in the literature on this subject, and will therefore also prove useful when performing your own research.

The environment is represented by an undirected graph $G = (V, E)$, where $V = \{v_1, v_2, ..., v_n\}$ is a finite set of vertices and $E = \{(v_i, v_j) | v_i, v_j \in V\}$ is a finite set of edges. Grids can be represented as graphs by constructing a vertex for each cell. In case of a 4-connected grid, edges are constructed for left, right, up, and down movements. In case of a 8-connected grid, edges for diagonal movement are also constructed.

Agents can be represented by a single set $A = \{a_1, a_2, ..., a_m\}$ and two functions a_s and a_g, where $a_s(a_i)$ returns the initial vertex of agent a_i, and $a_g(a_i)$ returns the goal vertex of agent a_i.

For every agent a_i, a movement plan π_i is calculated. An agent's movement plan is a an ordered list of positions which stores the location of the agent at every time step. We assume time is discretised, and that every agent performs one action (move or wait) in each time step. Let $\pi_i(j)$ denote the location of agent a_i at time step j. A feasible solution of the MAPF problem is then the combination of all different plans, $\pi = \bigcup_{a_i \in A} \pi_i$. The solution is valid when the following constraints are satisfied:

- Each individual plan is valid. That means that for every two succeeding time steps, the agent moves along an edge $e \in E$, or remains in the same position. Formally, if $\pi_i(j) = v$ and $\pi_i(j+1) = u$, then either $(u, v) \in E$ (move) or $u = v$ (wait).
- No collisions must exist between any two agents. Multiple types of collisions are possible, although some of them can be relaxed depending on the situation (see section II-D). The most common (Stern et al., 2019) types of collisions are:
  - **Vertex conflict.** This conflict occurs when more than 1 agent is present in a vertex at the same time.
  - **Edge conflict.** This conflict occurs when more than 1 agent moves along the same edge at the same time. Note that an edge conflict in the same direction is impossible if vertex conflicts are already avoided. We then only talk about edge conflicts where agents swap places.
  - **Following conflict.** This conflict occurs when an agent moves to a vertex at the same time that another agent leaves that vertex. A following conflict involving more than 2 agents is called a **cycle conflict**.

D. Variants

In section II-A we have described the basic instance of a MAPF problem. This instance can be extended to better fit the original problem, which can differ from the basic instance in several ways. We call these variants of the basic instance.

One variant introduces teamwork to solve goals that can be shared between multiple agents. A good example is the Package-Exchange Robot-Routing Problem (PERR) (Ma, Tovey, et al., 2016). In this problem, packages must be picked up and delivered at some destination. It does not matter which agent delivers the package, only that it gets delivered. In this particular example, an additional action is introduced to allow agents to exchange packages between agents.

Other variants are more concerned with quantifying the actions of the agents. This is because in reality, agents may require a different amount of time per action. For example, non-holonomic robots need to turn before they can change direction, and will take longer to reach the next location than if the robot would move in a straight line (Ma et al., 2017; Cirillo et al., 2014). Agents could also be of different size, which can be modelled by allowing some agents to occupy more than 1 vertex (Li et al., 2019).

Some variants have been designed to better suit the restricted movement of certain vehicles. Vehicles such as automobiles, aircraft and vessels cannot instantly accelerate/decelerate (aircrafts are also not allowed to stop mid-air). Neither can they turn 90 or 180 degrees in a single time step. The model in this variant describes the movement of an agent in terms of speed and angle (Pallottino et al., 2007).

The movements can also be made to be influenced by past movements (Jansen and Sturtevant, 2008) or by user-defined "high-ways" (Ma, Koenig, et al., 2016), to make the agents their movements more predictable in an environment where humans also work.
We will not further discuss these variants in later chapters, as they are too specific to allow a decent comparison to be made.

III. Algorithm Selection

Multi-agent path finding (MAPF) algorithms are used for different types of applications like controlling character movement in video games, controlling traffic flow or routing of planes in an aviation context (Felner et al., 2017). Because of this variety in MAPF algorithms, it is important to be able to compare different MAPF algorithms and make comparisons between those algorithms based on aspects that differentiate them from other algorithms. The aim of this chapter is to give the reader a better understanding of the way a MAPF algorithm can be selected. Each setting discusses an important

aspect of MAPF algorithms and explains what benefits and drawbacks a specific type of that aspect has. In the first section III-A we discuss the difference between a centralised and a distributed setting. We discuss whether the agents are in control of their own actions or whether there is some centralised system in control of the agents and their movements. In section III-B we differentiate between a coupled and decoupled solution approach used by MAPF algorithms. The solution approach used by a MAPF algorithm is closely related to the dependency between agents in the problem environment. Section III-C then differentiates algorithms that produce an optimal and a sub-optimal solution to a given problem. In some cases an optimal solution is not necessary and a trade-off is made between speed versus accuracy of the solution. We achieve this by discussing a few example algorithms. Finally all our findings are concluded in section III-D where we have a table with the various algorithms discussed and the different aspects the algorithms do or do not possess.

A. Controlling Agents: Setting

The way MAPF algorithms control agents in the environment of a given problem is called the setting. MAPF algorithms can be classified into two settings; centralised settings and distributed settings. Centralised settings are used when their is one computer in the real-world that control all agents while a distributed setting is used when each agent in the real world has its own computer. What setting to chose is important to consider since the two settings require different ways of implementation in the real world. A distributed setting is sometimes called a decentralised setting. To make the distinction between the two settings clear this paper uses the term distributed setting. This section discusses what both settings mean and when they should be used.

1) Centralised Setting

MAPF algorithms that use a centralised setting have one computational unit that makes decisions for the agents in the environment (Felner et al., 2017). Agents do not think for themselves, but rather receive simple instructions on what action to perform. A centralised setting is applicable to environments where the agents are fully or partially dependent. Centralised path finding is often used in combination with a coupled solution approach (see section III-B1). Centralised path finding algorithms are often optimal. Optimality is thoroughly discussed in section III-C.

2) Distributed Setting

MAPF algorithms that use a distributed setting have agents in the environment that all have their own computer for making decisions (Felner et al., 2017). A distributed setting is often used for problems where the agents are independent or almost independent of each other. Although the agents in a distributed setting make decisions on their own, collisions are still prevented. For example, Windowed Hierarchical Cooperative A* (WHCA*) reserves paths found for individual agents in a reservation table that is shared among all agents in the environment (Bnaya and Felner, 2014).

B. Solution Approach: Coupled & Decoupled

The process of finding feasible paths for all agents can be done in either a coupled, or a decoupled approach. The choice between these two options has an impact on the run-time of the algorithm and the quality of the solution. These approaches are not mutually exclusive and can be combined to get the best of both worlds.

1) Coupled Approach

In a coupled approach, the solutions of all agents depend on each other. This means that the entire search-space must be traversed to find a feasible solution for all agents. Due to this, the cost of finding a solution grows exponentially with the number of agents. On the other hand, it is well suited for finding optimal solutions. More on optimality can be found in section III-C. This dependency on each other requires the algorithm to store information on all agents. It is therefore best combined with a centralised approach.

2) Decoupled Approach

In a decoupled approach, each agent finds an individual (optimal) path for reaching its goal. The solutions these agents find are either fully or partially independent of each other. This means that conflicts between the paths are possible. There are several ways to resolve these conflicts. One way is by preventing collisions in the first place, by using reservation tables. Reservation tables are explained in III-C5b. Another way is to search for and resolve conflicts after the fact. For example, FAR performs such re-planning on-the-fly using predefined rules.

3) Combination of Coupled and Decoupled

The high computational cost of using a coupled approach leads to the approach being unsuitable for usage in environments with many agents. A decoupled approach is prone to suboptimality and deadlocks. A nice middle-ground can be found by combining both approaches. Conflict-Based Search is an example of an algorithm that uses a combination of the coupled and decoupled approaches and will be discussed in further detail in section III-C4b.

Subdimensional expansion is an example method that combines the coupled and decoupled approach (Wagner and Choset, 2015). Subdimensional expansion itself is not an algorithm but rather a method that can be used by MAPF algorithms for finding a solution. Subdimensional expansion initially uses a decoupled approach to find individual paths in a low-dimensional search-space. If a collision is found, a small part of the search-space around the collision is increased in dimension to calculate alternate paths. This is done in a coupled manner.

C. Quality of the Solution: Optimality

In most cases it takes significantly longer to find the best possible solution compared to finding a solution that approaches the best solution. Therefore there is a trade-off between speed and accuracy of the solution between sub-optimal and optimal solutions (Stern et al., 2019). Before we move on to the question of optimality we first need to explain the term completeness, which is done in the next subsection. On top of that it is important to determine what the algorithms we discuss in the next chapter are based on. We explain what it means to be a reduction based algorithm or an A* based algorithm in the subsections below.

1) Completeness

When a path planning algorithm is guaranteed to either find a path or to determine that no path exists it is deemed to be complete (Wagner and Choset, 2011). Depending on the application it can be very important to find a solution if one exists. On top of that completeness can assist in speeding up the algorithms. It is less costly to find any solution than it is to find an optimal or sub-optimal solution. This information can be used to see if the instance is solvable at all before executing the more costly algorithm.

2) Reduction based

In complexity theory, there exists a class of decision problems that can be solved in at most exponential time (relative to the input size), while verifying yes-instances requires only polynomial time. This class is called NP. A subset within NP, called NPC, is assumed to only contain decision problems to which any NP problem can be reduced in polynomial time. That is, any decision problem in the class NP (and by inclusion, also NPC) can be rewritten as an instance of any NPC problem, using less than exponential time. If this reduction is done correctly, a yes-instance in the original problem results in a yes-instance in the reduced problem, and vice-versa. Because of this property, a lot of time has been invested into creating solvers for some of these NPC problems. Since other NPC problems can be reduced to one of these already fleshed-out NPC problems, it prevents having to design a new algorithm for newly discovered NPC problems. The difficult part then lies only in the reduction. Some examples of NPC problems with existing solvers are Propositional Satisfiability (SAT), Integer Programming (IP), and Constraint Satisfaction Problem (CSP).

The decision variant of MAPF is proven to be a NPC problem (LaValle, 2006). The makespan decision variant gives an answer to the question: is there a solution to this MAPF instance, using no more than k time? Surynek has developed an efficient reduction from this decision variant to the SAT problem. Obtaining an optimal value is achieved by executing the decision variant on an increasing value of k ($k = k_0, k_0 + 1, ...$). Surynek calls this the incremental strategy.

3) A* based

A* is a single-agent path finding algorithm that is proven to be complete and optimal. A downside of A* is that executing A* on multiple agents will result in a search space which size grows exponentially with the number of agents. Newly developed algorithms that are based on A* therefore try to minimise the amount of search space that is traversed, while maintaining the optimal and completeness properties. Two very popular extensions to A* have been introduced by Standley, called Operator Decomposition (OD) and Independence Detection (ID).

ID tries to reduce the search-space by grouping agents together which do not have conflicts between each other. OD tries to reduce the search-space by making an educated choice in which possible movements to consider first. In the worst case, both improvements perform just as bad as A*. In a performance analysis in chapter IV, we will see that, in general, these improvements result in a faster performance.

4) Optimal

To find an optimal solution the required setting is a centralised setting (Khorshid et al., 2011). The reason for this is that for an optimal solution the routes of the agents are predetermined. Adjustments to the routes of the agents, like conflicts or deadlocks, cannot be resolved halfway through because an agent would have to adjust its optimal route, which could lead to a sub-optimal route and therefore a sub-optimal solution. Optimal solutions require at least as much time as sub-optimal solutions. An advantage is that the quality of the solution is better. These types of algorithms are used when it is more important to minimise either the makespan or the sum-of-costs instead of the time it takes to find the solution.
In the following subsections we will discuss variants of optimal solutions by using a few examples. These examples are state-of-the-art optimal MAPF algorithms. We will start off with discussing increasing cost tree search, afterwards we discuss conflict based search and finally we discuss branch, cut & price.

a) Increasing cost tree search

Increasing Cost Tree search (ICTS) search algorithms build a tree that consists of the costs of the best path for every agent. ICTS is an A* based algorithm. The algorithm builds a tree a using a top down approach where the root of the tree is the best path for every agent when the other agents are not taken into account. Children of a node are generated by adding a unit cost of 1 to one of the agents. For each of these nodes paths are calculated according to the distance that is written down for every agent. This search tries to combine individual paths of the agents so that there are no conflicts between

them. If this succeeds the optimal solution is found, otherwise it will try to do so on the next node.
The way the searches are performed on the nodes of the tree is in a breadth first search manner. By doing this an optimal solution will be found, because in every row in the tree the nodes have the same depth. Therefore every row has the same total sum-of-costs.

On the one hand the algorithm performs well when there are many chunks of open areas, but on the other hand it performs inefficiently when the environment is dense. Experimental results showed that on a number of domains ICTS outperforms the previous state-of-the-art A* approach by up to three orders of magnitude in many cases (Sharon et al., 2013).

b) Conflict-based search

Conflict-based algorithms are algorithms that find a set of constraints for individual agents so an optimal solution can be found. These types of algorithms do so by performing 2 types of searches. First there is a low level search which finds an optimal route. Secondly there is a high level search that finds constraints that it needs to impose on individual agents for an optimal solution. A constraint for an agent means that that agent cannot occupy a certain location at a certain time step. These constraints are in place to resolve conflicts between agents. Every time new constraints are imposed the low level search is performed again whilst taking the new constraints into account. By repeating this process eventually a set of constraints will be created that removes all the conflicts and returns an optimal solution (Sharon et al., 2015).

c) Branch, Cut & Price

Branch cut and price (BCP) is a more recent optimal state-of-the-art algorithm. It uses a combination of the strengths of CBS with the search performance of mixed integer programming (MIP). Because it uses MIP, it is a reduction based algorithm instead of an A* based algorithm (see III-C2). BCP is faster than CBS with a higher success rate (Lam et al., 2019). In the next chapter we are going to compare BCP and CBS more extensively.

5) Sub-optimal

Sub-optimal solutions in most cases use the distributed setting. That is because the solution does not need to be optimal. The benefits are that the algorithms scale better on larger graphs or with more agents. The algorithms are also in most cases faster than their centralised counterparts. A disadvantage is that in most cases no completeness guarantees are provided (Wang and Botea, 2008). Sub-optimal solutions can be divided into three types, namely rule-based, search-based and hybrid-based approaches. Below we will discuss all three with the use of examples.

a) Rule-based

Rule-based algorithms have agent-specific rules in place for different scenarios. They usually do not include a massive search like search-based algorithms. Rule-based solvers usually guarantee to find a solution very fast, but those solutions are in most cases far from optimal. An example of such an algorithm is tree based agent swapping strategy (TASS). TASS uses a centralised approach with a sub-optimal solution. The algorithm is centralised because all rules apply to all agents. It uses an algorithm called Graph-to-Tree-Decomposition (GTD) to induce trees from graphs. TASS can only be used on trees, according to Khorshid et al. this applies to a lot of MAPF instances.

TASS provides no guarantees to the quality of the solution, but it is complete for tree graphs. It is also very fast even with a cluttered search space where there are many agents in a small space. As was discussed in the paper on a tree with a 1000 nodes and 996 agents (only 4 empty spaces where agents are allowed to move) a solution was found in 8 seconds (Khorshid et al., 2011). Other rule-based algorithms are for example Bibox and push and rotate (Surynek, 2009), (de Wilde et al., 2014).

b) Search-based

Search-based approaches do not use any rules to reach their goal destination, but only searches. Collisions are avoided because of adjustments in the searches or other systems that are in place, as we will shortly see in the example. Search-based algorithms are not the fastest algorithms, but they create high quality solutions (Stern et al., 2019). There are also optimal search-based algorithms, for example CBS that we discussed previously (Barer et al., 2014).

An example of a sub-optimal search-based algorithm is windowed hierarchical cooperative A* (WHCA*). It uses a decoupled approach with single-agent searches. In cooperative path finding an agent has full knowledge of other agents' travel plans. It achieves this by using a reservation table. Every agent, after performing its search, puts their states in the reservation table. This ensures that locations at certain time steps are impassable for other agents. This process eliminates deadlocks and collisions. A drawback however is that the order of the agents matters. The path of one agent could lead to the next agent not being able to find a path to its location, because at certain time steps locations are in the reservation table. In this case, the order of the agents matters if the paths of the two agents are not mutually exclusive.
The problem can be partially resolved with the use of a window. The window in WHCA* determines the size of the partial solutions created by the agents. To make sure the agents still walk in the right direction, an abstract

Algorithm	Optimal	Setting	Coupled/Decoupled	Source
M*	Yes	Distributed	Subdimensional expansion	(Wagner and Choset, 2011)
FAR	No	Distributed	Decoupled	(Wang and Botea, 2008)
WHCA*	No	Centralised	Decoupled	(Bnaya and Felner, 2014)
TASS	No	Centralised	Decoupled	(Khorshid et al., 2011)
CBS	Yes	Centralised	Combination	(Sharon et al., 2012)
ICTS	Yes	Centralised	Coupled	(Sharon et al., 2013)
BCP	Yes	Centralised	Combination	(Lam et al., 2019)

TABLE I: Overview of different MAPF algorithms

search is used. This abstract search is performed from the location of the agent to its goal and is abstract because it ignores all other agents and their reservations. (Silver, 2005).

Another example of search-based algorithms are bounded sub-optimal solvers. Multiple of these algorithms are described in (Barer et al., 2014) as well as some optimal search-based algorithms.

c) Hybrid-based

Hybrid-based algorithms use rules as well as an initial search for the optimal route. Flow Annotation Replanning (FAR) is a hybrid-based algorithm which works on a grid. Initially, for every agent the optimal route is calculated. Conflicts are not taken into account in the planning step. All plans are then executed. Rules are in place to resolve conflicts and if they result in a deadlock, a local (instead of global) re-planning step is done to break the deadlock. In other cases where a deadlock consists of multiple agents, one of the agents is temporarily forced off its trajectory so the other agents can continue their route.
An example of a rule implemented in this system is that an agent has to reserve its move for the next time step before moving there. On top of that agents are aware of the next few steps nearby agents will take. This way two different agents will never move to the same location. FAR is faster and uses less memory than WHCA*. That is because the completeness and optimality of the solution are traded for efficiency (Wang and Botea, 2008).

D. Algorithm Overview

In table I an overview is given of MAPF algorithms and the aspects that are discussed in this section.

IV. Performance Of MAPF Algorithms

As shown in chapter III, there are many different types of algorithms. Since these algorithms work in different ways, and have different objectives, it can be very hard to directly compare them. It is important to make these comparisons, since this grants us knowledge of which algorithms work best. A method of comparing different algorithms is benchmarking. Using this method we can create certain scenarios, and let multiple algorithms create paths. We can then compare the results of all the algorithms to show which one works the best in that situation. Benchmarking is the best approach to this problem, because it eliminates the differences that make the comparisons difficult. All algorithms are ran on the same environment, with the same goal. In order to find the most representative results, it is important to standardise the graphs we perform analysis on. In section IV-A we give an overview of some of the more commonly used graphs in previous research. The published research seems to focus primarily on grids, so section IV-A will do the same. Besides the graph that the algorithm is ran on, there are different properties of the environment that can noticeably impact the performance of the algorithms. Some of these will be discussed in section IV-B. Then, in order to give some idea of what algorithms perform the best, we have analysed multiple papers that perform benchmarking. The benchmarking approach and results can be found in section IV-C. Finally, we will give some research recommendations in section IV-D. This way researchers are able to recognise some of the gaps we found in literature, which would be beneficial to the field as a whole.

A. Test benchmarks

There exist a large library of open source MAPF benchmark instances[1] which are being used more and more frequently in the literature. These benchmark instances mainly consist of different types of grids on which MAPF algorithms can be ran. It is important to standardise these, since this allows for better comparison between different algorithms. In a paper by (Stern et al., 2019), an overview is given of the most commonly used benchmark graphs. We will give a small overview of the different types here, since this helps to illustrate the types of graphs that are used in the analysed papers. Knowledge of the instances is critical to understand how different types of environment effect the result of the algorithms.

1) $N \times N$ grids

Because of their simplicity, square grids are an easy way of testing. Sizes of N generally range between 8 and 32. When left completely open, without any obstacles, they are best suited for testing the effect of high agent density. Since there are no obstacles, any large path lengths will be caused by the interaction between agents.
It is also possible to add random obstacles to the grid. This forces the agents to create more complex paths to reach their destination.

[1] movingai.com/benchmarks/

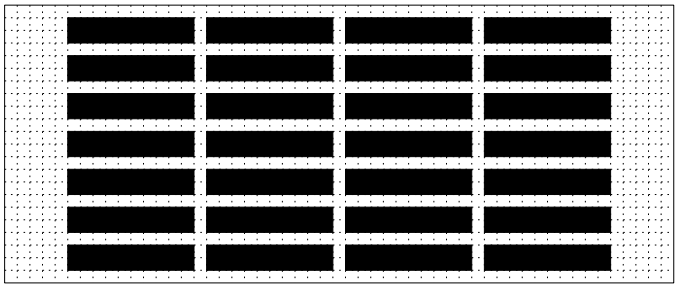


Fig. 4: Example of a warehouse grid. (Cohen et al., 2018)

2) Dragon Age Origins maps

The Dragon Age Origins maps have been standardised for testing grid algorithms by (Sturtevant, 2012), where they are available for download from a public repository too. These maps are popular instances in several MAPF benchmarks: (Felner et al., 2017; Stern et al., 2019). These maps are relatively large with on average over 20 000 walkable tiles, and contain very few obstacles. These maps better represent a real world scenario than square grids. They could, for instance, represent a small section of a city.

3) Warehouse grids

These grids are inspired by one of the older MAPF applications: warehouse robots. The grid is characterised by many narrow hallways, much like one would find in a warehouse (Liron Cohen et al., 2018). Figure 4 shows an example of a warehouse grid. Because it is modelled after a real world scenario, this type of grid is a good test for algorithms that want to solve this specific problem. If we are designing an algorithm to solve this specific problem, then it is not necessary for the algorithm to perform well on the other grid types.

B. Environmental difficulties

In certain scenarios the complexity of a problem cannot be represented by just the graph the algorithm is performed on. Luckily there are different properties that can be changed to perform more extensive testing. In this section we discuss some environmental properties that could cause problems for algorithms.

1) Sources and targets assignment

Besides the graph, the location of the start and end-points of each agent's path also have a great effect on the performance of an algorithm. There are multiple ways available to generate these **sources** and **targets** (Stern et al., 2019). Again we will give a small overview of the different techniques used.

- **Random:** Random assignment is the most common method found in the literature. This method is performed by giving each agent a random source and target from the pool of available grid slots.
- **Clustered:** For this method, the first agent gets source and target assigned at random. Every subsequent agent will have its source and target location chosen with a maximum distance from the first agent. This method is used to increase the difficulty of the MAPF problem, since it will ensure that the agents are close to each other.
- **Designated:** The final method can be used to better simulate real world scenarios. Here there are sets of designated source and target locations which every agent chooses from. In real world scenarios, like an automated warehouse, there are usually specific spots designated for humans. Using this method, it is possible to test the algorithm to path towards and from those spots in the most efficient way possible.

2) Dynamic environment

In general, MAPF environments are static (Majerech, 2017), meaning that the only thing that moves around are the agents. In a dynamic environment, it is also possible for the obstacles to move. When the obstacles move they might cut off the path of an agent, meaning that the agent has to find a new route. It is also possible that shorter paths are revealed. These continuous changes means that the algorithm has to continuously adjust itself to its new surroundings. Real world equivalents of a dynamic environment could be a bridge collapsing, or a street being blocked.

C. Experimental Results

While most papers perform some analysis with regards to performance, most of these Analyses are not very extensive. The benchmarks used are generally skewed towards the researchers' own algorithms. In this section we analyse a couple of papers which we deem extensive. These papers test different types of algorithms in different environments. The result of that analysis can be found in this section. This analysis is important, because it shows which algorithms work best in different situations. Since it is important what situation the algorithms are tested on, we have also provide a detailed explanation of the way the experiment is performed.

1) SAT-based approach

Surynek performs extensive bench-marking for a SAT-based approach (Surynek, 2017). A comparison is made between modified versions of the search-based Operator Decomposition/Independence Detection (OD+ID), Conflict-based Search (CBS), Increasing Cost Tree Search (ICTS), and 5 different encodings of a SAT-based approach. OD+ID, CBS, and ICTS are modified to optimise makespan instead of sum-of-cost. Surynek (2017) acknowledges that these modifications compromise the design of the algorithm to some extent. The graphs the algorithms were tested on were: 6x6, 8x8, and 12x12 4-connected grids with 20% of vertices occupied by randomly placed obstacles. The start and goal locations for each agent were chosen randomly. The algorithms were given a time-limit of 256 seconds to solve a single instance. The results were only taken into account when an algorithm could solve 10 random instances within this

time limit, for a certain number of agents. Analysis was performed on the run-time of the algorithms, as well as on the sum-of-costs.

When looking at the run-time in figure 5, we can clearly see that ICTS and CBS have an average run-time that is way faster than the SAT-based methods. All the search-based methods also perform better in terms of median run-time, which can be seen in figure 6. This result is present in all three grid sizes. However, the search based methods are only able to find solutions within the given time limit for agent numbers up to 7, 8, and 12 for the different grid sizes respectively. The SAT-based solutions however, are consistently finding results with agent counts about 2.5 times greater. This clearly shows that the SAT-based methods perform well when as the number of agents increases. Most of the different encoding types find solutions within 100 seconds on average, for all three map sizes.

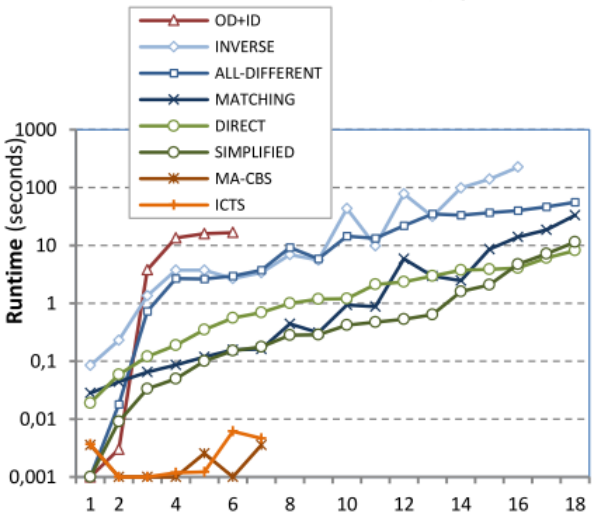


Fig. 5: Average run-time on 6 × 6 grid. Taken from (Surynek, 2017).

When looking at the sum-of-costs in figure 7, we can see a similar pattern. The search-based method OD+ID produces results with shorter paths with few agents, but cannot compute the paths when the agent number gets too big. It is also shown that the SIMPLIFIED encoding produces solutions with the fewest moves in about 75% of the cases.

a) Conclusion

From this paper we can conclude that SAT-based algorithms work very well on small graphs with high agent density. While search-based methods give good results both in run-time and solution quality, these algorithms do not scale well with the number of agents. It is also shown that the SIMPLIFIED encoding SAT-based algorithm yields results with the fewest moves in most cases. However, the experiment is only performed on relatively small grids with 20% obstacles. While Surynek states that the SAT-based algorithms work best on small, dense grids, it would have been interesting to see what

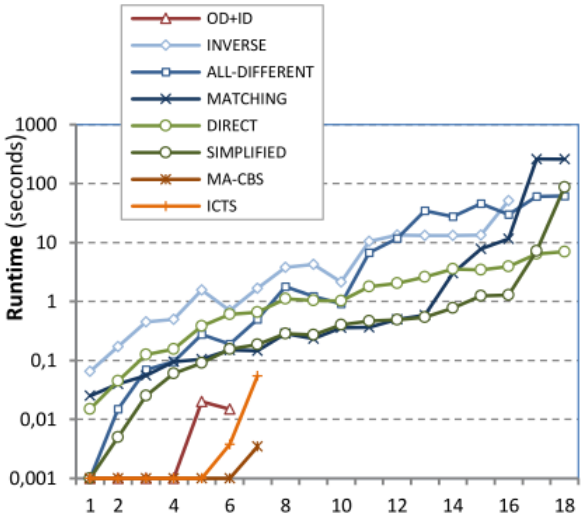


Fig. 6: Median run-time on 6 × 6 grid. Taken from (Surynek, 2017).

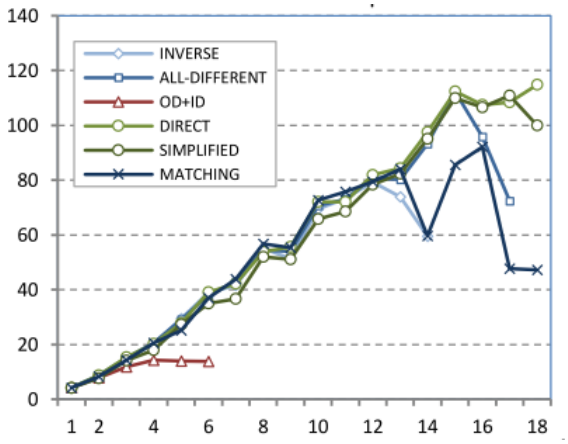


Fig. 7: Average sum-of-costs on 6 × 6 grid. Taken from (Surynek, 2017).

happens on larger, more sparsely populated grids.

*2) Cooperative A**

Silver has benchmarked a number of cooperative pathfinding algorithms (Silver, 2005). Comparisons are made between four algorithms: Local Repair A* (LRA*), Cooperative A* (CA*), Hierachical Cooperative A* (HCA*), and Windowed Hierachical Cooperative A* (WHCA*). WHCA* uses a window, therefore tests have been performed with three different window sizes: 8, 16, and 32. Tests are performed on a 32x32 4-connected grid, with impassable obstacles randomly placed in 20% of the grid locations. An example of this grid can be found in figure 1. Start and goal locations were randomly chosen on the grid, with the only constraint being that no two agents can share a start or goal location. Agents had to reach there destination within a limit of 100 turns, where each turn an agent either moves or waits. A possible move can also be for the agent to stay on the same spot. If an

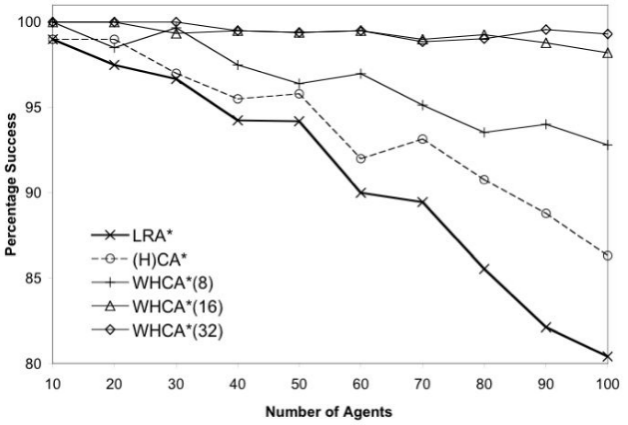


Fig. 8: Success percentage. Taken from (Silver, 2005).

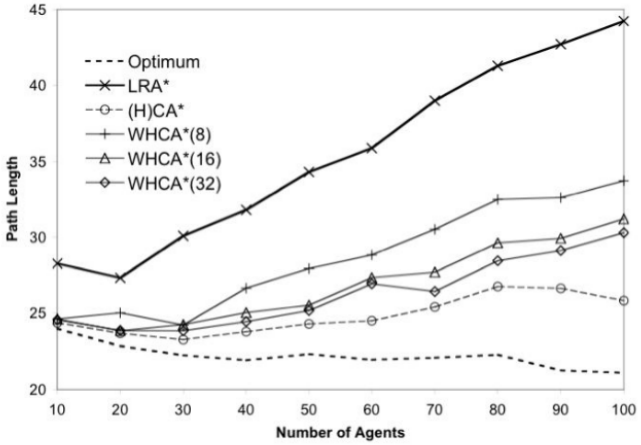


Fig. 9: Average path length. Taken from (Silver, 2005).

agent that exceeded this move-limit was unable to find a route, or collided with any other agent, the agent was considered a failure. Analysis was performed for: success percentage, average path length, number of cycles, and calculation time.

As figure 8 clearly shows, when there are only few agents present, all of the algorithms achieve a success rate near 100%. When the number of agents scales up, we can clearly see that LRA* has the biggest drop in performance. When there are 100 agents present, we see a success rate of only 85%. This result is in stark contrast with the results of WHCA* with window sizes 16 and 32, which still have success rates very close to 100%.

The resulting path lengths in figure 9 also shows clearly that LRA* is not very well optimised. While the average path lengths of all the other agents stay very close to the optimum, LRA* already shows an increase of approximately 5 steps. This difference changes to about 2 times as long when the number of agents is equal to 100. On this benchmark CA* and HCA* perform the best, consistently producing the shortest average paths. Of the windowed variants, the path length decreases when the window size is increased.

Next we look at the number of cycles an agent produces in its path. A cycle is counted when an agent revisits a grid location. This means that an agent uses a path that is inefficient, since the cycle could have also been skipped entirely. When looking at the number of cycles in figure 10, there really only is one outlier. LRA* consistently produces many cycles, which would also explain the large increase in path length. The average amount of cycles can increase to as much as 16 cycles per agent. All the other algorithms average at most 1.5 cycles with 100 agents present. This huge difference mostly comes from the fact that LRA* is not able to deal with bottleneck regions, while the other algorithms have a plan to work around those.

In this experiment, calculation time is divided into two

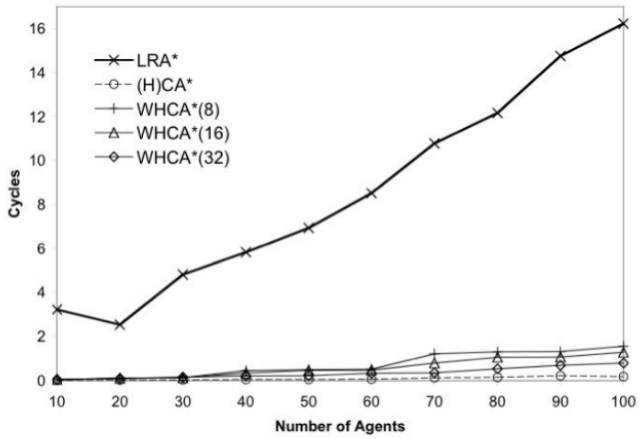


Fig. 10: Number of cycles. Taken from (Silver, 2005).

different types: total initial path calculation and maximum total calculation time per turn. Since the algorithms all need paths for individual agents first, this takes up a certain amount of time. In figure 11 we see that LRA*, WHCA* (8), and WHCA* (16) perform the best, all taking under 100 ms to calculate the paths. CA* and HCA* take way longer to initialise. This is caused by the fact that both these algorithms have to perform full-depth cooperative searches in space-time.

In figure 12 we see that LRA* has a very short calculation time per turn. This is quite a misleading statistic however, since LRA* takes many more turns than the other options. The window size of WHCA* has very little effect on the calculation time per turn.

a) Conclusion

All the benchmarks show that LRA* is outperformed by the Cooperative A* algorithms in environments with many agents. When increasing the number of agents, CA* and HCA* create the highest quality paths. However, because these algorithms take relatively long to calculate, they are not well suited for real-time applications. The window added by WHCA* gives the opportunity to create a trade-off between calculation speed and path

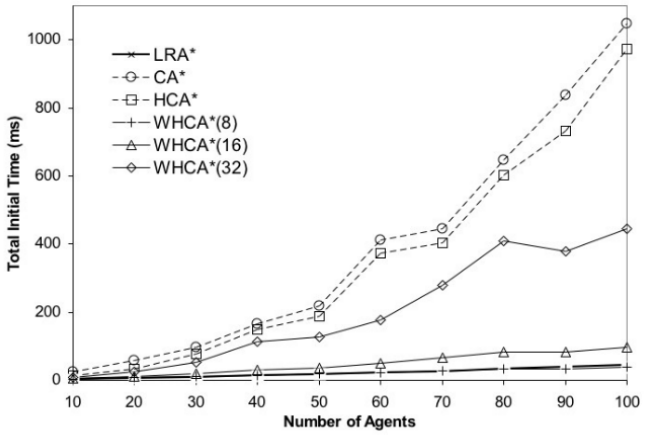


Fig. 11: Initial calculation time. Taken from (Silver, 2005).

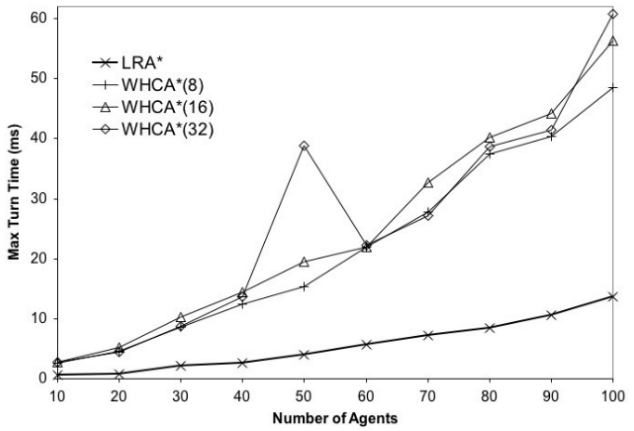


Fig. 12: Average calculation time per turn. Taken from (Silver, 2005).

quality. Increasing the window size will increase the initial calculation time, but also creating the highest quality paths. Behaviour will be similar to HCA*. Decreasing the window size will decrease the calculation time, but the path length will also increase. The behaviour of WHCA* is then more like LRA*. What this paper is missing to fully complete its analysis, is to add different types of algorithms to the comparison. All the algorithms are improved versions of each other, or different window sizes in the case of WHCA*. Right now, no conclusion of the true strength of the algorithms can be made.

3) Constraint Based Search

The optimal Constraint Based Search (CBS) algorithm is introduced in Sharon et al. (2012). To evaluate the performance of the algorithm, Sharon et al. let the algorithm compete against four others in two benchmarks. The algorithms used are A*, A* enhanced by OD (denoted A*+OD), ICTS, ICTS enhanced with a high-level pruning technique that uses information about small groups of up to 3 agents and their internal conflicts (ICTS+3E), and CBS.

a) 8x8 4-connected grid

Sharon et al. ran all algorithms on an 8x8 4-connected grid with 3 to 13 agents. The results are summarised in figure 13. For each instance, all algorithms were given a five minute time limit to find a solution. If an algorithm did not manage that, figure 13a shows NA. The run-times shown in figure 13a are averaged over 100 instances. The node number represents how many nodes the algorithm had to create in its search tree to find a solution. CBS has a high and a low level tree, so node numbers for both levels have been included. ICTS node numbers have been omitted because ICTS is not solely search-based. Figure 13b shows the success rate of some algorithms over a range of agent counts.

CBS outperforms A* and A*+OD by up to an order of magnitude in this benchmark, even though sometimes more nodes are generated. This is because CBS spends less time per node than A*+OD. The other two algorithms are better at keeping up. CBS only outperforms ICTS on instances with at least 8 agents, and ICTS+3E when instances have 9 or more agents.

b) Dragon Age: Origins maps

From the OPENAI library, 3 Dragon Age: Origins maps have been tested: *den520d*, *ost003d* and *brc202b* (Sturtevant, 2012). These maps are considerably larger than the 8x8 grid tested before, which is why only the three strongest algorithms, A*+OD, ICTS+3E and CBS, are present in this benchmark. Sharon et al. does show node counts for A* without OD and CBS, where can be seen that CBS almost always generates less nodes. Since CBS uses less time per node than A*, it is concluded that A* would be slower in all executed tests.

Figure 14 shows what the tested maps look like, with the achieved success rates in the graphs. From the presented results, one can conclude that ICTS+3E in all cases has a higher success rate than A*+OD. The results of CBS are mixed, however, ending up in first, second and third place depending on the map.

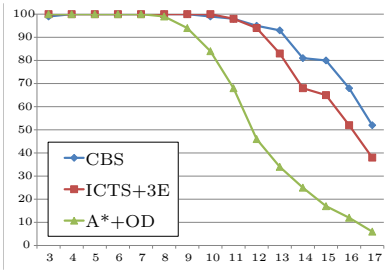
CBS ends up third in figure 14a, because there are barely any bottlenecks but abundant open spaces. In figure 14b there are a few bottlenecks and small open spaces, which resulted in a second place for CBS. The last map, figure 14c, has many narrow corridors and bottlenecks, and only few open spaces, resulting in CBS performing noticeably better than the other algorithms.

c) Conclusion

The presented benchmarks show that the relative performance of CBS highly depends on the graph used. A theoretical comparison with A* has shown that CBS performs better than A* based algorithms when there's a bottleneck, while A* performs better than CBS in open space. That finding is confirmed with the benchmarks. The paper acknowledges that more research is needed to show how much the used domain matters for the achieved performance. In 2019, Lam et al. created a

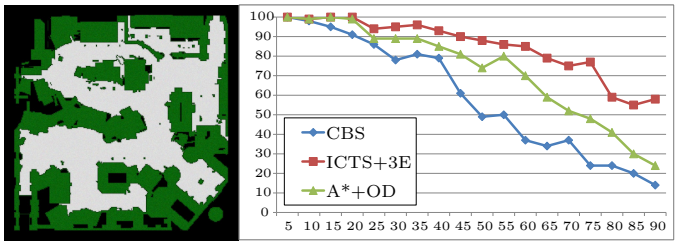
	#Generated Nodes				Run-time (ms)				
k	A*	A*+OD	CBS(hl)	CBS(ll)	A*	A*+OD	ICTS	ICTS3	CBS
3	414	**82**	7	229	14	3	1	1	2
4	2,843	**243**	31	1,135	380	10	2	1	13
5	19,061	**556**	42	1,142	9,522	50	5	5	13
6	64,734	**677**	42	1,465	47,783	90	**9**	10	16
7	NA	**4,451**	287	13,804	NA	1,122	92	**44**	186
8	NA	**8,035**	308	13,820	NA	1,756	271	**128**	211
9	NA	30,707	740	**25,722**	NA	7,058	2,926	921	**550**
10	NA	54,502	1,095	**34,191**	NA	21,334	10,943	2,335	**1,049**
11	NA	NA	2,150	**69,363**	NA	NA	38,776	5,243	**2,403**
12	NA	NA	11,694	**395,777**	NA	NA	NA	25,537	**15,272**
13	NA	NA	22,995	**838,149**	NA	NA	NA	45,994	**36,210**

(a) Node count and run-time on 8x8 grid

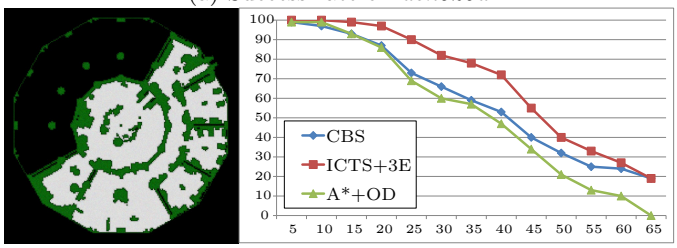


(b) Success rate on 8x8 grid

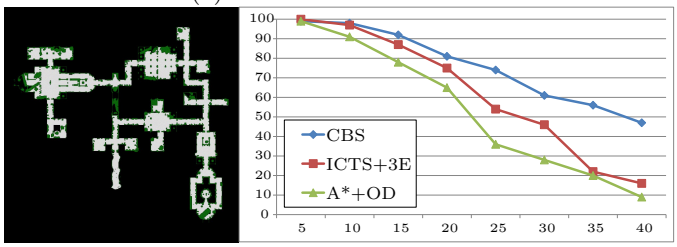
Fig. 13: 8x8 grid benchmark results from Sharon et al. (2012).



(a) Success rate on *den520d*



(b) Success rate on *ost003d*



(c) Success rate on *brc202b*

Fig. 14: Dragon Age: Origins benchmark results from Sharon et al. (2012).

new algorithm, branch-and-cut-and-price (BCP), which is faster than CBS in many cases (Lam et al., 2019). It would be interesting to see this benchmark rerun with BCP added to see how it copes with bottlenecks and open spaces compared to ICTS and A*+OD.

4) Search-based Optimal Solvers

Felner et al. (2017) gives a summary of search-based techniques for optimally solving MAPF under the sum-of-costs objective. Benchmarks where different classes of optimal algorithms are compared against each other are included too, with the goal to show that there is no universally winning algorithm. In Felner et al. (2017), five different optimal algorithm classes are identified: A*, M*, CBS, ICTS and SAT-based. For each family of algorithms, the best variant available to Felner et al. is reported: EPEA* for the A* family, ODrM* for the M* family, ICBS for the CBS family, ICTS+p for the ICTS family and MDD-SAT for the SAT based solvers designed for sum-of-costs.

Felner et al. performed three benchmarks using these algorithms. For each algorithm, the success rates on different instances is reported. Success rate refers to the percentage of instances that could be solved within a five minute time limit.

a) Dragon Age: Origins

For the first benchmark, Felner et al. used Dragon Age: Origins map *ost003d* from the Moving AI repository (Sturtevant, 2012). The measured success rates are shown in figure 15a. The plot shows that when less than 55 agents are used, ICBS has the highest success rate. Above that, ICTS generally has a higher success rate.

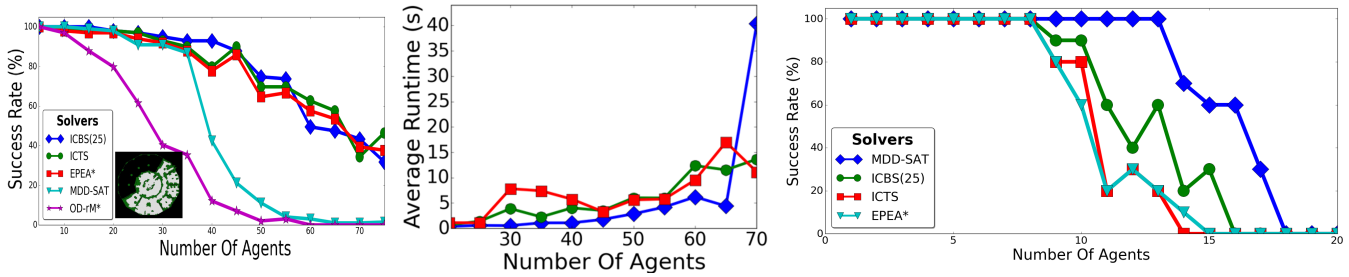
Felner et al. also measured consumed CPU time. Figure 15b shows the consumed CPU time for all instances solvable by all algorithms tested in this paper, excluding MDD-SAT and ODrM* because of their exceptional poor performance in this category. Here, ICBS almost always outperforms all other measured algorithms, even above 55 agents.

b) An 8x8 grid with 10% random obstacles

The second benchmark Felner et al. (2017) presents measures the performance of several algorithms on an eight-by-eight grid, where a random 10% of the available squares are filled with obstacles. The results, shown in figure 15c, show that MDD-SAT clearly outperformed all other tested algorithms. ODrM* has been excluded from this benchmark for an unknown reason.

c) Maze with corridor widths 1 and 2

The last benchmark published in Felner et al. (2017) evaluates the performance of all algorithms tested in this paper on two mazes, also taken from the Moving AI



(a) Benchmark 1 success rate (b) Benchmark 1 CPU time (c) Benchmark 2

	Success rate					Runtime (ms)				
W	EPEA*	ICTS	ICBS	SAT	M*	EPEA*	ICTS	ICBS	SAT	M*
1	84%	43%	51%	7%	87%	3,016	23,535	7,778	¿111,805	2,974
2	100%	100%	100%	59%	99%	2,033	2,012	239	¿243,045	1,935

(d) Benchmark 3

Fig. 15: Benchmark results from Felner et al. (2017).

repository (Sturtevant, 2012). The results are presented in figure 15d. The first maze, *512-1-{2,6,9}*, is a maze with one cell wide corridors. EPEA* and ODrM* had the highest success rate and lowest runtime here. The second maze, *512-2-{2,5,9}*, is a maze with two cell wide corridors. Again, EPEA* and ODrM* show very good results, achieving close to 100% success rate. This time, however, a lot more algorithms achieve 100% success rate. ICBS stands out, as it managed to get 100% success rate in almost $\frac{1}{10}$th of the runtime that the runner-up required.

d) Conclusions

The differences between the acquired results demonstrates that no universal winner can be assigned. The worst performer in some cases is the best in others, which shows the importance of benchmarking different algorithms for each use-case. Felner et al. did, however, provide some guidance on how the analysed algorithms would perform on different scenarios. Felner et al. concluded that CBS based algorithms performs better than A* based algorithms when there are more bottlenecks, small areas where only a few agents can pass through simultaneously. If there are areas with a high agent density, CBS will have to process too many conflicts. In such situations, A* based algorithms will usually perform better. ICTS is effective in situations where the sum of individual costs heuristic is close to the optimal solution and there are a lot of agents. The performance of MDD-SAT strongly depends on the quality of the used solver. MDD-SAT is strong if the problem is hard and the solver is still able to gather enough information on dependencies between the defined variables. This was the case in figure 15c. (Felner et al., 2017) is by far the most extensive benchmark covered in this section, with a representative algorithm from each main category. Unfortunately, it only covers optimal algorithms. It might be interesting to execute a similar benchmark for the representative algorithms of the sub-optimal categories and compare the results with this benchmark.

D. Recommended Further Research

At this point in time, there is no standardised method for testing the performance of a MAPF algorithm. This issue originates mostly from the fact that researchers create algorithms to solve specific issues. The benchmarking that is then performed is also generally skewed towards the newly created algorithm. The chosen graph and other environment variables are often chosen in a way that is favourable for that algorithm. Besides this, algorithms are often implemented in different programming languages and ran on different hardware every time researchers perform benchmarking. This means that even when an algorithm is tested in separate papers, it is still not possible to create a fair comparison between the two. We suggest that research is performed on a large set of different algorithms, where the researcher has no affiliation with any of the tested algorithms. This will eliminate any bias in chosen environment and algorithms. The importance of the research lies mostly in the algorithm diversity, since this is very much missing at this point in time. We believe that this research will help to improve the field of MAPF drastically.

V. Conclusion

The aim of this paper is to give a guide to readers that are currently searching for a solution to a pathfinding problem for multiple agents. We believe that the current literature does not explain MAPF at an entry level and does not give a broad enough overview of the different approaches within MAPF. Chapter II attempted to help readers to establish whether their problem can be (in part) modelled to this abstract version of pathfinding. Chapter III described several classes of algorithms, in the same way that is present in the current literature. This is for readers that have decided to use MAPF, or for engineers that are not satisfied with the performance of their current implementation. Current state-of-the-art algorithms are showcased and categorised according to

these classes. Chapter IV provided an empirical analysis into the actual performance of these algorithms. We looked at differences in both run-time and total path lengths. We also discussed how differences in performance are determined by the instance on which the algorithms run.

We have made an attempt to unify multiple publications in the field of MAPF. Significant effort has already been invested in this goal by the creators and contributors of the website mapf.info (Sven Koenig, 2019). The website provides a large database of publications, as well as open-source implementations, benchmarks, and a list of contact details on researchers that are currently working on this topic.

We recognise our study is not all encompassing. One of the shortcomings of this paper is the diversity of sources. We primarily used the aggregated publications on mapf.info, which may be prone to a "bubble" of topics and solutions. Even though the website is regularly managed by a handful of people, we cannot assume that the included research adequately covers all discoveries that are made in this field. Another shortcoming would be that we have not performed any research of our own. Our understanding is therefore limited by the findings that have been published in the current literature. Consequently, we cannot make concrete statements on the applicability of different algorithms in different environments (as discussed in section IV-B). Further studies could be performed to run a larger amount of algorithms on the already existing variety of benchmark environments. This would provide a more approachable way for engineers to decide which algorithm to use in their situation.

What we found to be lacking in this field of study is an agreement on terminology. Terms such as decoupled, coupled, centralised and distributed are used freely and it is often unclear what is actually meant. Furthermore, publications that introduce a new algorithm are often only interested in their performance relative to algorithms that are related to their own. While this is to be expected - as no researcher has the time or energy to compare it with all existing algorithms - evaluation of performance is limited to their choice. On the website mapf.info a beginning has been made to solve this problem. There is however still a lot of work to be done. We recommend readers that want to perform their own performance analysis to use one of the benchmarks provided on the website.

Acknowledgments

We would like to thank Jesse Mulderij and Mathijs de Weerdt for their supervision and feedback in writing this paper.

References

Barer, M, G Sharon, R Stern, and A Felner (2014). "Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem". In: *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pp. 961–962.

Bnaya, Z and A Felner (2014). "Conflict-Oriented Windowed Hierarchical Cooperative A*". In: *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pp. 3743–3748.

Cirillo, M, T Uras, and S Koenig (2014). "A Lattice-Based Approach to Multi-Robot Motion Planning for Non-Holonomic Vehicles". In: *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pp. 232–239.

Cohen, L, G Wagner, D Chan, H Choset, N Sturtevant, S Koenig, and S Kumar (2018). "Rapid Randomized Restarts for Multi-Agent Path Finding Solvers". In: *Proceedings of the Symposium on Combinatorial Search (SoCS)*, pp. 148–152. URL: http://arxiv.org/abs/1706.02794.

Cohen, Liron, Matias Greco, Hang Ma, Carlos Hernandez, Ariel Felner, T. K. Satish Kumar, and Sven Koenig (2018). "Anytime focal search with applications". In: *IJCAI International Joint Conference on Artificial Intelligence* 2018-July, pp. 1434–1441. ISSN: 10450823.

Felner, A, R Stern, E Shimony, M Goldenberg, G Sharon, N Sturtevant, G Wagner, and P Surynek (2017). "Search-Based Optimal Solvers for the Multi-Agent Pathfinding Problem: Summary and Challenges". In: *Proceedings of the Symposium on Combinatorial Search (SoCS)*, pp. 28–37.

Jansen, M and N Sturtevant (2008). "Direction Maps for Cooperative Pathfinding". In: *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*, pp. 185–190.

Khorshid, M, R Holte, and N Sturtevant (2011). "A Polynomial-Time Algorithm for Non-Optimal Multi-Agent Pathfinding". In: *Proceedings of the Symposium on Combinatorial Search (SoCS)*, pp. 76–83.

Kim, Dong Gyun, Katsutoshi Hirayama, and Gyei Kark Park (2014). "Collision avoidance in multiple-ship situations by distributed local search". In: *Journal of Advanced Computational Intelligence and Intelligent Informatics*. ISSN: 18838014.

Koenig, Sven (2019). *Learn all about Multi-Agent Path Finding (MAPF)*. URL: https://mapf.info (visited on 04/06/2020).

Lam, Edward, Pierre Le Bodic, Daniel Harabor, and Peter J Stuckey (2019). "Branch-and-cut-and-price for multi-agent pathfinding". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19), International Joint Conferences on Artificial Intelligence Organization*, pp. 1289–1296.

LaValle, Steven M. (2006). *Planning Algorithms*. Cambridge University Press. ISBN: 9780511546877. URL: https://doi.org/10.1017/cbo9780511546877.

Li, Jiaoyang, Pavel Surynek, Ariel Felner, Hang Ma, T. K. Satish Kumar, and Sven Koenig (2019). "Multi-Agent Path Finding for Large Agents". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. Vol. 33, pp. 7627–7634.

Ma, H, S Koenig, N Ayanian, L Cohen, W Hoenig, S Kumar, T Uras, H Xu, C Tovey, and G Sharon (2016). "Overview: Generalizations of Multi-Agent Path Finding to Real-World Scenarios". In: *Proceedings of the IJCAI-16 Workshop on Multi-Agent Path Finding*.

Ma, H, J Li, S Kumar, and S Koenig (2017). "Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks". In: *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 837–845.

Ma, H, C Tovey, G Sharon, S Kumar, and S Koenig (2016). "Multi-Agent Path Finding with Payload Transfers and the Package-Exchange Robot-Routing Problem". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 3166–3173.

Majerech, Ondřej (2017). "Solving Algorithms for Multi-agent Path Planning with Dynamic Obstacles". In:

Pallottino, Lucia, Vincenzo G. Scordio, Antonio Bicchi, and Emilio Frazzoli (2007). "Decentralized cooperative policy for conflict resolution in multivehicle systems". In: *IEEE Transactions on Robotics*. ISSN: 15523098.

Sharon, G, R Stern, A Felner, and N Sturtevant (2012). "Conflict-Based Search for Optimal Multi-Agent Path Finding". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

Sharon, G, R Stern, A Felner, and N Sturtevant (2015). "Conflict-Based Search for Optimal Multi-Agent Pathfinding". In: *Artificial Intelligence* 219, pp. 40–66.

Sharon, G, R Stern, M Goldenberg, and A Felner (2013). "The Increasing Cost Tree Search for Optimal Multi-Agent Pathfinding". In: *Artificial Intelligence*. Vol. 195, pp. 470–495.

Sigurdson, D, V Bultiko, W Yeoh, C Hernandez, and S Koenig (2018). "Multi-Agent Pathfinding with Real-Time Heuristic Search". In: *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*. Vol. 2018-Augus, pp. 1–8. ISBN: 9781538643594.

Silver, David (2005). "Cooperative pathfinding". In: *Proceedings of the 1st Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2005*, pp. 117–122.

Standley, Trevor (2012). "Independence detection for multi-agent pathfinding problems". In: *AAAI Workshop - Technical Report*.

Stern, R, N Sturtevant, A Felner, S Koenig, H Ma, T Walker, J Li, D Atzmon, L Cohen, S Kumar, E Boyarski, and R Bartak (2019). "Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks [Position Paper]". In: *Proceedings of the Symposium on Combinatorial Search (SoCS)*, (in print).

Sturtevant, N R (2012). "Benchmarks for Grid-Based Pathfinding". In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.2, pp. 144–148. ISSN: 1943-0698 VO - 4.

Surynek, P (2009). "A Novel Approach to Path Planning for Multiple Robots in Bi-Connected Graphs". In: *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pp. 3613–3619.

Surynek, P (2017). "Time-Expanded Graph-Based Propositional Encodings for Makespan-Optimal Solving of Cooperative Path Finding Problems". In: *Annals of Mathematics and Artificial Intelligence* 81.3–4, pp. 329–375.

Surynek, P, A Felner, R Stern, and E Boyarski (2016). "An Empirical Comparison of the Hardness of Multi-Agent Path Finding under the Makespan and the Sum of Costs Objectives". In: *Proceedings of the Symposium on Combinatorial Search (SoCS)*, pp. 145–147.

Teng, Teck Hou, Hoong Chuin Lau, and Akshat Kumar (2017). "A multi-agent system for coordinating vessel traffic". In: *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*. ISBN: 9781510855076.

Wagner, G and H Choset (2011). "M*: A Complete Multirobot Path Planning Algorithm with Performance Bounds". In: *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pp. 3260–3267.

Wagner, G and H Choset (2015). "Subdimensional Expansion for Multirobot Path Planning". In: *Artificial Intelligence* 219, pp. 1–24.

Wang, C and A Botea (2008). "Fast and Memory-Efficient Multi-Agent Pathfinding". In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 380–387. ISBN: 9781577353867.

de Wilde, B., A. W. Ter Mors, and C. Witteveen (Oct. 2014). "Push and Rotate: a Complete Multi-agent Pathfinding Algorithm". In: *Journal of Artificial Intelligence Research* 51, pp. 443–492. URL: https://doi.org/10.1613/jair.4447.

Wurman, Peter R., Raffaello D'Andrea, and Mick Mountz (Mar. 2008). "Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses". In: *AI Magazine* 29.1, pp. 9–19. URL: https://aaai.org/ojs/index.php/aimagazine/article/view/2082.