Solving the Colored Multi-Agent Path Finding with Waypoints Problem using SAT

Master Thesis

Jeroen van Dijk

Delft University of Technology

TU Delft

Solving the Colored Multi-Agent Path Finding with Waypoints Problem using SAT

by

Jeroen van Dijk

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday October 14, 2022 at 13:00 AM.

Student number:	4554590	
Project duration:	November 5, 2021 – October 14, 2022	
Thesis committee:	Prof.Dr. M.M. de Weerdt,	TU Delft, advisor
	Dr. E. Demirović,	TU Delft, supervisor
	Dr. J.T. van Essen,	TU Delft, examiner

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Preface

If anyone would have told me 12 years ago I would be writing this master thesis I would not have believed them one bit. But somehow here we are and I am quite happy with where I am. Just like this thesis, the journey was not always easy but if you enjoy what you are doing and you have people that help you than in the end anything is doable. Therefore I would like to thank my supervisors Mathijs de Weerdt and Emir Demirović for their invaluable advice during this project, and my parents and friends for their support during the overall journey.

Jeroen van Dijk
Rijnsburg, October 2022

Abstract

The Multi-Agent Path Finding (MAPF) problem is the problem of planning paths for multiple agents without any collisions. There are also many variants such as the waypoint variant, where each agent also has a set of waypoints it must visit before reaching its goal. The colored variant, in which the agents are grouped into teams and each team has a set of targets that need to be reached. And the colored MAPFW variant is a combination of the two. This paper presents an SAT-solver for the sum-of-costs objective for these variants based on the sum-of-costs SAT solver for MAPF. It also introduces a MaxSAT solver for the makespan objective while also minimizing the sum-of-costs, where instead of having a cardinality constraint on the cost it is minimized. Experimental evaluation showed that these methods perform well on different graph types compared to existing algorithms and that MaxSAT solves more instances than SAT but is only optimal in 90% of the instances.

Contents

Preface i

Abstract ii

1 Introduction 1

2 Problem definition 2
- 2.1 MAPF . . . 2
- 2.2 MAPFW . . . 2
- 2.3 Colored MAPF . . . 2
- 2.4 Colored MAPFW . . . 2

3 Related work 3
- 3.1 A*-based solvers . . . 3
  - 3.1.1 Independence detection . . . 3
  - 3.1.2 Operator decomposition . . . 4
  - 3.1.3 EPEA* . . . 4
  - 3.1.4 M* . . . 4
- 3.2 Increasing Cost Tree Search . . . 4
- 3.3 Conflict-based search . . . 4
- 3.4 Reduction based solvers . . . 6
- 3.5 SAT . . . 6
  - 3.5.1 Time expansion graph . . . 6
  - 3.5.2 SAT encoding for sum-of-costs . . . 6
  - 3.5.3 MDD-SAT . . . 8
  - 3.5.4 SMT-CBS . . . 9
  - 3.5.5 Sparse-SMT-CBS . . . 9
  - 3.5.6 DPLL(MAPF) . . . 9
- 3.6 Solving Colored MAPF problems . . . 9
- 3.7 Solving MAPFW problems . . . 10
- 3.8 Comparison between algorithms . . . 10

4 Contribution 11
- 4.1 Basic Encoding . . . 11
- 4.2 The waypoint encoding . . . 12
- 4.3 The colored encoding . . . 13
- 4.4 MaxSAT Encoding . . . 14
  - 4.4.1 The difference between SAT and MaxSAT . . . 14
  - 4.4.2 Finding an upperbound . . . 15

5 Experimental Evaluation 16
- 5.1 The graph types . . . 16
- 5.2 The experimental setup . . . 16
- 5.3 Discussion . . . 17
  - 5.3.1 Colored MAPF . . . 17
  - 5.3.2 MAPFW . . . 18
  - 5.3.3 colored MAPFW . . . 18

6 Conclusion 19

7 Future work 20

References 21

A Results 25

1

Introduction

The pathfinding problem, in which an agent must traverse a graph from a starting vertex to a goal vertex, is a classic AI problem that has been solved efficiently for more than 50 years [18]. More recently the Multi-Agent Path Finding (MAPF) problem has seen a lot of research because of its use in warehouses [59], aircraft-towing vehicles [29], videogames [38] and train shunting units [30]. In the MAPF problem, there are multiple agents, which all need to traverse the graph from their starting vertex to their goal vertex without colliding with the other agents. There are 5 different types of collisions [43] but generally, only the vertex and edge conflict are used. A vertex collision occurs when two agents are at the same vertex at the same time, while an edge conflict occurs when two agents want to traverse the same edge at the same time. The MAPF problem has two objective functions, but finding optimal solutions is NP-hard for both [64] [46] [31]. The two objective functions are minimizing the makespan or the sum-of-costs of the solution. The sum-of-costs is the total sum of all paths while makespan [63] is the total number of timesteps. There are also two types of solving techniques. Reduction-based solvers, which reduce the MAPF problem to another NP-hard problem, generally solve the makespan objective. While search-based solvers such as A* generally solve the sum-of-costs objective.

MAPF also has extensions, such as anonymous MAPF, in which all the agents share a set of goals and it does not matter which agent goes to which target. A more generalized version of this problem is colored MAPF, in which there are teams of agents that have a set of goals. Now every agent needs to travel to one of the targets assigned to its team. MAPFW is the variant of MAPF where each agent must also visit a set of waypoints. While there has been a lot of research into MAPF, there has been little research into the variants of MAPF and generally only for search-based solvers that optimize the sum-of-costs. Thus there is a gap for reduction-based solvers for these variations for both objective functions. So this paper will introduce a reduction-based solver for both variants under the sum-of-costs objective and compare them to an existing search-based solver. It also introduces a new reduction-based solving method that solves the makespan objective while also minimizing the sum-of-costs. This method is compared to the introduced reduction-based solver to see if it is more efficient and to compare the quality of solutions. Lastly, this paper introduces the colored MAPFW problem, which combines the colored MAPF and MAPFW problems into one and which resembles real-life instances even more. But since there has been no research into this problem, our methods used for the individual variants will be combined and compared to look at the difficulty of the colored MAPFW variant.

This paper is divided into five sections, the first of which provides a summary of the quite extensive existing literature. In the following section, my contribution to the literature is introduced. Then there is a section that describes my experimental setup, shows the results, and explains them. The last two sections are the conclusion and the future work.

2

Problem definition

2.1. MAPF

The Multi-Agent Pathfinding (MAPF) problem [43] is the problem of finding paths for n agents $a_1, a_2, ..., a_n \in A$ in the graph $G = (V, E)$. Each agent a has a starting vertex s_i and a goal vertex g_i, and at every time-step t the agent can either move to a neighbour vertex or wait at its current vertex. A solution is then a path π_i for every agent $a_i \in A$ where $\pi_i(0) = s_i$ and $\pi_i(t) = g_i$, without any vertex or edge conflicts. Where a vertex conflict occurs when $\pi_i(t) = \pi_j(t)$ and an edge conflict occurs when $\pi_i(t) = \pi_j(t+1) \wedge \pi_i(t+1) = \pi_j(t)$ for any $t \wedge i \neq j$. The objective function is either given by the sum of costs meaning that the cost of a solution is given by the sum of all individual solutions $\sum_{i=1}^{n} |\pi_i|$, or by the makespan which is the maximum cost of all the individual paths, or formally defined as $max_{1 \leq i \leq n}(|\pi_i|)$. Lastly, in this paper it is assumed that an agent does not disappear upon reaching its target and that waiting at the target location adds no extra cost to the solution. [43]

2.2. MAPFW

In the Multi-Agent Pathfinding with waypoints (MAPFW) problem each agent must visit a set of waypoints W_i. A solution now also has the constraint that $\forall w \in W_i$ is in π_i for every agent.

2.3. Colored MAPF

Colored MAPF is a generalization of anonymous MAPF, in which there is a set of targets and it does not matter which agent goes to which target as long as there is a one-to-one mapping. In colored MAPF there are teams of agents which have a set of targets and now the agents need to move to a target from the set assigned to its team. Anonymous MAPF is therefore colored MAPF but with only one team. More formally, in Colored MAPF [40] there are j teams consisting of k_j agents such that $\sum_{i=1}^{j} k_i = n$. Each team j has a set of agents $a_j = \{a_1^j, a_2^j, ..., a_{k_j}^j\}$ and a set of k_j goal vertices $g_j = \{g_1^j, g_2^j, ..., g_{k_j}^j\}$. Now each agent a_i^j has a path π_i^j where $\pi_i^j(0) = s_i^j$ and $\pi_i^j(t) \in g_j$. A solution is now valid when at $t = max(\pi_i^j) \forall j, \forall i \in a_j$ every $g \in g_j \forall j$ contains an agent.

2.4. Colored MAPFW

To obtain the combined problem the Colored MAPF problem will be extended by assigning every agent a_i^j a set of waypoints W_i^j that it needs to visit. This means that for a solution to be correct $\forall w \in W_i^j$ has to be in π_i^j must hold.

3

Related work

Because the MAPF problem is a NP-hard problem [64] [46] [31] there has been done both research on both optimal and sub-optimal solving methods. The focus of this paper will mainly be on optimal solvers, and this section will first take a look at the search-based solvers such as A*-based solvers and conflict-based search that optimize the sum-of-costs objective. Then the research into reduction-based solvers that optimize the makespan objective but which can also be used for the sum-of-costs objective [54] will be explained. The sections that follow will look at the research that has been done into the relevant variants of the MAPF problem and the final section attempts to make a comparison between the existing algorithms.

3.1. A*-based solvers

A* is a well-known polynomial algorithm that solves the pathfinding problem for a single agent by expanding nodes in such a way that the node with the lowest cost + heuristic is chosen until it finds the target. In the MAPFW version of A*, a node represents the location of all agents and neighboring nodes are the non-conflicting combinations of actions that the agents can take. Thus the branching factor $b = \Pi_{i=1}^{k} b(a_i)$ where $b(a_i)$ is the branching factor of agent a_i is exponential. Because given a 4-connected grid $b(a_i) = 5$ holds most of the time and thus $b = 5^k$. This is the main disadvantage of the A* algorithm, but there are improvements to the algorithm such as independence detection (ID) and operator decomposition (OD) [42] as well as the Enhanced Partial Expansion A* (EPEA*) [17] and M* algorithm [58].

3.1.1. Independence detection

The main idea of Independence detection (ID) [42] is to split the agents into independent groups to reduce the number of agents in the problem, which grows exponentially with the number of agents. Two groups of agents are independent when both groups have an optimal solution that does not have conflicts. ID works by first solving the problem for all the agents and ignoring the other agents. It then groups the agents whose paths contain a conflict and finds a solution for this group. The merging of agents into larger groups continues until there is a solution without conflicts. The runtime is now determined by the number of agents in the largest group rather than the total number of agents.

Algorithm 1 The ID framework [34]

```
Assign each agent to a singleton group
Plan a path for each group
while No conflicts occur do
    validate the combined solution
    if conflict found then
        Merge two conflicting groups into a single group
        Plan a path for the merged group
return paths of all groups combined
```

3.1.2. Operator decomposition

Operator decomposition (OD) [42] shrinks the branching factor by limiting the number of moves per node. It does so by ordering the agents and only expanding the first one. This will generate intermediate nodes which when expanded will only expand the next agent, resulting in more intermediate nodes. Only regular nodes will be generated when the last agent is expanded. The main advantage of this method is that the A* planner does not need to distinguish between standard and intermediate nodes, allowing it to always expand the best node.

3.1.3. EPEA*

Partial Expansion A* (PEA*) [60] reduces the branching factor by only expanding nodes with the same cost as the parent node. Thus when node n is expanded, only the children n_c are added for which $f(n) = f(n_c)$. This greatly reduces the generated nodes but ensure that the optimal path is not discarded n is added as well with $f(n) = f(n_d)$ where n_d is the discarded child with the lowest f value higher than $f(n)$. Enhanced Partial Expansion A* (EPEA*) [17] further improves upon PEA* by employing an Operator Selection Function which generates the list of operators that will generate children with the value of $f(n)$. This greatly reduces the runtime since it does not generate nodes that are immediately discarded.

3.1.4. M*

M* [58] is similar to ID in the fact that it changes the dimensionality of agents based on conflicts. When a node in M* is expanded all the agents that are not in conflict will follow their optimal move, resulting in only one child node. When a conflict occurs the agents involved are added to the conflict set, which is then added to all nodes in the current branch and which are added again to open. When such a node is expanded, all the non-conflicting agents make their optimal move, but all the possible moves of conflicting agents are added. Recursive M* (rM*) uses M* recursively when a conflict is found to generate an optimal solution for the agents in the conflict. When expanding nodes in the main function this can be used as the optimal path for conflicting agents. It is also possible to build rM* on top of OD resulting in ODrM* [13]. This could then be used as the underlying planner for the MA-CBS algorithm described later.

3.2. Increasing Cost Tree Search

Increasing Cost Tree Search (ICTS) [36] is a two-level solver that is different from A* but introduces a key concept that will be used by other solvers.

The high level of ICTS searches the increasing cost tree (ICT). Every node consists of a k-ary vector $[C_1, C_2, ..., C_k]$ where C_i is the cost of the solution of agent a_i. The vector of the root of the tree consists of all the costs of all optimal paths $[opt_1, opt_2, ..., opt_n]$. The k children are then generated by increasing the cost of one agent by 1 giving the vectors $[opt_1 + 1, opt_2, ..., opt_n], [opt_1, opt_2 + 1, ..., opt_n], ..., [opt_1, opt_2, ..., opt_n + 1]$. The total cost of a node is the sum of the vector, and thus all nodes on the same height have the same total cost. This means that searching the tree breadth-first yields an optimal solution. An example of an ICT can be found in Figure 3.1.

The low level must find non-conflicting solutions for the ICT nodes visited by the high level such that the cost of the path of agent a_i is exactly C_i. To do so a multi-value decision diagram (MDD) is used which stores all the paths for agent a_i with cost C_i [41]. As a result, the low-level only searches the Cartesian product of the MDDs for a set of k paths without conflicts. Generating the MDD can be done efficiently because it is a breadth-first search to depth C_i from the starting location of agent a_i. Furthermore, if C_i increases by one, the previously generated MDD can be reused to quickly generate the new one.

3.3. Conflict-based search

Conflict-based search (CBS) [34] consists of a low-level and a high-level solver. In the high-level solver, a constraint tree (CT) is created. Each node N in this tree has a set of constraints, a solution consistent with these constraints, and the cost of that solution. Constraints are either a vertex constraint $< a_i, v, t >$, in which an agent a_i can not be at vertex v at time t, or an edge constraint $< a_i, v, u, t >$ which prevents an agent a_i from traversing the edge between u and v at time t. The root of the tree has

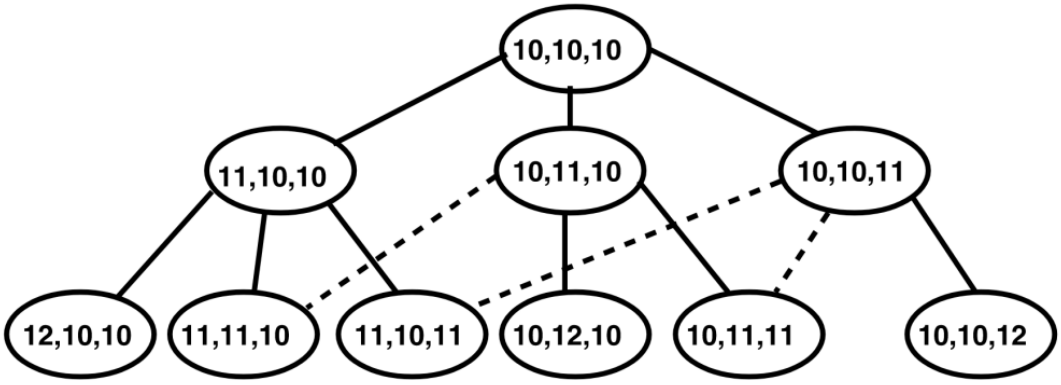

Figure 3.1: ICT for three agents. [34]

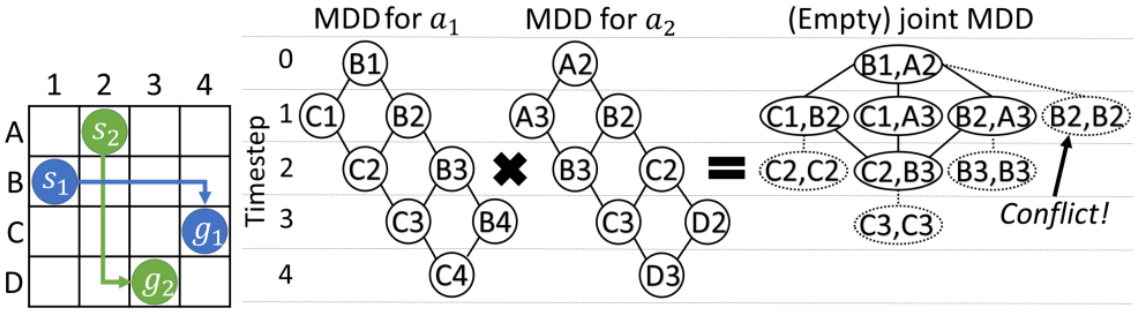

Figure 3.2: A MAPF instance on a 4-neighbor grid with the corresponding MDDs and joint MDD. [22]

no constraints and a solution is generated by the low-level solver. In case of a conflict in the solution, two children will be generated with a constraint for each of the agents in the conflict. These children also inherit all the other constraints of the parent. An example of a MAPF instance with CT can be seen in Figure 3.3.

CBS has many improvements such as meta-agent CBS (MA-CBS) [35], in which conflicting agents are merged into a meta-agent, which is then considered as a single agent. A meta-agent will never be split up lower in the tree but can merge with other (meta-)agents. The low-level search for a meta-agent is now a MAPF solver.

MA-CBS has been further improved to create improved CBS (ICBS) [4] by using Merge and Restart (MR), prioritizing conflicts (PC), and bypass (BP) [5]. In MR a new CT tree is created when two agents are merged instead of merging them at the current node. With PC the cardinal conflicts are split before the semi- and non-cardinal conflicts where a cardinal conflict is one that always increases the cost of the solution. BP attempts to reroute agents to bypass a conflict, but it can only work with semi- and non-cardinal nodes. These types of conflicts are found by using MDD [41]. A cardinal conflict at timestep t can be found if the widths of the MMDs of the two agents are 1. Figure 3.2 shows the MDDs for a_1 and a_2 and because the width is 2 at timestep 2 the conflict $< a_1, a_2, B2, 1 >$ is non-cardinal [22].

ICBS-h [12] has extended ICBS by adding an admissible heuristic to it, which has then been further researched and improved into CBS with improved heuristics (CBSH2) [22]. CBSH with rectangle reasoning by MDDs (CBSH-RM) [24] appears the best way to handle rectangle conflicts and a general improvement. CBSH with Mutex Propagation (CBSH-MP) seems to be the final improvement, and it uses techniques also seen in CBSH2. Lastly, if there are many corridors, a chain of connected vertices with a degree of 2, CBS could benefit from breaking the corridor symmetry [23].

In problems with many agents and high contention Lazy CBS (LCBS) [15] performs better than

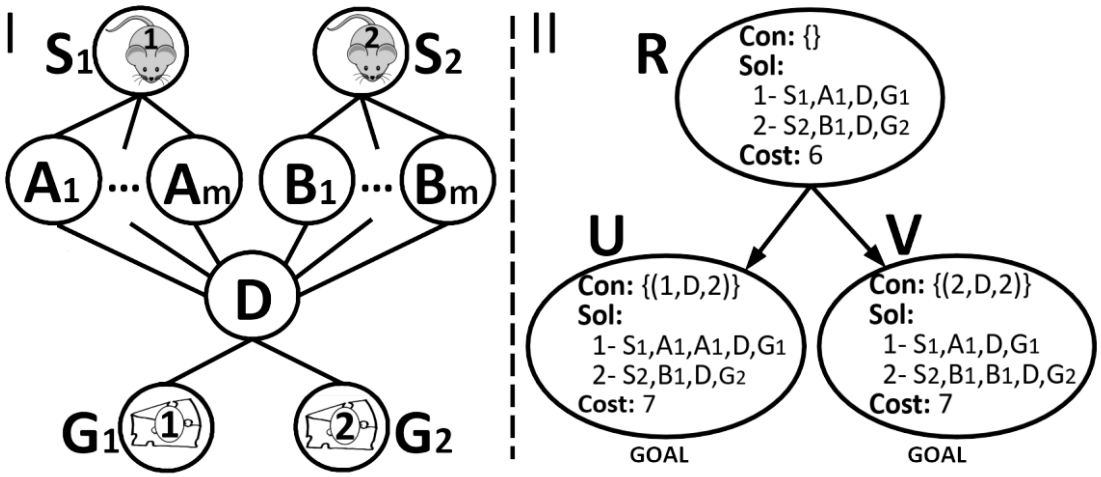

Figure 3.3: (I) A MAPF instance. (II) Its CT. [34]

ICBS-h. The high-level solver of CBS is replaced in LCBS by a lazily constructed constraint programming model with nogoods. This will store the results of conflicts, so it does not need to resolve conflicts it has already solved unlike CBS and its improvements.

3.4. Reduction based solvers

Reduction-based solvers reduce the MAPF problem to an NP-hard problem with state-of-the-art solvers. One such reduction is done by modeling MAPF as a network flow problem and solving it with Integer Linear Programming (ILP) [61]. Answer Set Programming can also be used to solve MAPF optimally [11], but the reduction to Boolean Satisfiability (SAT) is the most promising [52]. This algorithm uses a fast polynomial algorithm to find a makespan-suboptimal solution and replaces the sub-sequences with computed optimal sub-solutions. This paper introduced the inverse and all-different encodings but two new encodings have been researched since then, namely matching [47] and direct [45]. There are also two approaches [49] [50] that generate optimal solutions immediately without the need for a makespan-suboptimal solution.

All of these reduction-based solvers solve the MAPF problem for the makespan objective and adjusting them to solve the sum-of-costs objective generally requires a new reduction. Nonetheless, a sum-of-costs encoding for SAT [54] has been developed, and even more improvements have been made [56] [7]. Because this is the basis for this paper the sections that follow will explain in detail how this algorithm works, but first an important data structure known as the time expansion graph is explained.

3.5. SAT

3.5.1. Time expansion graph

A time expansion graph (TEG) is a directed acyclic graph. At every timestep from 0 to μ all of the vertices of the graph G are duplicated. The edges between successive timesteps will then represent all possible actions. An example can be seen in Figure 3.4, where it is also clear that diagonal edges are the moves and the horizontal edges are the waiting action. Formally a TEG is defined as follows:

Definition 1. Time expansion graph of depth μ corresponding to graph $G = (V, E)$ is $TEG(\mu) = (V', E')$ where $V' = \{u_j^t | t = 0, 1, ..., \mu \wedge u_j \in V\}$ and $E' = \{(u_j^t, u_l^{t+1}) | t = 0, 1, ..., \mu - 1 \wedge (\{u_j, u_l\}) \in E \vee j = l)\}$ [54]

3.5.2. SAT encoding for sum-of-costs

This section will first show how the SAT encoding for makespan works and how it can be adapted to work for sum-of-costs. Given a makespan μ the encoder will generate a $TEG_i(\mu)$ for each agent a_i,

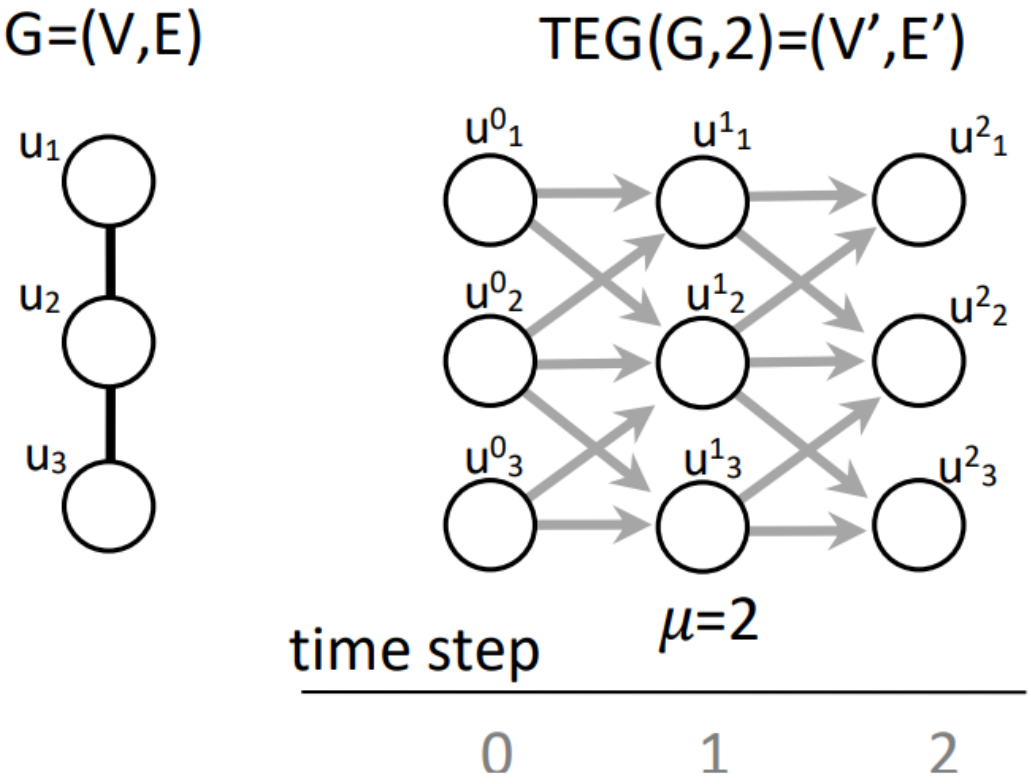

Figure 3.4: An example of a TEG with depth 2. [54]

and because non-conflicted paths in the TEGs corresponds to non-conflicted paths in the graph the existence of non-conflicting paths in the TEGs are encoded as an SAT problem. An optimal makespan can thus be found by starting at $\mu = 0$ and increasing it until the encoding is satisfiable.

Similarly, finding an optimal sum-of-costs will be accomplished by converting the optimization problem into a sequence of decision problems that checks if there is a solution of a sum-of-costs of ξ. This encoding does require the use of cardinality constraints for bounding ξ and bounding the makespan for the sum-of-costs. A cardinality constraint [39] [2] [27] is satisfied if given a set B of Boolean variables and a bound ξ there are no more than ξ variables in B set to $TRUE$. In the SAT encoding every agent's actions are mapped to a Boolean variable and then a cardinality constraint ξ is applied to these variables. This results in a sum-of-costs upper bound because no more than ξ moves can be performed by all agents. An optimal sum-of-costs solution can now be found by increasing ξ but the problem is that both the number of time expansions μ and the upper bound for the sum-of-costs ξ are variables.

Bounding the makespan for the sum-of-costs

So to find the relation between μ and ξ the following requirement is needed:

Requirement 1. All possible solutions with sum-of-costs ξ must be possible for a makespan of at most μ [54]

The following will be defined to find a μ that satisfies R1. Let $\xi_0(a_i)$ be the cost of the shortest individual path for agent a_i, and let $\xi_0 = \sum_{a_i \in A} \xi_0(a_i)$. ξ_0 is called the sum of individual costs (SIC) [36]. ξ_0 is an admissible heuristic for optimal sum-of-costs search algorithms since ξ_0 is a lower bound on the minimal sum-of-costs. ξ_0 is calculated by relaxing the problem by omitting the other agents. Similarly, μ_0 is defined as $\mu_0 = max_{a_i \in A} \xi_0(a_i)$. μ_0 is the length of the longest of the shortest individual paths and is thus a lower bound on the minimal makespan. Finally, let Δ be the extra cost over SIC. That is, let $\Delta = \xi - \xi_0$. [54]

Proposition 1. For makespan μ of any solution with sum-of-costs ξ, where $\xi = \xi_0 + \Delta$, it holds that $\mu \leq \mu_0 + \Delta$. Hence (R1) is satisfied for setting $\mu = \mu_0 + \Delta$. [54]

Using Proposition 1, it is possible to encode the decision problem whether there is a solution with a sum-of-costs of ξ, since if there is a solution of cost ξ it will be found within $\mu = \mu_0 + \Delta$ time expansions. Thus this proposition shows that the relation between the variables μ and ξ is decided by Δ. Thus in every iteration $\mu = \mu_0 + \Delta$ and the TEGs of depth μ for all agents are built. These TEGs are then encoded into a decision problem whether there is a solution with sum-of-costs $\xi = \xi_0 + \Delta$ and makespan μ. Thus an optimal solution can be found by starting with $\Delta = 0$ and increasing it until the encoding is satisfiable. Algorithm 2 shows the resulting algorithm.

Improving the use of the cardinality constraint

The encoding of a cardinality constraint is quadratically bounded by the number of variables in the set [39][27]. Thus bounding all the possible actions for the agents may be the bounding factor of the size of the total encoding. This is redundant because every agent a_i must move at least $\xi_0^i = \xi_0(a_i)$ times, and so only the actions of agents that make extra moves have to be bounded by Δ. This is done by generating a slightly modified TEG called TEG_i. TEG_i introduces two kind of edges E_i and F_i. E_i are the standard edges whose destination is at timestep $t \leq \xi_0(a_i)$ and F_i are the extra edges whose destination is at timestep $t > \xi_0(a_i)$. Figure 3.5 shows a TEG_i with depth 3 for an agent that can reach its goal in 2 steps. It also displays the dotted edges that belong to F_i and which will have a cardinality constraint. Thus only the extra edges will be bounded and they must sum to Δ.

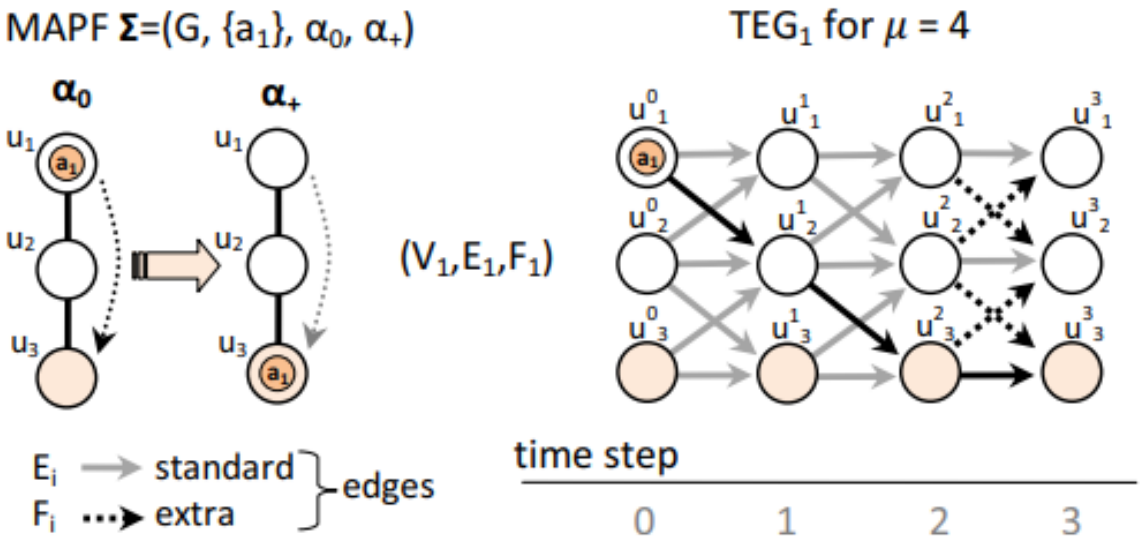

Figure 3.5: An example of a TEG_i with depth 3. [54]

3.5.3. MDD-SAT

The previously described encoding is called BASIC-SAT and it can be improved upon by reducing the size of the TEGs. To reduce the size of the TEGs vertices need to be removed without making the TEGs unsatisfiable. This can be done by using MDD as the data structure for TEG_i^μ which is TEG_i for μ time expansions. MMD_i^μ represents all paths from the current vertex of a_i to its goal vertex g_i of cost μ, and has thus a source node the vertex of a_i at $t = 0$ and a sink node g_i at $t = \mu$. It is self-evident that MMD_i^μ is a subgraph of TEG_i, because TEG_i includes all vertices of G at every timestep, whereas MMD_i^μ only includes the vertices that are on a path from a_i to g_i with cost μ. Furthermore TEG_i^μ can be replaced with $MMD_i^{\xi_0(a_i)+\Delta}$ because $\xi_0(a_i) + \Delta$ is the maximum cost that agent a_i can create. This is a further improvement because $\xi_0(a_i) + \Delta \leq \mu_0 + \Delta = \mu$. The encoding that uses MDD is called MDD-SAT [54].

It is also possible to combine MDD-SAT with independence detection (ID) resulting in MDD-SAT+SID and MDD-SAT+ID [55] [57]. It is not clear how much of an improvement these are compared

to MDD-SAT overall but there are cases where it is an improvement [56]. However, the sections that follow will only discuss MDD-SAT improvements and will not mention MDD-SAT+SID.

3.5.4. SMT-CBS

One of the main disadvantages of MDD-SAT is that the complete Boolean model must be fully specified. This limitation has been addressed in SMT-CBS [53], an SAT-based solver that uses incomplete Boolean models. In the incomplete Boolean model, not the entire MAPF problem is specified, but only the implication between the solvability of the MAPF problem and the Boolean model holds. Formally, this is defined as follows:

Definition 2. Incomplete propositional model Propositional formula $\mathcal{H}(\xi)$ is an incomplete propositional model of MAPF Σ if the following condition holds:
$\mathcal{H}(\xi)$ is satisfiable $\Longleftarrow$ Σ has a solution of sum-of-costs ξ. [53]

The solution of this incomplete model must be checked to ensure that all moves are legal. Similar to CBS if this is not the case a solution has been found and otherwise the model is refined by adding constraints that forbid the found illegal moves and a new solution is found.

3.5.5. Sparse-SMT-CBS

The lazy compilation to reduce the Boolean formulae is the main advantage of using SMT-CBS over MDD-SAT. However, this does not reduce the number of nodes and edges in the MDDs, so the number of variables and constraints of the paths in the MDDs are the same in both SMT-CBS and MDD-SAT, implying that on large maps this will be the bounding factor of the runtime. Therefore Sparse-SMT-CBS [51] introduces sparsification of the set of candidate paths of each agent, resulting in smaller formulae and thus a faster solving time by the SAT solver. This reduces the impact of large graphs on the runtime.

3.5.6. DPLL(MAPF)

The other major disadvantage of MDD-SAT which is emphasized in SMT-CBS is the fact that the SAT solver is a black box with which the main solver can not interact and must wait until it is finished. Especially in SMT-CBS, this can be problematic because a MAPF rule violation may be decided early on and the SAT solver does not know it is incorrect, and the main solver is unable to interact with it. DPLL(T) [32] [21] is an algorithm that uses the SAT solver with a decision procedure for the conjunctive fragment of some first-order theory T. This means that the decision procedure for the conjunctive fragment checks whether truth-value assignments are consistent with T. In our case, T will be the MAPF movement rules [7]. DPLL(MAPF) does not use the original Davis-Putnam-Logemann-Loveland algorithm [8] but rather the modern variant Conflict-Driven Clause Learning (CDCL) [26]. But the literature uses DPLL(MAPF) rather than CDCL(MAPF).

3.6. Solving Colored MAPF problems

The anonymous MAPF problem is a variant of the Colored MAPF problem, in which all the agents are on the same team, and which can be solved makespan-optimally in polynomial time [62]. The Colored MAPF problem can also be solved for the makespan objective by modifying the SAT and ILP reduction-based solvers [3] [47]. The Conflict-Based Min-Cost-Flow (CBM) algorithm [25] outperforms both of these algorithms on larger maps and both CBS and the ILP-based MAPF solver [61] on MAPF problems and can solve Colored MAPF problems as well. CBM is based on CBS where the high-level solver combines the agents in the same team into a single meta-agent. As a result, the low-level solver is executed per team and conflicts only happen between teams. The low-level solver uses a polynomial-time min-cost max-flow algorithm [16] on a time-expanded graph to generate a solution for the team such that no constraints are broken and each agent travels to a unique target. The objective function of CBM is to minimize the makespan but CBMxSOC [1] was created to optimize the sum of costs.

Other MAPF algorithms have been extended to solve the Colored MAPF problem such as A* [6], M* (pmM*) [10], EPEA* [20] and ICTS [36]. These algorithms use exhaustive matching to reduce the Colored MAPF problem to a set of MAPF problems, which can then be solved using the respective algorithm. This was further improved by first calculating a heuristic for each MAPF problem instance

and using this to first sort the MAPF problem instances and then prune the ones with a heuristic value greater than the lowest cost found thus far.

3.7. Solving MAPFW problems

Algorithms such as A* [37], CBS (CBSW) [19], M* (WM*)[9], MLA* (EMLA*) [14] and Branch-Price-and-Cut (BCP-MAPF)[28] have also been extended to solve the MAPFW problem. Furthermore, M* has also been extended to MS* to solve the Multi-agent simultaneous multi-goal sequencing and pathfinding (MSMP) problem [33]. These algorithms converted their underlying planners to a TSP-solver so that the optimal path to the goal through the waypoints could be calculated. With the use of caching the TSP-solvers did not explode the runtime too much in case of not too many waypoints.

A related problem is the multi-goal multi-agent pathfinding problem, in which an agent has a list of goals it all needs to visit but it can end at anyone. The Hamiltonian CBS (HCBS) [48] algorithm is similar to CBSW but it now uses a solver for the Hamiltonian path problem. SMT-CBS is also combined with HCBS to create SMT-HCBS which underperforms compared to HCBS but which might be further improved by using a lazy approach or the DPLL framework [DPLL(MAPF)].

3.8. Comparison between algorithms

It is difficult to compare all of the algorithms mentioned. Because, as previously stated, most algorithms have only been developed for one of the objective functions and thus comparisons have generally been limited to a class of algorithms. Furthermore, not all algorithms have been compared directly, so they have to be compared based on their comparison to a third algorithm. For example, LCBS, CBSH-RM, CSBH-MP, and CBSH2 all outperform CBSH which outperforms ICBS, even though there are only very few direct comparisons between the four. Furthermore, all the sum-of-costs SAT solvers are only compared to CBS and ICBS. The results show that MDD-SAT outperforms ICBS in the cases that are compact with a high number of agents and thus a high number of conflicts. However, ICBS is generally faster in the simpler cases with more open space because a search-based algorithm will know more about the map layout and MDD-SAT has some overhead as well. Therefore, it is expected that combining the two algorithm classes will result in a state-of-the-art solver. This appears to be the case for SMT-CBS which combines MDD-SAT and CBS but it slightly outperforms MDD-SAT and there are still instances that are easy for ICBS and hard for MDD-SAT. Although more research is needed it appears that DPLL(MAPF) could potentially be an improvement to SMT-CBS yet the MAPF consistency check seems too expensive to be called every time there is an assignment.

4

Contribution

4.1. Basic Encoding

The basic SAT encoding that will be extended for the variants is heavily based on the sum-of-costs BASIC-SAT encoding [54] and will be called the Basic Encoding. Some of the constraints have been replaced, and some additional constraints are added. As described in the paper the encoding is satisfied if a solution of ξ exists. To do so $\xi_0(a_i)$ the shortest path for an individual agent a_i is calculated. This can be used to calculate $\mu_0 = max_{a_i \in A} \xi_0(a_i)$, the length of the longest of the shortest individual paths of all agents and thus a lower bound for the makespan optimal solution. Δ which starts at 0 and increases every time a solution is not found as seen in Algorithm 2 is declared and used to update μ. Then a MDD_i of depth μ is generated for each agent $a_i \in A$. The vertices in MDD_i that agent a_i can occupy is denoted by V_i and the set of edges with a destination at timestep $t > \xi_0(a_i)$ is denoted by F_i. Because an agent, a_i must take at least $\xi_0(a_i)$ steps only the extra steps F_i need to be bound. The edges of waiting on the target are thus not in F_i since these do not add an extra cost to the solution. To create the encoding, first define the three variables that will be used in the encoding:

Algorithm 2 SAT algorithm for MAPF

```
Σ ← (G = (V, E), A, s_i, g_i)
μ_0 ← max_{a_i ∈ A} ξ_0(a_i)
Δ ← 0
while Solution not found do
    μ ← μ_0 + Δ
    for a_i ∈ A do
        build MDD_i(μ)
    solution ← SAT-Solver(Σ, μ, Δ)
    if solution not statisfied then
        Δ++
```

1. $\chi_j^t(a_i)$ for every $t \in 0, 1, ..., \mu$ and $u_j^t \in V_i$. $\chi_j^t(a_i)$ is true if agent a_i is at node v_j at timestep t
2. $\mathcal{E}_{j,k}^t(a_i)$ for every $t \in 0, 1, ..., \mu - 1$ and $u_j^t, u_k^{t+1} \in V_i$. This models the transition of agent a_i from vertex v_j to v_k through the edge (j, k) at the time from t to $t + 1$.
3. $\mathcal{C}_{j,k}^t(a_i)$ for every $t \in 0, 1, ..., \mu - 1$ and $u_j^t, u_k^{t+1} \in F_i$. This models the cost of movement across edges in F_i at the time from t to $t + 1$.

Let $T_\mu = 0, 1, ..., \mu - 1$ and all constraints apply to all agents $a_i \in A$ and for every timestep $t \in T_\mu$. The constraints can then be defined as follows:

C1: If an agent appears in a vertex at a given timestep, it must take one edge for the following time step. This is encoded by the constraint below, which must hold for every $u_u^t \in V_i$

$$\chi_j^t(a_i) \Rightarrow \bigvee_{(u_j^t, u_k^{t+1}) \in V_i} \mathcal{E}_{j,k}^t(a_i) \tag{1}$$

C2: If an agent takes an edge, it must be at the starting vertex of the edge at timestep t and the target vertex at timestep $t+1$. This is ensured by the following constraint for every $(u_j^t, u_k^{t+1}) \in V_i$

$$\mathcal{E}_{j,k}^t(a_i) \Rightarrow \chi_j^t(a_i) \wedge \chi_k^{t+1}(a_i) \tag{2}$$

C3: If agent a_i takes edge (j,k) no other agent can take the edge (k,j) because this causes an edge conflict.The following constraint ensures this for every $(u_j^t, u_k^{t+1}) \in V_i$

$$\mathcal{E}_{j,k}^t(a_i) \Rightarrow \bigwedge_{a_l \in A \wedge a_i \neq a_l} \neg \mathcal{E}_{k,j}^t(a_l) \tag{3}$$

C4: No two agents can appear in the same vertex at the same time since this would result in a vertex conflict. This is ensured by the following constraint for every pair of agents $a_i, a_l \in A$ such that $i \neq l$

$$\bigwedge_{u_j^t \in V_i \cap V_l} \neg \chi_j^t(a_i) \vee \neg \chi_j^t(a_l) \tag{4}$$

C5: An agent must occupy exactly one vertex at each timestep. The following constraint ensures this, but because it is an at-most-one constraint, it could also be implemented in other ways.

$$\bigwedge_{j,k \in V_i \wedge k<j} \neg \chi_j^t(a_i) \vee \neg \chi_k^t(a_i) \tag{5}$$

C6: If an extra edge is traversed, the cost will increase. This is accomplished by the following constraint for each extra edge $(u_j^t, u_k^{t+1}) \in F_i$

$$\mathcal{E}_{j,k}^t(a_i) \Rightarrow \mathcal{C}_{j,k}^t(a_i) \tag{6}$$

C7: The cardinality constraint. This bounds the total cost, by ensuring that no more than Δ extra edges are traversed. The following cardinality constraint ensures this:

$$\leq_\Delta \left\{ \mathcal{C}_{j,k}^t(a_i) | i = 1, 2, ..., n \wedge t = 0, 1, ..., \mu - 1 \wedge (u_j^t, u_k^{t+1}) \in F_i \right\} \tag{7}$$

So the majority of the constraints remain the same as proposed in the paper, with the two changes being that the constraint that an agent must occupy exactly one vertex at every timestep replaces the constraint that dictates that an agent must take exactly one edge every timestep. Because there are fewer vertices than edges, the encoding is smaller, resulting in a faster runtime. Furthermore, the constraint that ensures that an edge's target is empty was removed, and constraint 3 was added to ensure that no edge conflicts occur. This results in a sum-of-costs optimal SAT encoding for MAPF without vertex and edge conflicts.

4.2. The waypoint encoding

Extending this encoding to the waypoint variant of MAPF is rather trivial. First, the heuristic of agents must be calculated as the shortest path from the starting vertex to the goal vertex that also visits all waypoints. This can be done optimally by using a traveling salesman problem (TSP) solver. Furthermore, an additional constraint is added that ensures that an agent visits a waypoint at a timestep between $t = 0$ and $t = m$. Thus for each agent $a_i \in A$ and $w_j \in W_i$

$$\bigvee_{t \in \{1,2,...,\mu-1\}} \chi_j^t(a_i) \tag{8}$$

The original algorithm will produce an optimal solution to MAPFW problem instances since this constraint ensures that an agent will visit all of its waypoints and the optimal makespan bound can still be found using TSP. Of course, TSP is an NP-hard complete problem, and thus the runtime scales exponentially with the number of waypoints, but a good algorithm can solve up to 18 waypoints in a

couple of seconds. Normally, this is more than used in benchmarks and real-life instances, but if there is a need for more waypoints then a heuristic algorithm can also be used. This results in faster runtimes for the calculations of the heuristics but will lead to a sub-optimal solution and thus a higher makespan bound. This causes a larger encoding than the optimal makespan bound since the MDDs will contain more edges and vertices. Furthermore, a sub-optimal makespan bound will yield a sub-optimal solution for the problem, and thus an optimal solver for TSP is recommended and used in this study. Using this new encoding in algorithm 2 will result in an algorithm for MAPFW.

4.3. The colored encoding

Creating an encoding for the colored MAPF problem will be done by first calculating the individual heuristics for each agent. Because it is unclear which agent will go to which target in this problem variant, the individual heuristics will be calculated per team while ignoring the agents of the other teams. A bipartite graph will be created to calculate the heuristic by connecting all of the starting locations of the agents on this team to the target location they can reach [47]. The edge value will be the distance between the two vertices. This problem can then be solved by a modified Hungarian algorithm in $O(n^3)$ time. This provides the optimal sum-of-costs heuristic for a team, but it does not necessarily give the minimum makespan since a different solution with the same sum-of-costs but with a different makespan may exist. As a result, the Hungarian algorithm needs to be run again, but with a limit of the edge cost in the bipartite graph. Starting at 1 and increasing it until a solution is found with the same sum-of-costs as the optimal solution yields the minimum makespan under the optimal sum-of-cost. The pseudocode for this algorithm can be found at Algorithm 3.

Algorithm 3 Calculating the minimum makespan for colored MAPF

```
Given team j and empty graph G
for s_i ∈ j do
    for g_i ∈ j do
        dist ← dist(s_i, g_i)
        add ((s_i, g_i), dist) to G
sum-of-costs ← Hungarian(G)
makespan ← 1
while Solution not found do
    for s_i ∈ j do
        for g_i ∈ j do
            dist ← dist(s_i, g_i)
            if dist <makespan then
                add ((s_i, g_i), dist) to G_2
    temp ← Hungarian(G_2)
    if temp = makespan then
        break
    makespan + = 1
```

It is impossible to track when an individual agent makes an extra move because multiple configurations can have the same sum-of-cost. Therefore, the cost variable used in the previously introduced encodings is discarded. Instead, a variable is introduced that tracks the number of waiting moves on the target made by all agents. A variable to track the number of edges taken by all agents that add a cost is also possible but this leads to a large encoding. Furthermore, waiting on the target adds no cost and is therefore related to the number of edges taken by all agents that do add a cost. Given that $\mu = \mu_0$, $\Delta = 0$, and ξ is the sum of all team heuristics, it follows that $n * \mu$ edges are taken in total. ξ of those must be taken since this is the optimal sum-of-cost. Because $\Delta = 0$, $(n * \mu) - \xi$ edges must be taken where an agent waits on its target vertex. If this is not a feasible solution, both μ and Δ will be increased by one. This means that one additional step can be taken or that there are $n - 1$ additional waiting steps. So the number of waiting steps is bound by $(n * \mu) - (\xi + \Delta)$, and this is a lower bound since the goal is to maximize the number of waiting steps.

The sixth and seventh constraints of the Basic Encoding will be replaced by two new constraints and a new cardinality constraint. It should be noted that this could have also been done per team,

but doing so would have added a lot of cardinality constraints, so it was decided to create a larger one for all agents instead. Before introducing the new constraints, a new set WE is defined as a set that contains all the edges where an agent waits on its target location, or more formally as $(u_j^t, u_k^{t+1}) \in V_i$ where $j = k$ and $j \in g_a$ where g_a is the team of agent a goal vertices. This is used in the new variable $\mathcal{W}_{j,k}^t(a_i)$ for every $t \in 0, 1, ..., \mu - 1$ and $u_j^t, u_k^{t+1} \in WE$, which tracks the total number of waiting moves made. Using this, the new constraints that replace the basic encoding's sixth and seventh constraints are:

C6: If an agent waits at the target vertex, the number of waiting moves will be increased. This is done by the following constraint for every edge $(u_j^t, u_k^{t+1}) \in WE$:

$$\mathcal{W}_{j,k}^t(a_i) \Rightarrow \mathcal{E}_{j,k}^t(a_i) \tag{9}$$

C7: If an agent waits at the target vertex, it can not move away from that vertex. Thus, once a waiting move has been made the agent can not make any other moves. So for every edge $(u_j^t, u_k^{t+1}) \in WE$:

$$\mathcal{W}_{j,k}^t(a_i) \Rightarrow \bigwedge_{t2=t,t+1,...,\mu} \chi_j^{t2}(a_i) \tag{10}$$

C8: The cardinality constraint. This bounds the total number of waiting moves. The bound β is given by $(n * \mu) - (\Delta + \xi_0)$.

$$\geq_\beta \left\{\mathcal{W}_{j,k}^t(a_i) | i = 1, 2, ..., n \wedge t = 0, 1, ..., \mu - 1\right\} \tag{11}$$

Once again algorithm 2 can be used if the new encoding is used for the solver and algorithm 3 is used to calculate μ_0.

4.4. MaxSAT Encoding

This section will present a new method to solve MAPF, MAPFW, and colored MAPF problem instances, where MaxSAT will be used as the solver instead of SAT. The cardinality constraint will therefore be discarded and the cost or waiting variables will be the only variables with an assigned weight that the MaxSAT solver will try to optimize. This means that a solution will be found for a smaller or equal Δ than using SAT since it finds a solution for a given makespan while minimizing the sum of cost and does not have to satisfy the cardinality constraint. This method is therefore also not optimal under the sum-of-costs objective but finds the best sum-of-costs solution given the optimal makespan. This is because μ_0 starts as the optimal makespan and is then used to find a solution with the lowest sum-of-cost. If a solution is found, it certainly has the optimal makespan and lowest sum-of-costs given this optimal makespan. If no solution is found this means that there is no solution for this makespan and thus increasing it will eventually result in the correct solution.

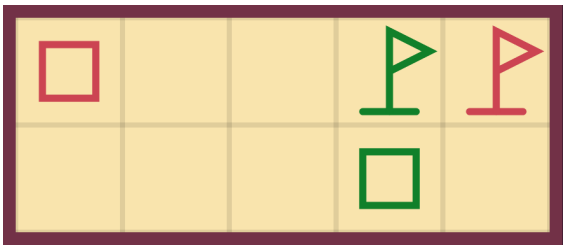

Figure 4.1: An example of a problem to showcase the difference between SAT and MaxSAT.

4.4.1. The difference between SAT and MaxSAT

The difference between the two methods will be demonstrated using the simple MAPF problem in Figure 4.1. The individual heuristics are 1 for green and 4 for red and thus the optimal makespan μ_0 is 4 and Δ starts at 0. Now, for SAT, red must take the shortest path since it must arrive at the goal at timestep 4 and the distance is exactly 4 steps. Meanwhile, green can only reach its goal at the final timestep because moving away from it and then returning to let red pass would be considered extra

steps because its individual heuristic is 1. Thus Δ will be increased by 1, but this has no effect because red still can only take the shortest path of 4 and green still requires two extra steps. Thus, when $\Delta = 2$, this solution is finally satisfiable with a cost of 8 since both move 4 times. But, red can now take a different path as well that goes around the green's target. Then red takes 6 steps and green only takes 1 before waiting on its target leading to a solution with a sum-of-costs of 7 that is indeed optimal. MaxSAT on the other hand does not need to increase Δ because it finds the solution with makespan 4 and sum-of-costs 8 in which red takes its optimal path and green waits until red has passed.

Although MaxSAT technically solves the problem for a different objective function, it still finds good sum-of-costs solutions and it finds the optimal sum-of-costs solution in cases where μ_0 is the optimal makespan. Furthermore, using SAT Δ gets increased more often, particularly in problems with many collisions which adds a lot of extra runtime since the encoding needs to be generated every time and the solver needs to check whether it is satisfiable. MaxSAT, on the other hand, will find solutions for a much lower Δ and should therefore find its solution much faster. For these reasons, comparing the runtime and quality of the solution of MaxSAT to SAT would be interesting.

4.4.2. Finding an upperbound

Currently, μ_0 is used as the upper bound for the solver, but using a larger upper bound will yield better sum-of-costs solutions and in some cases even optimal ones. Finding the upper bound that will result in an optimal sum-of-costs without any calculations seems impossible. In the example given in figure 4.1 it must be increased by two but this example can be made even more complex, leading to the belief that no universal rule can be created. However, using the sum-of-costs of the optimal makespan solution as the upper bound will result in the optimal sum-of-costs solution. Given the earlier example from Figure 4.1, where MaxSAT finds a solution with a sum-of-costs of 8, it is clear that 8 is an upper bound for the optimal sum-of-costs solution and that using this as an upper bound will lead to the optimal sum-of-costs solution because no agent needs to make more actions since this will lead to a higher sum-of-costs solution. This method of computing the optimal sum-of-costs solution is impractical because such a large upper bound results in a large encoding that takes much longer to solve. Of course, there may be better methods of findings a smaller upper bound for the optimal sum-of-costs solution, but that was not the focus of this research. In the experiments, MaxSAT will be used with both the upper bound being μ_0, as well as an inflated version where the upper bound is $\mu_0 * 1.25$. The latter is used to see how much better the solutions are with a higher upper bound compared to the expected runtime loss.

5

Experimental Evaluation

The main goal of the experiments is to compare the introduced SAT algorithm and MaxSAT algorithm on different graph types to see which algorithm performs better and if the graph types influence that. As discussed earlier, it is expected that MaxSAT will find solutions faster but not always the sum-of-costs optimal solution, and thus both the number of instances solved by the methods as well as the quality of the solutions will be compared. To further improve this comparison inflated MaxSAT is also used where the starting upper bound μ_0 is increased by 25% and this is expected to lie between the two. Furthermore, the search-based algorithm M* is also used in all comparisons, although it is not a state-of-the-art solver it performs adequately and is consistent for all the variants. M* is expected to perform better at larger maps with longer path lengths because of its search, but these cases are also more difficult for (Max)SAT since larger maps and longer paths lead to a larger encoding and thus a longer solving time. For colored MAPF, prematching is also used for some experiments. Prematching turns the colored MAPF instance into normal MAPF instances for all possible configurations, and although there are some optimizations available it is expected that after a certain number of agents it drops off since generating all pairings takes an exponential amount of time.

5.1. The graph types

The experiments will be done on 4-connected grid graphs of varying sizes with random obstacles and a 32x32 rooms graph [44]. These are both standard setups for testing MAPF algorithms, with 4-connected grid graphs being used in many comparisons. For these experiments, grid sizes of 8x8, 16x16, and 32x32 are used with 10% of the nodes being an obstacle. An 8x8 and 32x32 grid will cover both a compact map without a lot of room to maneuver agents and a map with more open spaces. The 32x32 rooms will be used since this combines the narrow corridors property with the isolated rooms, and these smaller maps should be more interesting and comparable to real-life problem instances. Therefore the commonly used Dragon Age maps which are a collection of large maps that include maps with a lot of open space, narrow corridors, and isolated rooms will not be used in these experiments. Experiments also include a new type of graph that resembles the train shunting problem called a carrousel graph [30]. A carrousel graph is a star graph where every branch is a line graph and then there are a few extra edges between the individual branches. A simple example can be found in figure 5.2.

5.2. The experimental setup

For colored MAPF the experiments will be divided into two parts. The first part is where all of the agents are on the same team, also known as the anonymous MAPF problem. This is an interesting subvariant because there are algorithms that solve it in polynomial time[62]. As a result, it will be interesting to see how SAT performs on it. The other part is colored MAPF where the agents are spread over 3 teams. In the MAPFW experiments, all agents will have a set of 3 waypoints they must visit. Lastly, there will be some experiments on the combined problem called colored MAPFW, where the agents are spread over 3 teams and each agent has 3 waypoints.

All experiments had a 3-minute timeout and were performed on an Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz with 8GB of memory, and using the Glucose 3 solver for SAT and RC2 for MaxSAT.

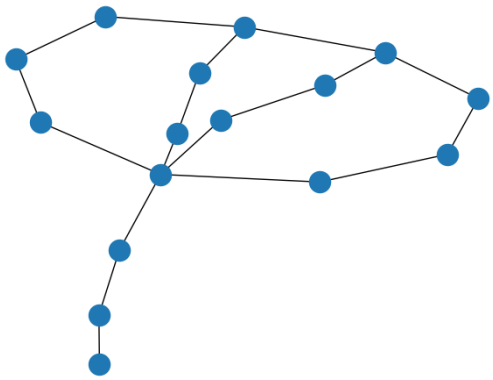

Figure 5.1: An example of a carrousel graph

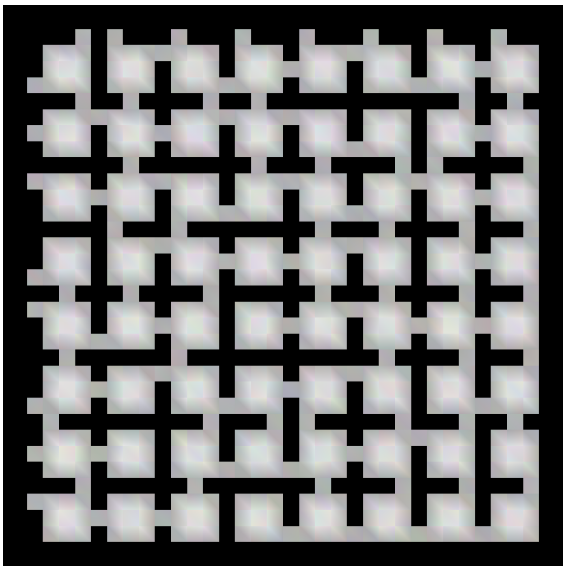

Figure 5.2: The 32x32 rooms graph.
Source: https://movingai.com/benchmarks/mapf/index.html

5.3. Discussion

The results for the colored MAPF can be seen in Figure A.1 for the 3 team problem instances and Figure A.2 for the 1 team or anonymous MAPF problem instances. This includes the 8x8 grid, 32x32 grid, the carrousel graph, and the rooms graph. The same graphs are used for MAPFW and can be found in Figure A.3. For the colored MAPFW problem, there is only the 8x8 grid, the 16x16 grid, and the carrousel graph and these results can be found in Figure A.4. Although every graph has a legend the same algorithms will have the same color throughout all the graphs. Table 5.1 shows the percentage that the found solution of the MaxSAT algorithms was optimal per individual experiment.

5.3.1. Colored MAPF

From both Figure A.1 and Figure A.2 it is clear that the SAT and MaxSAT approaches always outperform M* by a significant margin and that the in-matching approaches always outperform the prematching approaches. This was expected and thus only SAT, MaxSAT, and inflated MaxSAT will be compared. Therefore the prematching MaxSAT solutions are also not in Table 5.1. For the 3 team problem instances both MaxSAT approaches outperform the SAT approach for the number of agents it can solve problems for, yet in cases where both approaches find a solution, MaxSAT finds a solution with a cost higher than the optimal cost at a maximum of 20% of the cases. For the inflated approach this drops down to an upper limit of 6% yet it also solves fewer instances than the MaxSAT approach.

As discussed earlier, increasing the minimum makespan causes the encoding to become much larger and thus the solver needs more time, but it can find better solutions. Lastly, the 32x32 rooms problem seems more difficult than the 32x32 grid with random obstacles, which is due to the fact that the rooms graph is laid out to have chokepoints, which causes collisions.

Interestingly, for the anonymous MAPF problem where all agents are on the same team, SAT performs better than the MaxSAT approaches. This is likely because there are so many configurations possible that the optimal one used for the heuristics is not the configuration of the optimal solution. As a result, MaxSAT spends a lot of time looking for a solution with that minimal makespan that does not exist, while SAT realizes that this is not feasible and keeps increasing the makespan until it does find a feasible solution. MaxSAT does find solutions with the optimal cost in almost all cases but with the lesser instances solved it loses its main advantage. It is also worth noting that this variant can be solved in polynomial time and that SAT and MaxSAT solve the 8x8 grid for almost all instances. The problem is trivial with a low number of agents, but it also becomes increasingly easier the closer the number of agents gets to the number of nodes because if there are N nodes with N agents then all agents can stay on their starting vertex. For the larger map sizes, it does eventually drop off since the map size does attribute to the runtime even if it would be a polynomial algorithm it could take more time than the 3-minute timeout. Whether or not these approaches are a polynomial approach for this problem will not be proved in this research.

	MaxSAT	inflated
8x8, 1 team	99%	100%
8x8, 3 teams	79%	94%
8x8, 3 waypoints	81%	92%
8x8, combined	69%	99%
32x32, 1 team	100%	100%
32x32, 3 teams	95%	99%
32x32, 3 waypoints	100%	100%
16x16, combined	94%	100%
carrousel, 1 team	98%	100%
carrousel, 3 teams	80%	97%
carrousel, 3 waypoints	80%	100%
carrousel, combined	79%	100%
rooms, 1 team	100%	100%
rooms, 3 teams	89%	95%
room, 3 waypoints	100%	100%

Table 5.1: The percentage that the found solution of the MaxSAT algorithms was optimal per individual experiment.

5.3.2. MAPFW

MAPFW is a much more difficult problem than colored MAPF because it can be solved for significantly fewer agents. This is due to the fact that by adding the waypoints the paths of all agents become much longer, thus increasing the makespan which influences the size of the encoding. Because the makespan is now much higher the MDDs of every agent will include more nodes because it still only considers all the paths from the starting location to the target location with that length. It does not consider only the paths that also visit all the waypoints since this is incompatible with the way MDDs are efficiently generated and thus represents a different type of problem that can be addressed in future research. So, when you consider that there are far more timesteps to consider and that each timestep contains far more nodes and edges, it is easy to see that the MAPFW problem is far more difficult for SAT than the colored MAPF problem. Once again MaxSAT generally outperforms SAT with an upper limit of 20% on finding a worse solution. However, on the larger graphs, even M* has similar or even better results, because the increase of the makespan increases the runtime of SAT so significantly while M* can use its search to find a good path through the waypoints.

5.3.3. colored MAPFW

As expected the combined problem is more difficult than the individual problems. This is likely due to the increase of the makespan from the MAPFW problem, which, as discussed in the previous section, is the more difficult variant of the two. But this is more prevalent in larger problem instances since more instances get solved on the 8x8 grid than for the MAPFW 8x8 grid instances. While for the 32x32 grid and the rooms graphs very few instances were solved and those were therefore omitted, and a 16x16 graph was used instead. Running these with a larger timeout would likely improve the results since it takes much more time to build the encodings and solve them, but now all the timeouts are consistent. The results are expected with MaxSAT solving more instances than SAT, but the upper limit on instances with a higher cost is also increased to 30%.

6

Conclusion

In this paper, an extension to the existing SAT solver for the sum-of-cost MAPF problem which solves the waypoint and colored variants of MAPF was introduced. Furthermore, a new encoding that uses a MaxSAT solver and which solves MAPF and the two aforementioned variants for makespan while optimizing the sum-of-costs was developed. These methods were compared to each other and the existing search-based M* algorithm for problem instances of colored MAPF, MAPFW, and the introduced colored MAPFW problem. The colored MAPF experiments also included the prematching SAT algorithm. These experiments were done on 4-connected grids, a rooms graph, and the introduced carrousel graph. From the results, it was concluded that the SAT and MaxSAT algorithms perform much better than M* especially on the smaller instances. The larger instances are more difficult for the SAT and MaxSAT since it takes time to set up the entire encoding, while M* can use its search to its advantage. This is also why MAPFW instances are more difficult to solve than colored MAPF problem instances because the path length with MAPFW is longer and therefore the encoding becomes much larger as well. MaxSAT generally outperforms SAT but it does not always find the optimal solution. Yet, SAT is comparable to MaxSAT in the colored MAPF problem instances where all agents are on the same team. Lastly, the combined problem is harder than the individual problems the larger the instances become but can be reasonable well solved in the 3-minute timeout for smaller graphs.

7

Future work

Nevertheless, many improvements could have been made to this paper's research. To begin with, many improvements to the sum-of-costs SAT algorithm were introduced in the related works section but could not be applied to the algorithms introduced in this paper. Especially the combination with CBS into SMT-CBS seems like a good improvement. CBS or even better CBSH2 or CBSH-MP should have also been used in the experiments for a more accurate comparison to a search-based solver instead of M*. Lastly, improvements upon the experiments could also be achieved by having a larger timeout, especially for the colored MAPFW problem instances. There were also some intriguing questions raised during this paper that merit further research. Such as can the MDD structure for the MAPFW problem be improved so it only contains the paths that go through all the waypoints. This should reduce the size of the MDDs thus making the encoding much smaller as well. However, this does not seem like a trivial problem and the increase in the results may not be that significant. Another one is whether the introduced methods are polynomial in the case of anonymous MAPF. Finally, although it may appear to be impractical, can an upper bound for the MaxSAT algorithm be calculated such that it finds a sum-of-costs optimal solution.

References

[1] Robbin Baauw. "Adapting CBM to optimize the Sum of Costs". Bachelor's Thesis. 2021.

[2] Olivier Bailleux and Yacine Boufkhad. "Efficient CNF Encoding of Boolean Cardinality Constraints". In: Principles and Practice of Constraint Programming – CP 2003. Ed. by Francesca Rossi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 108–122. ISBN: 978-3-540-45193-8.

[3] Roman Barták, Marika Ivanová, and Jiří Švancara. "Colored Multi-Agent Path Finding: Solving Approaches". In: The International FLAIRS Conference Proceedings 34 (Apr. 2021). DOI: 10.32473/flairs.v34i1.128535. URL: https://journals.flvc.org/FLAIRS/article/view/128535.

[4] Eli Boyarski et al. "ICBS: Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding". In: Proceedings of the 24th International Conference on Artificial Intelligence. IJCAI'15. Buenos Aires, Argentina: AAAI Press, 2015, pp. 740–746. ISBN: 9781577357384.

[5] Eli Boyrasky et al. "Don't Split, Try To Work It Out: Bypassing Conflicts in Multi-Agent Pathfinding". In: Proceedings of the International Conference on Automated Planning and Scheduling 25.1 (Apr. 2015), pp. 47–51. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/13725.

[6] Ivar de Bruin. "Solving Multi-Agent Pathfinding with Matching using A*+ ID+ OD". Bachelor's Thesis. 2021.

[7] Martin Capek and Pavel Surynek. "DPLL(MAPF): an Integration of Multi-Agent Path Finding and SAT Solving Technologies". In: SOCS. 2021.

[8] Martin Davis, George Logemann, and Donald Loveland. "A Machine Program for Theorem-Proving". In: Commun. ACM 5.7 (July 1962), pp. 394–397. ISSN: 0001-0782. DOI: 10.1145/368273.368557. URL: https://doi.org/10.1145/368273.368557.

[9] Jeroen van Dijk. "Solving the Multi-Agent Path Finding with Waypoints Problem Using Subdimensional Expansion". Bachelor's Thesis. 2020.

[10] Jonathan Dönszelmann. "Matching in Multi-Agent Pathfinding using M". Bachelor's Thesis. 2021.

[11] Esra Erdem et al. "A General Formal Framework for Pathfinding Problems with Multiple Agents". In: Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence. AAAI'13. Bellevue, Washington: AAAI Press, 2013, pp. 290–296.

[12] Ariel Felner et al. "Adding Heuristics to Conflict-Based Search for Multi-Agent Path Finding". In: Proceedings of the International Conference on Automated Planning and Scheduling 28.1 (June 2018), pp. 83–87. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/13883.

[13] Cornelia Ferner, Glenn Wagner, and Howie Choset. "ODrM* optimal multirobot path planning in low dimensional search spaces". In: 2013 IEEE International Conference on Robotics and Automation. 2013, pp. 3854–3859. DOI: 10.1109/ICRA.2013.6631119.

[14] Arjen Ferwerda. "Extending the Multi-Label A* Algorithm for Multi-Agent Pathfinding with Multiple Waypoints". Bachelor's Thesis. 2020.

[15] Graeme Gange, Daniel Harabor, and Peter J. Stuckey. "Lazy CBS: Implicit Conflict-Based Search Using Lazy Clause Generation". In: Proceedings of the International Conference on Automated Planning and Scheduling 29.1 (May 2021), pp. 155–162. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/3471.

[16] A. Goldberg and R. Tarjan. "Solving Minimum-Cost Flow Problems by Successive Approximation". In: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing. STOC '87. New York, New York, USA: Association for Computing Machinery, 1987, pp. 7–18. ISBN: 0897912217. DOI: 10.1145/28395.28397. URL: https://doi.org/10.1145/28395.28397.

[17] Meir Goldenberg et al. "Enhanced Partial Expansion A*". In: J. Artif. Int. Res. 50.1 (May 2014), pp. 141–187. ISSN: 1076-9757.

[18] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. In: IEEE Transactions on Systems Science and Cybernetics 4.2 (1968), pp. 100–107. DOI: 10.1109/TSSC.1968.300136.

[19] Noah Jadoenathmisier. “Extending CBS to efficiently solve MAPFW”. Bachelor’s Thesis. 2020.

[20] Jaap de Jong. “Multi-Agent Pathfinding with Matching using Enhanced Partial Expansion A”. Bachelor’s Thesis. 2021.

[21] Guy Katz et al. “Lazy Proofs for DPLL(T)-Based SMT Solvers”. In: Proceedings of the 16th Conference on Formal Methods in Computer-Aided Design. FMCAD ’16. Mountain View, California: FMCAD Inc, 2016, pp. 93–100. ISBN: 9780983567868.

[22] Jiaoyang Li et al. “Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search”. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 442–449. DOI: 10.24963/ijcai.2019/63. URL: https://doi.org/10.24963/ijcai.2019/63.

[23] Jiaoyang Li et al. “New Techniques for Pairwise Symmetry Breaking in Multi-Agent Path Finding”. In: Proceedings of the International Conference on Automated Planning and Scheduling 30.1 (June 2020), pp. 193–201. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/6661.

[24] Jiaoyang Li et al. “Symmetry-Breaking Constraints for Grid-Based Multi-Agent Path Finding”. In: Proceedings of the AAAI Conference on Artificial Intelligence 33.01 (July 2019), pp. 6087–6095. DOI: 10.1609/aaai.v33i01.33016087. URL: https://ojs.aaai.org/index.php/AAAI/article/view/4565.

[25] Hang Ma and Sven Koenig. “Optimal Target Assignment and Path Finding for Teams of Agents”. In: Proceedings of the 2016 International Conference on Autonomous Agents Multiagent Systems. AAMAS ’16. Singapore, Singapore: International Foundation for Autonomous Agents and Multiagent Systems, 2016, pp. 1144–1152. ISBN: 9781450342391.

[26] J.P. Marques-Silva and K.A. Sakallah. “GRASP: a search algorithm for propositional satisfiability”. In: IEEE Transactions on Computers 48.5 (1999), pp. 506–521. DOI: 10.1109/12.769433.

[27] Joao Marques-Silva and Inês Lynce. “Towards Robust CNF Encodings of Cardinality Constraints”. In: Principles and Practice of Constraint Programming – CP 2007. Ed. by Christian Bessière. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 483–497. ISBN: 978-3-540-74970-7.

[28] Andor Michels. “Multi-agent pathfinding with waypoints using Branch-Price-and-Cut”. Bachelor’s Thesis. 2020.

[29] Robert Morris et al. “Planning, scheduling and monitoring for airport surface operations”. In: Workshops at the Thirtieth AAAI Conference on Artificial Intelligence. 2016.

[30] Jesse Mulderij et al. “Train Unit Shunting and Servicing: a Real-Life Application of Multi-Agent Path Finding”. In: CoRR abs/2006.10422 (2020). arXiv: 2006.10422. URL: https://arxiv.org/abs/2006.10422.

[31] Bernhard Nebel. “On the Computational Complexity of Multi-Agent Pathfinding on Directed Graphs”. In: CoRR abs/1911.04871 (2019). arXiv: 1911.04871. URL: http://arxiv.org/abs/1911.04871.

[32] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. “Solving SAT and SAT Modulo Theories: From an Abstract Davis–Putnam–Logemann–Loveland Procedure to DPLL(T)”. In: J. ACM 53.6 (Nov. 2006), pp. 937–977. ISSN: 0004-5411. DOI: 10.1145/1217856.1217859. URL: https://doi.org/10.1145/1217856.1217859.

[33] Zhongqiang Ren, Sivakumar Rathinam, and Howie Choset. “MS: A New Exact Algorithm for Multi-agent Simultaneous Multi-goal Sequencing and Path Finding”. In: 2021 IEEE International Conference on Robotics and Automation (ICRA). 2021, pp. 11560–11565. DOI: 10.1109/ICRA48506.2021.9561779.

[34] Guni Sharon et al. “Conflict-based search for optimal multi-agent pathfinding”. In: Artificial Intelligence 219 (2015), pp. 40–66. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2014.11.006. URL: https://www.sciencedirect.com/science/article/pii/S0004370214001386.

[35] Guni Sharon et al. “Conflict-based search for optimal multi-agent pathfinding”. In: Artificial Intelligence 219 (2015), pp. 40–66. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2014.11.006. URL: https://www.sciencedirect.com/science/article/pii/S0004370214001386.

[36] Guni Sharon et al. “The increasing cost tree search for optimal multi-agent pathfinding”. In: Artificial Intelligence 195 (2013), pp. 470–495. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2012.11.006. URL: https://www.sciencedirect.com/science/article/pii/S0004370212001543.

[37] Stef Siekman. “Extending A* to solve multi-agent pathfinding problems with waypoints”. Bachelor’s Thesis. 2020.

[38] David Silver. “Cooperative pathfinding”. In: Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment. Vol. 1. 1. 2005, pp. 117–122.

[39] Carsten Sinz. “Towards an Optimal CNF Encoding of Boolean Cardinality Constraints”. In: Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming. CP’05. Sitges, Spain, 2005, pp. 827–831. ISBN: 9783540292388. DOI: 10.1007/11564751_73. URL: https://doi.org/10.1007/11564751_73.

[40] Kiril Solovey and Dan Halperin. “K-Color Multi-Robot Motion Planning”. In: Int. J. Rob. Res. 33.1 (Jan. 2014), pp. 82–97. ISSN: 0278-3649. DOI: 10.1177/0278364913506268. URL: https://doi.org/10.1177/0278364913506268.

[41] Arvind Srinivasan et al. “Algorithms for discrete function manipulation”. In: 1990 IEEE international conference on computer-aided design. IEEE Computer Society. 1990, pp. 92–93.

[42] Trevor Standley. “Finding Optimal Solutions to Cooperative Pathfinding Problems”. In: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence. AAAI’10. Atlanta, Georgia: AAAI Press, 2010, pp. 173–178.

[43] Roni Stern et al. “Multi-agent pathfinding: Definitions, variants, and benchmarks”. In: Twelfth Annual Symposium on Combinatorial Search. 2019.

[44] Nathan R. Sturtevant. “Benchmarks for Grid-Based Pathfinding”. In: IEEE Transactions on Computational Intelligence and AI in Games 4.2 (2012), pp. 144–148. DOI: 10.1109/TCIAIG.2012.2197681.

[45] Pavel Surynek. “A Simple Approach to Solving Cooperative Path-Finding as Propositional Satisfiability Works Well”. In: Dec. 2014, pp. 827–833. ISBN: 978-3-319-13559-5. DOI: 10.1007/978-3-319-13560-1_66.

[46] Pavel Surynek. “An Optimization Variant of Multi-Robot Path Planning Is Intractable”. In: Proceedings of the AAAI Conference on Artificial Intelligence 24.1 (July 2010), pp. 1261–1263. URL: https://ojs.aaai.org/index.php/AAAI/article/view/7767.

[47] Pavel Surynek. “Compact Representations of Cooperative Path-Finding as SAT Based on Matchings in Bipartite Graphs”. In: Nov. 2014, pp. 875–882. DOI: 10.1109/ICTAI.2014.134.

[48] Pavel Surynek. “Multi-goal multi-agent path finding via decoupled and integrated goal vertex ordering”. In: Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 35. 14. 2021, pp. 12409–12417.

[49] Pavel Surynek. “Mutex reasoning in cooperative path finding modeled as propositional satisfiability”. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems. 2013, pp. 4326–4331. DOI: 10.1109/IROS.2013.6696977.

[50] Pavel Surynek. “Optimal cooperative path-finding with generalized goals in difficult cases”. In: Proceedings of the 10th Symposium on Abstraction, Reformulation, and Approximation, SARA 2013 (Jan. 2013), pp. 119–122.

[51] Pavel Surynek. “Sparsification for Fast Optimal Multi-Robot Path Planning in Lazy Compilation Schemes”. In: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2021), pp. 7931–7938.

[52] Pavel Surynek. “Towards Optimal Cooperative Path Planning in Hard Setups through Satisfiability Solving”. In: Proceedings of the 12th Pacific Rim International Conference on Trends in Artificial Intelligence. PRICAI’12. Kuching, Malaysia: Springer-Verlag, 2012, pp. 564–576. ISBN: 9783642326943. DOI: 10.1007/978-3-642-32695-0_50. URL: https://doi.org/10.1007/978-3-642-32695-0_50.

[53] Pavel Surynek. “Unifying Search-based and Compilation-based Approaches to Multi-agent Path Finding through Satisfiability Modulo Theories”. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 1177–1183. DOI: 10.24963/ijcai.2019/164. URL: https://doi.org/10.24963/ijcai.2019/164.

[54] Pavel Surynek et al. “Efficient SAT Approach to Multi-Agent Path Finding under the Sum of Costs Objective”. In: Proceedings of the Twenty-Second European Conference on Artificial Intelligence. ECAI’16. The Hague, The Netherlands: IOS Press, 2016, pp. 810–818. ISBN: 9781614996712. DOI: 10.3233/978-1-61499-672-9-810. URL: https://doi.org/10.3233/978-1-61499-672-9-810.

[55] Pavel Surynek et al. “Integration of Independence Detection into SAT-based Optimal Multi-Agent Path Finding - A Novel SAT-based Optimal MAPF Solver”. In: Jan. 2017, pp. 85–95. DOI: 10.5220/0006126000850095.

[56] Pavel Surynek et al. “Migrating Techniques from Search-based Multi-Agent Path Finding Solvers to SAT-based Approach”. In: Journal of Artificial Intelligence Research 73 (2022), pp. 553–618.

[57] Pavel Surynek et al. “Variants of Independence Detection in SAT-Based Optimal Multi-agent Path Finding”. In: Agents and Artificial Intelligence. Ed. by Jaap van den Herik, Ana Paula Rocha, and Joaquim Filipe. Cham: Springer International Publishing, 2018, pp. 116–136. ISBN: 978-3-319-93581-2.

[58] Glenn Wagner and Howie Choset. “M*: A complete multirobot path planning algorithm with performance bounds”. In: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems. 2011, pp. 3260–3267. DOI: 10.1109/IROS.2011.6095022.

[59] Peter R Wurman, Raffaello D’Andrea, and Mick Mountz. “Coordinating hundreds of cooperative, autonomous vehicles in warehouses”. In: AI magazine 29.1 (2008), pp. 9–9.

[60] Takayuki Yoshizumi, Teruhisa Miura, and Toru Ishida. “A* with Partial Expansion for Large Branching Factor Problems”. In: Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence. AAAI Press, 2000, pp. 923–929. ISBN: 0262511126.

[61] Jingjin Yu and Steven LaValle. “Planning Optimal Paths for Multiple Robots on Graphs”. In: Proceedings - IEEE International Conference on Robotics and Automation (Apr. 2012). DOI: 10.1109/ICRA.2013.6631084.

[62] Jingjin Yu and Steven M. LaValle. “Multi-agent Path Planning and Network Flow”. In: Algorithmic Foundations of Robotics X. Ed. by Emilio Frazzoli et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 157–173. ISBN: 978-3-642-36279-8.

[63] Jingjin Yu and Steven M. LaValle. “Optimal Multirobot Path Planning on Graphs: Complete Algorithms and Effective Heuristics”. In: IEEE Transactions on Robotics 32.5 (2016), pp. 1163–1177. DOI: 10.1109/TRO.2016.2593448.

[64] Jingjin Yu and Steven M. LaValle. “Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs”. In: Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence. AAAI’13. Bellevue, Washington: AAAI Press, 2013, pp. 1443–1449.

A

Results

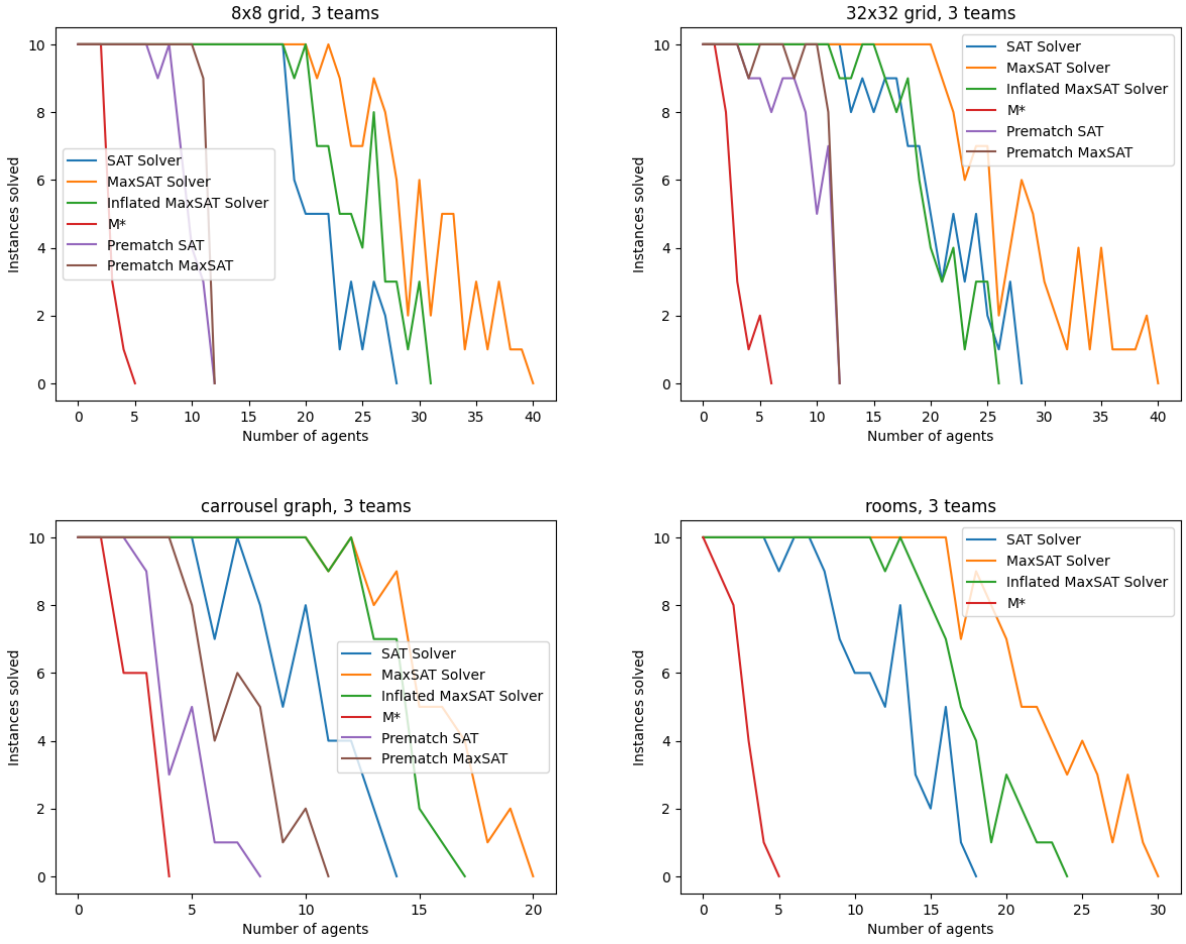

Figure A.1: Number of instances solved for colored MAPF with 3 teams

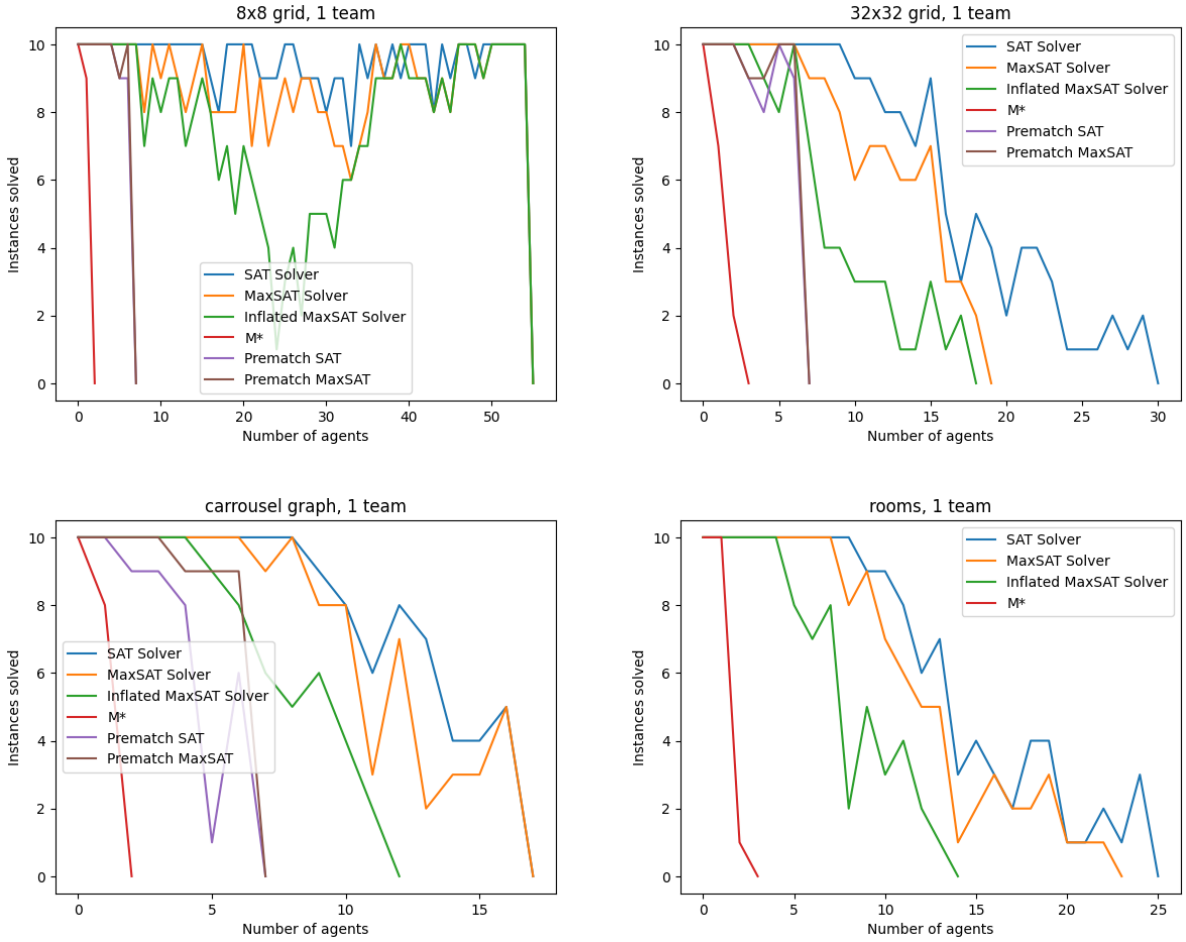

Figure A.2: Number of instances solved for colored MAPF with 1 team (anonymous MAPF)

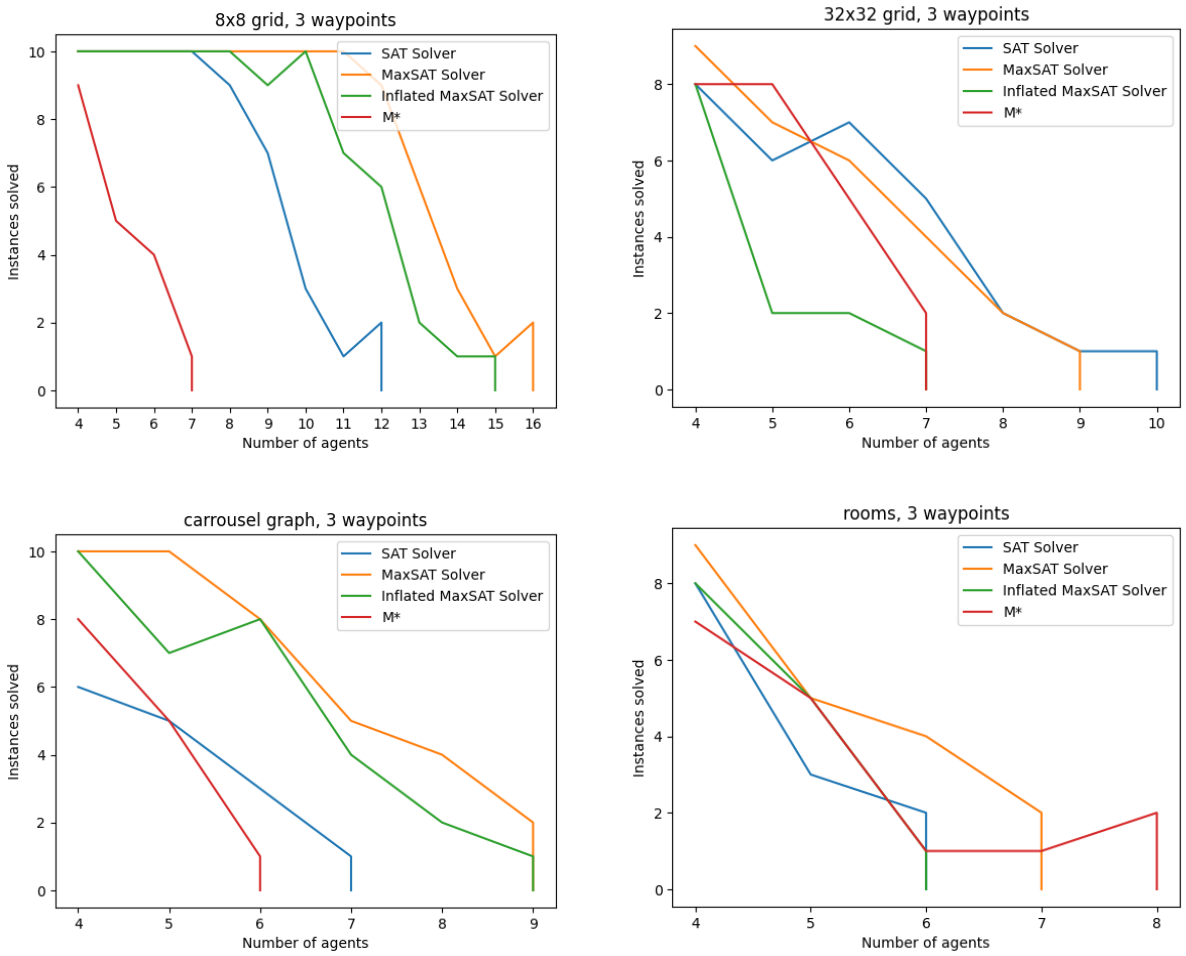

Figure A.3: Number of instances solved for MAPFW with 3 waypoints

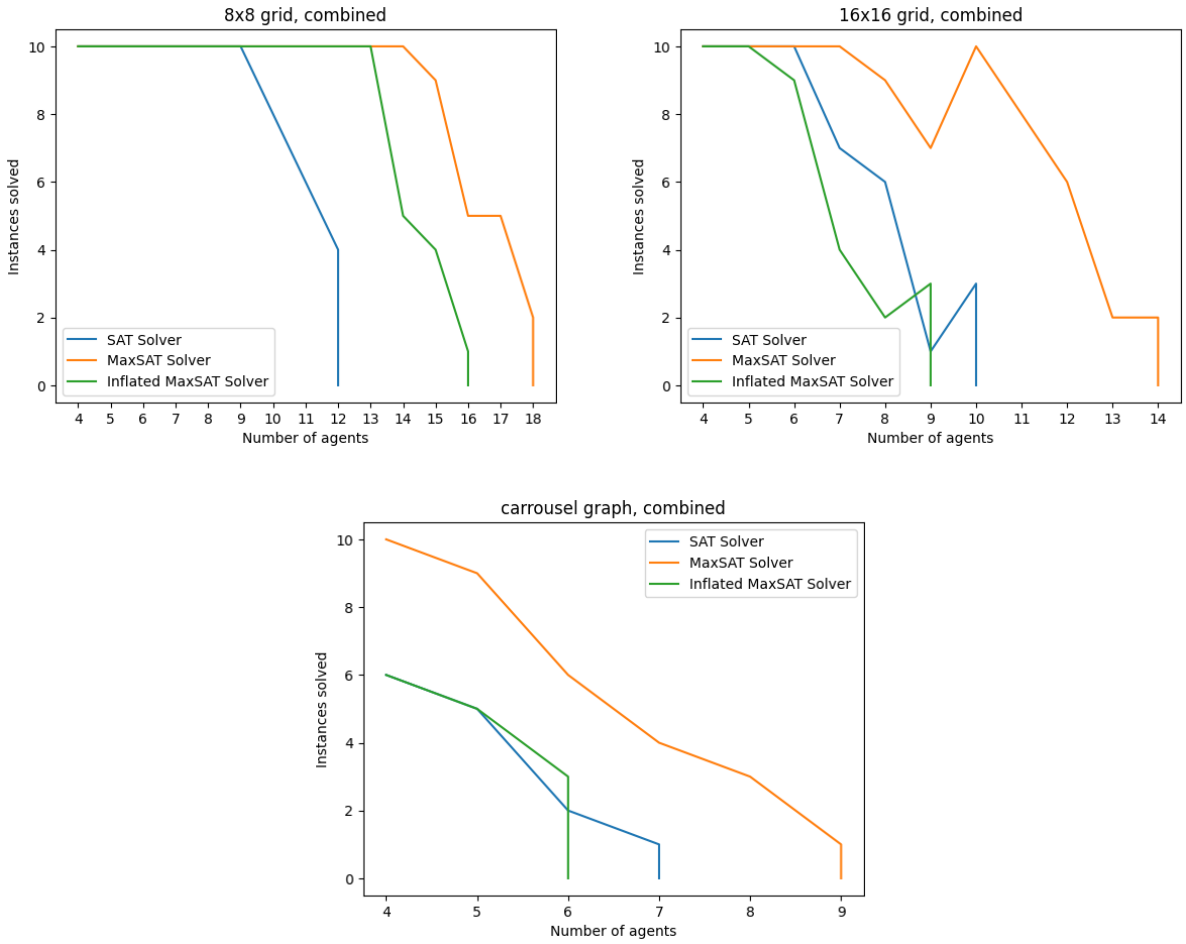

Figure A.4: Number of instances solved for colored MAPFW with 3 waypoints and 3 teams