Modelling Multi-Agent Pathfinding Problems by Integrating Connectivity and No-Collision Constraints

Anonymous submission

Abstract

Multi-agent pathfinding (MAPF) is the problem of finding collision-free paths for a set of agents in a shared environment, typically represented as a graph. One of the approaches to solving MAPF represents the problem as a Boolean satisfiability problem. However, due to the encoding required to represent the valid paths, this method can produce extremely large Boolean formulas, both in terms of variables and clauses. Herein, we propose two encodings of MAPF designed for SAT Modulo Theories solvers. Our approach delegates all the valid path reasoning to a monotonic theory supporting source-target connectivity. This is then combined with a 2-SAT Boolean formula to prevent collisions between agents. Together, these components create an effective separation of concerns: the SAT solver focuses on resolving conflicts, while the theory solver handles the connectivity constraints. Our experiments are conducted in both makespan and sum of costs optimisation settings, empirically demonstrating a notable reduction in both the size of the MAPF encoding and the time required to generate it. In addition, when fixing the SAT solver across experiments, results demonstrate considerable performance improvements when transitioning from pure SAT to our proposed SMT encodings.

Introduction

Multi-agent pathfinding (MAPF) is the task of navigating a set of mobile agents in a shared environment from their starting location to desired goal locations while avoiding collisions (Stern et al. 2019). It has many practical applications in warehousing (Wurman, D'Andrea, and Mountz 2008), airplane taxiing (Morris et al. 2016), video game control (Silver 2005), and traffic junctions (Dresner and Stone 2008).

Finding an optimal MAPF solution has been shown to be computationally hard for a wide range of cost objectives. Instead of creating domain-specific optimal solvers, a popular approach in theoretical computer science is to translate hard problems into a well-established formalism and use an existing, highly optimized solver for that formalism. There is a wide range of formalisms to choose from; the most notable mentions include Boolean Satisfiability (SAT) (Biere et al. 2021), Answer Set Programming (ASP) (Baral 2003), Mixed-Integer Programming (MIP) (Schrijver 1999), and Constraint Programming (CP) (Dechter 2003). Many previous works have translated MAPF into an instance in a different formalism (Surynek 2022). Some of these works have given exhaustive comparisons between translations to different formalisms (Achá et al. 2021; Barták et al. 2017), highlighting the strengths and weaknesses of each. These approaches must all translate the two main constraints of the MAPF problem: (i) path existence, ensuring valid paths exist from start to goal for each agent, and (ii) collision avoidance, ensuring agents do not interfere with each other. Solving via translation to SAT is a popular approach with clear strengths, especially in densely populated environments (Surynek 2016). It can also easily leverage many years of research effort in engineering efficient SAT solvers, as well as benefit from their future improvements.

In translating MAPF into SAT, one of the most challenging tasks is to encode graph-related constraints such as connectivity[1]. To address this, the work presented herein does not encode the whole problem in SAT. We instead leverage MonoSAT (Bayless et al. 2015), a SAT Modulo Theories (SMT) solver for monotonic theories, which provides graph propagators. We make these propagators responsible for the constraints on the existence of valid source-target paths, while the collision avoidance constraints are still modelled by a Boolean formula passed to the SAT solver.

Contributions

The main contributions of this work are as follows. We introduce SMT encodings with dedicated graph propagators to model MAPF. These delegate the complex connectivity constraints to a graph theory solver while efficiently expressing collision avoidance as 2-SAT formulas. We implement this approach using two well-known encodings: *Pass* variables, which track agent movement through vertices and edges, and *Shift* variables, which represent relative movements between time-steps. These encodings are evaluated on two standard cost functions: makespan and sum of costs. Finally, we provide a comprehensive empirical evaluation comparing all combinations of encoding strategies and cost functions across a diverse MAPF benchmark set, offering insights on how each combination affects both formula size and solving efficiency. The results highlight the usefulness of the approach, paving the way for future research.

[1] Sometimes *reachability* is used in the literature instead. In our work, we reserve the term reachability for reachability of a location in a given time-step and will use *connectivity* instead. In either case, it relates to an existence of a path between given s and t.

Background and definitions

An instance of *multi-agent pathfinding (MAPF)* is a pair $\mathcal{M} = (G, A)$, where $G = (V, E)$ is a graph representing the shared environment and A is a set of n agents, each represented by a start and goal location, i.e., $a_i = (s_i, g_i)$. Time is considered discrete and at each time-step each agent can wait in its current location or move to one neighbouring.

The task is to find a plan π_i for each agent a_i. A plan is a sequence of locations the agent occupies at each time-step, where $\pi_i(t) = v$ represents that agent a_i is located in vertex v at time-step t. A plan π_i is valid if it is a path from the agent's start location to its goal location, i.e., $\pi_i(1) = s_i$ and $\pi_i(|\pi_i|) = g_i$, and for every $t = (1, \dots |\pi_i| - 1)$ either $\pi_i(t) = \pi_i(t + 1)$ or $(\pi_i(t), \pi_i(t + 1)) \in E$. Furthermore, there can be no *vertex conflicts*, i.e., no two agents can be present at the same vertex at the same time, and no *swapping conflicts*, i.e., no two agents can move over the same edge in the opposite direction simultaneously. The solution of MAPF is the collision-free *joint plan* $\Pi = \bigcup_{a_i \in A} \pi_i$.

In addition to finding a valid solution, it may be required to find a solution that is optimal. In the literature, two cost functions are often used to describe the quality of the found solution – *makespan* (Surynek 2014) and *sum of costs* (Sharon et al. 2011). Let T_i^* denote the last time-step agent a_i arrives at its goal g_i. Then the makespan of plan Π is defined as $mks(\Pi) = \max_{a_i \in A} T_i^*$, and its sum of costs is defined as $soc(\Pi) = \sum_{a_i \in A} T_i^*$. Intuitively, the makespan is the first time all of the agents are present in their respective goal locations, and the sum of costs is the sum of time-steps until all of the agents are in their goal locations and are not required to move again. After reaching the goal location, the agent remains in the environment and is able to move out of the goal if required to make way for some other agent. However, this agent contributes to the cost until it arrives at the goal for the last time and does not move again. Note that optimizing either makespan or sum of costs is an NP-Hard problem (Yu and LaValle 2013; Surynek 2015).

Connectivity Reasoning in SAT solvers

Encoding connectivity in SAT is a significant challenge, as it is fundamentally a transitive property that requires reasoning about paths in graphs. Therefore, it does not naturally map to the Boolean logic structure of SAT. Various works over the years (Rankooh and Rintanen 2022) have dealt with the challenging problem of encoding variants of connectivity into standard propositional clauses. These encodings would need to be further generalised to consider the various additional MAPF requirements, such as for nodes to be revisited by agents or the self-loops introduced by wait actions.

An alternative approach is to introduce a propagator that is able to deal with these concerns efficiently. A *propagator* in the context of a SAT solver is responsible for analysing the current partial assignment against a set of constraints during the search. Its primary function is to deduce further variable assignments that are logically entailed by the current state, and to identify a conflict if the current assignment violates these constraints. For directed graphs translated to SAT, Gebser, Janhunen, and Rintanen (2014) introduced a propagator for acyclicity, which they show can be combined with a graph encoding to model *s-t* connectivity efficiently. An expansion of this idea is MonoSAT (Bayless et al. 2015), an SMT solver for monotonic theories. It supports not only acyclicity, but also a wide set of graph predicates, such as *s-t* connectivity, shortest paths, or maximum flow.

Encoding

In this section, SAT-based encodings to solve the MAPF problem are described. First, we describe known encodings that create purely CNF formulas. We then demonstrate how a significant fraction of the clauses from these encodings can be delegated to the graph theories of the SMT solver.

Pure SAT encoding

To model the position of each agent in time, Boolean variables $At(a_i, v, t)$ are created for each triple *(agent, vertex, time-step)*. Constraints on the agents' movements for a specified cost are expressed as a CNF formula, forwarded to an underlying SAT solver. For a satisfiable formula, the assignment of *At* variables corresponds to the plan of each agent. An important improvement in reduction-based MAPF solving was the introduction of preprocessing of reachable positions (Sharon et al. 2011). When examining the distance between the start location s_i, any vertex v, the goal location g_i, and the current bound on the cost, it can be reasoned whether the vertex v can be reached by the specific agent a_i or not due to the distance and the cost limitation. If a vertex cannot be visited, neither the corresponding variable nor the corresponding constraints need be created. We will use this reachability preprocessing in all of the following encodings.

Variables

Pass encoding (P) While MAPF may be modelled using only the *At* variables, additional variables describing movement are used to decrease the size and number of some constraints. Boolean variables $Pass(a_i, u, v, t)$ are added, modelling the movement of an agent a_i over an edge (u, v) starting in time-step t. *Self-loop* edges (v, v) are added to allow waiting. Using *At* and *Pass* variables, the following constraints are created to model MAPF (Barták and Švancara 2019). Note that the SAT solver requires CNF, but we provide the constraints with implications for perspicuity. For the same reason, we leave out the quantification of agents and vertices when it is clear from context. All of the constraints are created for all agents in all positions reachable at that time-step. The per-agent end time T_i depends on the used cost function, as explained in Cost Functions below.

$$At(a_i, s_i, 1) \tag{1}$$

$$At(a_i, g_i, T_i) \tag{2}$$

$$\forall t \in \{1, \dots, T_i\}, \forall u, v \in V, u \neq v : \\ \neg At(a_i, u, t) \vee \neg At(a_i, v, t) \tag{3}$$

$$\forall t \in \{1, \dots, T_i - 1\} : \\ At(a_i, u, t) \implies \bigvee_{(u,v) \in E} Pass(a_i, u, v, t) \tag{4}$$

$$\forall t \in \{1, \ldots, T_i - 1\}:$$
$$\mathit{Pass}(a_i, u, v, t) \implies \mathit{At}(a_i, v, t+1) \quad (5)$$
$$\forall a_i, a_j \in A, a_i \neq a_j, \forall t \in \{1, \ldots, \min(T_i, T_j)\}:$$
$$\neg \mathit{At}(a_i, v, t) \vee \neg \mathit{At}(a_j, v, t) \quad (6)$$
$$\forall a_i, a_j \in A, \forall t \in \{1, \ldots, \min(T_i, T_j) - 1\}, \forall (u, v) \in E:$$
$$\neg \mathit{Pass}(a_i, u, v, t) \vee \neg \mathit{Pass}(a_j, v, u, t) \quad (7)$$

Unit clauses (1) and (2) ensure that each agent starts and ends in the correct location. Constraint (3) ensures that each agent is located in only one vertex per time-step. Constraints (4) and (5) ensure correct movement of each agent. First, an agent must leave through exactly one neighbouring edge. Second, when moving over an edge, the agent must arrive at the correct vertex in the next time-step. The last two constraints explicitly forbid the conflicts between agents – (6) forbids vertex conflict and (7) forbids swapping conflicts.

Shift encoding (S) The drawback of *Pass* variables is that they are created per agent. A more recent encoding (Achá et al. 2022) uses Boolean variables $\mathit{Shift}(u, v, t)$ modelling the movement of *something* over an edge (u, v) in a time-step t. Additional constraints are added to map the movement to a specific agent. Using *At* and *Shift* variables, the following constraints are created to model the MAPF problem. Again, we leave out the quantification of agents and vertices when it is clear from context. All the constraints are created for all agents in all positions reachable at that time-step in the per-agent end time T_i. Constraints (1), (2), (3), and (6) are shared with the *Pass* encoding. In addition:

$$\forall t \in \{1, \ldots, T_i - 1\}:$$
$$\mathit{At}(a_i, u, t) \implies \bigvee_{(u,v) \in E} \mathit{At}(a_i, v, t+1) \quad (8)$$
$$\forall t \in \{1, \ldots, T_i - 1\}:$$
$$(\mathit{Shift}(u, v, t) \wedge \mathit{At}(a_i, u, t)) \implies \mathit{At}(a_i, v, t+1) \quad (9)$$
$$\forall t \in \{1, \ldots, T_i - 1\}:$$
$$(\mathit{At}(a_i, u, t) \wedge \mathit{At}(a_i, v, t+1)) \implies \mathit{Shift}(u, v, t) \quad (10)$$
$$\forall t \in \{1, \ldots, \max_{a_i \in A}(T_i) - 1\}, \forall (u, v), (u, w) \in E:$$
$$\neg \mathit{Shift}(u, v, t) \vee \neg \mathit{Shift}(u, w, t) \quad (11)$$
$$\forall t \in \{1, \ldots, \max_{a_i \in A}(T_i) - 1\}, \forall (u, v) \in E:$$
$$\neg \mathit{Shift}(u, v, t) \vee \neg \mathit{Shift}(v, u, t) \quad (12)$$

Constraints (8)–(11) describe correct movement and connect *Shift* variables to specific agents: An agent can move only to a neighbouring vertex; If an agent is located in a vertex and at the same time there is something moving from the vertex, the agent is relocated by this movement; If an agent is located in a vertex and at the next time-step it is located in the neighbouring vertex, there must have been a movement between these vertices; All movement from one vertex sums up to at most one in a single time-step. Lastly, constraint (12) forbids swapping conflicts. Reachable *At* preprocessing remains the same, but $\mathit{Shift}(u, v, t)$ is not created only if no agent can traverse edge (u, v) at time-step t.

Cost Functions

Makespan All of the presented constraints relate to a per-agent end time T_i. To model a decision problem of finding a solution in makespan T, T_i for each agent is set to T. Specifically, the latest time of arrival to the goal for each agent in constraint (2) is set to T. Modelling makespan is therefore straightforward as it corresponds to the created *At* variables.

To optimize the makespan, T is increased by one until a solvable formula is produced. To save numerous computations, a lower bound on the makespan of $LB(mks) = \max_{a_i \in A} \mathit{dist}(s_i, g_i)$ is the first T used, where $\mathit{dist}(s_i, g_i)$ is the distance between s_i and g_i.

Sum of costs Optimising the sum of costs means minimising the time-step of arrival to the goal location per agent, via the minimisation of the sum of those time-steps. The latest time-step of arrival to goal T_i is set separately for each agent and depends on the shortest path from s_i to g_i. We start with setting the limit on the sum of costs as a lower bound $LB(soc) = \sum_{a_i \in A} \mathit{dist}(s_i, g_i)$. Let δ denote the number of increments to the cost limit (initially $\delta = 0$). The δ represents an extra allowed movement shared by all agents. Then each agent a_i needs to be present at their goal at time-step $T_i = \mathit{dist}(s_i, g_i) + \delta$, as there is certainly not enough allowed movement to arrive later. While it is possible that all of the extra movement is consumed by a single agent, all of the extra movement of all agents must sum to δ.

Recall that an agent is allowed to move from its goal after reaching it, but it incurs additional cost until the last return to the goal. Therefore, limiting the sum of costs is not as simple as limiting the time-steps spent outside of the goal location. We create additional Boolean variables $\mathit{Late}(a_i, t)$ to indicate whether an agent a_i is late at time-step t (i.e. it has not arrived at its goal location for the last time). For an agent a_i, the *Late* variables are created only for time-steps $\mathit{dist}(s_i, g_i), \ldots, \mathit{dist}(s_i, g_i) + \delta$, as before that time-step, the agent is not late ($\mathit{dist}(s_i, g_i)$ time-steps are certainly needed), and after that time-step, the agent has to be in the goal (due to the preprocessing, the agent cannot be located anywhere else, also no other agent can be located in g_i from $\mathit{dist}(s_i, g_i) + \delta$ onward). The following formula ensures that if an agent is not in the goal location, it is late.

$$\neg \mathit{At}(a_i, g_i, t) \implies \mathit{Late}(a_i, t) \quad (13)$$

If an agent is not late at a certain time, it cannot be late in the future. This information is propagated in time, to ensure that any waiting in goal is counted as being late if the agent leaves at any time-step in the future.

$$\neg \mathit{Late}(a_i, t) \implies \neg \mathit{Late}(a_i, t+1) \quad (14)$$

To enforce that at most δ movement is spent, at most δ *Late* variables among all agents are allowed to be set to *True*. This can be achieved in numerous ways in SAT through *at-most-k* constraint (Philipp and Steinke 2015).

In case of an unsatisfiable formula, δ and each T_i are increased by one. It was proven that this process provides a sum of costs optimal solution upon reaching the first satisfiable formula (Surynek et al. 2016).

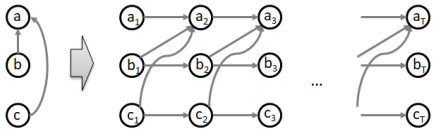

Figure 1: An example of a graph on 3 vertices being transformed into a time-expanded graph with T layers (i.e. T time-steps). Figure taken from (Barták and Švancara 2019).

Encoding with Graph Theories

Irrespective of the cost function, the number of time-steps is increased after an unsuccessful call to the SAT solver. This process is represented by a time-expanded graph (TEG), consisting of a number of copies of the original graph G, each forming a layer in the TEG and corresponding to a time-step. Vertex v in layer t is denoted with a superscript, v^t. If there is an edge (u, v) in G, a directed edge from vertex u^t to vertex v^{t+1} exists in the TEG for all time-steps t. 'waiting' edges are also created that connect copies of vertices in subsequent layers. Figure 1 presents an example TEG. Each agent moves over the TEG from layer to layer, always moving forward in time. In the preceding encodings, a TEG is not explicitly created. Rather, it serves as a graphical representation of each agent's position over time.

Variables

Pass+Graph encoding (PG) The SMT solver with graph theories allows the user to define one or more graphs and associate each vertex and edge to a Boolean variable[2]. The associated variable is set to `True` iff the given vertex (respectively edge) is present in the graph. To model the movement of the agents, we create a TEG for each agent a_i with T_i layers, TEG_i. The vertices and edges in the TEGs are associated with the *At* and *Pass* variables, respectively, thus the same set of variables is created as in the pure SAT encoding.

The SMT solver allows the user to add any number of connectivity constraints between two vertices and associate them with Boolean variables. The associated variable is set to `True` iff there is a path between the two specified vertices. We add a connectivity constraint for each agent a_i between vertices s_i^1 and $g_i^{T_i}$ in TEG_i. Each such constraint is associated with a fresh Boolean variable. These variables are forced to be assigned to `True` via unit clauses: there must be a path in the TEG for each agent from their start to goal location. This connectivity constraint replaces constraints (1), (2), (4), and (5). Indeed, the path starts and ends in the correct location. The latter two clauses force the connection to be a path. Constraint (3) forbidding duplication of agents is not needed. If an agent is duplicated in the TEG, *phantom agents* may appear. However, it is impossible for those to produce an invalid solution, as the connectivity constraints ensure at least one valid connected path. If some other variables are set to `True`, but not connected to both s_i^0 and $g_i^{T_i}$, they may be filtered out by simple post-processing (simple DFS to find the connected path). Figure 2 presents an example of two TEGs with highlighted possible solution.

[2]In fact, only edges are associated with Boolean variables. In our implementation, each vertex is split into two vertices connected by an edge, which is associated to the variable. For simplicity, we say that vertices can be associated with variables as well.

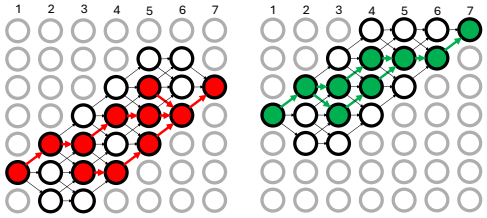

Figure 2: A solution by *Pass+Graph* encoding for an instance with seven vertices and two agents. The two TEGs have seven layers corresponding to time-steps $\{1, \ldots, 7\}$. Unreachable vertices are greyed out and the corresponding edges are not present. Coloured vertices and edges correspond to *At* and *Pass* variables set to `True`. The only reachable vertices in layers 1 and 7 are the start and goal, respectively. The solution for the red agent contains vertices that do not create a valid path from start to goal. These can be filtered out by simple post-processing. For the green agent, there are two possible valid paths. Either is valid as it cannot conflict with the red agent and has the same cost.

To ensure that there are no collisions between the agents, the constraints forbidding the vertex (6) and swapping conflicts (7) are kept. Note that both of those are binary clauses.

Together, the *Pass+Graph* encoding consists of $|A|$ number of TEGs associated with variables *At* and *Pass* (using the same preprocessing as in pure SAT encoding *Pass*), $|A|$ number of unit clauses for the connectivity constraints, and constraints (6) and (7) forbidding conflicts.

Shift+Graph encoding (SG) Creating a separate TEG for every single agent may seem wasteful, as some parts of the different TEG_i represent the same location, but are occupied by different agents. We draw inspiration from the *Shift* encoding and create a single TEG shared by all agents, that is TEG_+. Each edge in the TEG_+ is associated with a corresponding *Shift* variable. Note that in this case, the vertices are *not* associated with the *At* variables. The *At* variables copy the location for each agent, which we try to avoid. Rather, we navigate all agents through the same TEG and make sure that their paths do not cross. The SMT solver allows for multiple connectivity constraints over the same graph. For each agent a_i, the connectivity constraint from s_i^1 to $g_i^{T_i}$ is created. Again, this connection is forced by creating a unit clause. To ensure that the paths of the agents do not cross, it is forced that there is at most one outgoing *Shift* via clause (11) and at most one incoming *Shift* via clause (15).

$$\forall t \in \{1, \ldots, \max_{a_i \in A}(T_i) - 1\}, \forall (u, v), (w, v) \in E : \\ \neg \mathit{Shift}(u, v, t) \vee \neg \mathit{Shift}(w, v, t) \tag{15}$$

By enforcing non-overlapping paths in the TEG_+, vertex conflicts are implicitly forbidden. Indeed, each path corresponds to a single agent and no two paths can enter the

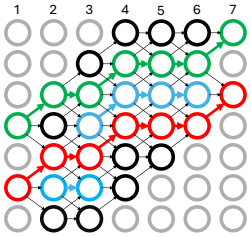

Figure 3: Example as in Fig. 2 solved by *Shift+Graph* encoding. Coloured edges correspond to the *Shift* variables set to `True`. There are no variables for the vertices; we highlight them to show the paths clearly. The only reachable positions in layers 1 and 7 are the starts and goals, respectively. Different colours are again present to improve readability. Red and green represent valid paths of the two agents, and blue edges represent phantom agents. Phantom agents do not connect any start and goal and no two paths can intersect.

same vertex in the TEG_+. To forbid swapping conflicts, clauses (12) are added as in the pure SAT encoding.

From a single *Shift* variable set to `True` it is unclear which agent uses this movement. To extract the solution from the variable assignment, a simple DFS from each s_i^1 is run to find a valid path to the corresponding $g_i^{T_i}$. While no path can branch and no two paths can merge, there is still a possibility of *phantom agents* as some *Shift* variables may be set to `True` while not being connected to any start or goal vertex in the 1-st and T_i-th layer, respectively. Again, these are inconsequential, as the DFS post-processing does not depend on them. Figure 3 presents an example.

The *Shift+Graph* encoding thus consists of one TEG associated with only variables *Shift* (with the same preprocessing as the pure SAT encoding), $|A|$ unit clauses for the connectivity constraints, clauses (11) and (15) modelling non-overlapping paths, and (12) forbidding swapping conflicts.

Cost Functions

Makespan Deciding the existence of a solution with makespan T is straightforward, as it corresponds to the number of created layers in the TEG. The connectivity requirements are then set from s_i^1 to g_i^T for each agent a_i. The preprocessing on the initial number of layers and the increment scheme is the same as with the pure SAT encoding. Correctness follows from the correctness of the pure SAT encoding.

Sum of costs The individual latest time-step of arrival to goal T_i is as the pure SAT encoding: $T_i = \mathit{dist}(s_i, g_i) + \delta$. The limit on the sum of costs is again enforced by creating $\mathit{Late}(a_i, t)$ variables and consequently at most δ *Late* variables are allowed to be `True`. Propagation of being late is again done by formula (14). However, deciding if an agent is late in a given time-step is not as straightforward. Recall that in both *Graph* encodings, there may be *phantom agents* setting any variable $\mathit{At}(a_i, g_i, t)$ (respectively $\mathit{Shift}(g_i, g_i, t)$) to *True*, even though these variables are not the ones connecting the start and goal location. Therefore, we cannot be certain that an agent is in the goal: we need to enforce that the agent is not anywhere else other than in the goal.

Using the *Pass+Graph* encoding, the following formula ensures that if an agent is located anywhere other than at its goal, it is late in that time-step. The formula is created for all time-steps $t \in \{\mathit{dist}(s_i, g_i), \ldots, \mathit{dist}(s_i, g_i) + \delta\}$.

$$\forall v \in V, v \neq g_i : \mathit{At}(a_i, v, t) \implies \mathit{Late}(a_i, t) \qquad (16)$$

Enforcing that each agent is not duplicated using constraint (3) would ensure that constraint (13) can be used instead of (16). However, (3) creates many more clauses compared to (13). Nevertheless, both options will be evaluated.

Using the *Shift+Graph* encoding, we want to reason about the agent not being in a goal without the *At* variables. Due to the connectivity requirement, if the formula is satisfiable, there is a connected path from s_i^1 to $g_i^{T_i}$, where $T_i = \mathit{dist}(s_i, g_i) + \delta$. Such a path has to use some variable $\mathit{Shift}(v, g_i, t)$ to move the agent into g_i from some other vertex v. Whenever such movement happens, the agent is late in the time-step t, which is expressed in the following formula. The formula is created for all time-steps $t \in \{\mathit{dist}(s_i, g_i), \ldots, \mathit{dist}(s_i, g_i) + \delta\}$.

$$\forall v \in V, v \neq g_i : \mathit{Shift}(v, g_i, t) \implies \mathit{Late}(a_i, t) \qquad (17)$$

Eliminating *phantom agents* to simplify (17) requires limiting the number of *Shift* variables set to `True` in each layer. As this requires several *at-most-k* constraints, expensive in SAT, we omit such encoding from the experiments.

In both encodings – *Pass+Graph* and *Shift+Graph* – the preprocessing ensures that no other agent a_j interferes with the goal location of a_i, as the variables $\mathit{At}(a_j, g_i, t)$, respectively $\mathit{Shift}(v, g_i, t)$ are not created for $t \geq \mathit{dist}(s_i, g_i) + \delta$.

Empirical evaluation

The instances in our experiments are inspired by a commonly used benchmark set available online (Stern et al. 2019). Generally, the most important attributes contributing to solving speed are map size, placement of obstacles on the map, and number of agents. Other attributes can affect performance, such as interaction between agents, congestion, etc. However, these are hard to quantify and generate automatically. Therefore, we vary the size of the 4-connected grid maps, starting with the size of 20×20 and increasing by 20 until reaching 100×100. Next, we change the layout of the obstacles in the map, producing *empty*, *random*, and *warehouse* maps (Figure 4 (a) and (b)). Additionally, we test large game maps *brc202d*, *den520d*, and *ost003d* (Figure 4 (c) – (e)). Lastly, a set of agents is created (a scenario). The intended way to use the benchmark set is to create an instance of MAPF from a map and a number of agents from a

(a) *random* (b) *warehouse* (c) *brc202d* (d) *den520d* (e) *ost003d*

Figure 4: Maps used in the experiments, omitting *empty*.

		20	40	60	80	100	brc	den	ost	total
mks	*P*	380	77	16	4	1	0	0	0	478
	S	402	133	25	4	1	0	0	0	565
	PG	392	**353**	**227**	**131**	**91**	**1**	**4**	**6**	**1205**
	PG3	**424**	156	33	10	2	0	0	0	625
	SG	207	227	172	113	71	0	3	5	798
soc	*P*	123	207	215	271	236	29	68	47	1196
	S	139	220	224	273	235	25	64	47	1227
	PG	132	**238**	**290**	**324**	**347**	**36**	**77**	**56**	**1500**
	PG3	130	224	257	297	274	31	73	51	1337
	SG	**140**	188	173	202	187	20	47	36	993

Table 1: The number of solved instances by each encoding using the MonoSAT solver. The results are split by map size and by the optimized cost function. The best performing encoding for each size and cost function is highlighted.

		20		40		60		80		100	
mks	*P*	3.2	0.9	87.7	7.5	419.7	16.3	2559.1	57.4	7318.5	105.6
	S	3.4	0.2	89.8	1.9	425.2	4.1	2575.3	14.0	7348.9	26.4
	PG	0	0.9	0	7.5	0	16.3	0	57.4	0	105.6
	PG3	2.9	0.9	85.7	7.5	414.5	16.3	2543.5	57.4	7289.4	105.6
	SG	0.2	0.01	1.3	0.09	3.9	0.2	10.3	0.7	19.4	1.4
soc	*P*	0.005	ϵ	0.02	ϵ	0.05	ϵ	0.1	ϵ	0.2	ϵ
	S	0.008	ϵ	0.03	ϵ	0.07	ϵ	0.2	ϵ	0.3	ϵ
	PG	0	ϵ	0	ϵ	0	ϵ	0	ϵ	0	ϵ
	PG3	0.002	ϵ	0.01	ϵ	0.03	ϵ	0.1	ϵ	0.2	ϵ
	SG	0.002	ϵ	0.006	ϵ	0.01	ϵ	0.02	ϵ	0.03	ϵ

Table 2: Average number of clauses (in millions) for instances with a fixed number of 30 agents. The results are split by map size and by the optimized cost function. The two provided numbers correspond to movement clauses on the left and conflict clauses on the right. The unit clauses are omitted, as there is too few of them in all cases. A value of ϵ is used for less than a thousand.

scenario. If the instance is solved within the given time limit, additional agents from the same scenario are added, and thus a new MAPF instance is produced. Once the instance cannot be solved within the time limit, we can reasonably assume that further increasing the number of agents cannot make the instance solvable. We start with 5 agents, adding 5 more each time the instance is solvable within the given time limit. For each map, we create 5 different scenario files with randomly placed start and goal locations.

The time limit per instance is 5 minutes using a single CPU and a maximum of 64GB of memory. Experiments were performed on a cluster using AMD EPYC 7543 @ 2.80GHz. Creation of the CNF formula is implemented in C++[3]. We use MonoSAT (Bayless et al. 2015), an SMT solver with graph propagator. The PBlib (Philipp and Steinke 2015) library is used to create cardinality constraints for the sum of costs optimal models. The cardinality constraint is fixed for all encodings: comparing different cardinality constraints is beyond the scope of this paper. We do not compare our novel encoding to different optimal MAPF algorithms (such as the popular search-based algorithm CBS (Sharon et al. 2015)), since it was shown that different algorithm paradigms excel at different types of instances (Kaduri, Boyarski, and Stern 2020). Instead, we compare with the state-of-the-art SAT encoding.

Results

Table 1 presents number of solved instances by encoding. The difference in results for makespan and sum of costs agrees with a previous study (Švancara et al. 2024): optimising makespan is more successful on smaller instances but drastically falls off as the size of the map increases, which does not affect sum of costs to the same extent. This can be explained by the increase in formula size presented in Table 2, where the number of clauses for makespan optimization is several orders of magnitude larger.

The *Pass+Graph* encoding is the most successful in most cases. For the smallest maps, the difference is not as significant and *Pass+Graph+(3)* [4] and *Shift+Graph* are the most successful by a tiny margin. Comparing the *Graph* encodings with their pure SAT counterparts, both the *Pass+Graph* and *Pass+Graph+(3)* are always able to solve more instances. The difference is more significant for *Pass+Graph* and for makespan optimisation as the size of the map grows. *Pass+Graph* solved more than double the number of instances than *Pass* in makespan optimisation. For sum of costs, it managed to solve about a quarter more instances, which is still a significant improvement. The advantage of *Pass+Graph+(3)* quickly falls off as the size of the map increases, which can be explained by the number of created clauses of type (3) (Table 2). The number of clauses created by constraint (3) almost matches the whole formula created by the pure SAT encoding *Pass*. This is because the other movement constraints grow linearly, while (3) grows quadratically in the number of vertices. The benefit of forbidding *phantom agents* and simplifying the constraints on sum of costs is outweighed by the number of clauses. For a fixed $\delta = 10$, the number of clauses to limit sum of costs using constraint (13) is 4601, while using (16) is around 16500 (this number is approximate as it depends on the reachable positions), which is a minuscule growth compared with (3).

The most surprising result is the underperformance of the *Shift+Graph* encoding compared with *Pass+Graph*, although *Shift* does perform better than *Pass*. The number of movement clauses is not as much reduced for *Shift+Graph* compared with *Pass+Graph*, but it is reduced compared with *Shift*. In addition, the number of variables is reduced (recall that *At* variables are not created). Yet, the pure SAT encoding *Shift* solves more instances when optimizing the sum of

[3] [Link to repository in case of acceptance]

[4] A *Graph* encoding that in addition explicitly forbids *phantom agents* via constraint (3). *PG*3 for short.

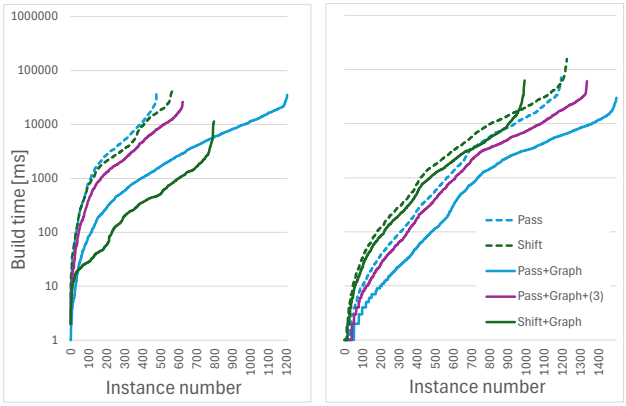

Figure 5: The building time in *ms* of all solved instances by each of the presented encodings optimizing makespan (left) and sum of costs (right) objective. On the *x*-axis is the number of the instance. The instances are sorted by their build time. The *y*-axis is in logarithmic scale.

costs, and the improvement of *Shift+Graph* when optimizing makespan is not great compared with the improvement of other *Graph* encodings. Our current conjecture is that graph reasoning over a single TEG is difficult when the paths must be disjoint. Further investigation, with a better understanding of the inner workings of MonoSAT, is required.

Figure 5 shows the building time of all solved instances by each presented encoding while optimizing makespan and sum of costs. While the number of clauses presented in Table 2 correlates with building time, the TEGs required by the *Graph* encodings also require some time to be built. However, TEG building time depends on the number of variables, not the number of clauses, which is usually smaller. Figure 5 (left) shows that the building time while optimizing makespan for all *Graph* encodings is smaller than that of the pure SAT encodings, as expected. For sum of costs, in Figure 5 (right), the building time of *Shift+Graph* is closer to the build times of the pure SAT encodings, which corresponds to the lower performance than the other *Graph* encodings. Creation of the Boolean variables is also included in the building time. Lastly, we highlight that a faster building time does not mean that the solver is guaranteed to be more successful, as the smaller created formula might be more challenging for the underlying solver. However, lower building time allows a larger proportion of the time limit to be spent on solving.

So far, we have fixed the underlying solver, but MonoSAT is not updated with the most recent SAT solver techniques. For comparison, we run the pure SAT encodings on a state-of-the-art SAT solver Kissat (Biere et al. 2024). The number of solved instances by Kissat is reported in Table 3. Kissat is able to solve more instances than MonoSAT using the pure SAT encoding, highlighting the improvement in SAT solver technologies and so the potential for even greater performance for our approach if a more recent SAT solver like Kissat were hybridised with graph theory. Even now, the best performing *Graph* encoding, *Pass+Graph*, solved with the

		20	40	60	80	100	brc	den	ost	total
mks	*P*	512	**366**	**198**	**80**	**41**	0	0	0	**1197**
	S	**522**	302	145	61	30	0	0	0	1060
soc	*P*	125	216	222	273	254	29	70	50	1239
	S	**143**	**238**	**271**	**308**	**272**	**32**	**84**	**56**	**1404**

Table 3: The number of solved instances by the pure SAT encodings using the Kissat solver. The results are split by map size and by the optimized cost function.

older MonoSAT solver, still outperforms the best performing pure SAT encoding solved with Kissat.

Related Work and Closing Discussion

We have proposed a novel approach for solving MAPF by decomposing the problem into two distinct components: path validity and collision avoidance. Our method leverages an SMT solver, MonoSAT, to which we delegate graph reachability constraints while handling collision avoidance via a Boolean formula. We applied this approach to two different styles of encoding, *Pass* and *Shift*, combined with two different objective functions, makespan and sum of costs. In all settings, our approach achieves a drastic reduction in the size of the encoding and a commensurate reduction in the time taken to generate it. In terms of runtime performance, for a fixed solver, the *Pass* encoding benefits greatly from connectivity reasoning via the graph propagator for makespan and sum of costs objectives, doubling and improving by a quarter the number of solved instances respectively. Runtime performance for the *Shift* encoding is improved significantly for the makespan objective but degraded for sum of costs. Investigating this latter finding is an important item of future work. Our present conjecture is that connectivity reasoning over this more compact graph is not as strong.

Only a handful of papers aim to solve MAPF problems via SMT. SMT-CBS^R algorithm (Surynek 2019a) solves MAPF with continuous time, geometric agents, and Euclidean space. Hence, a direct comparison to our work is not possible. SMT-CBS (Surynek 2019b) solves the classical MAPF problem by lazy reduction to SAT, where the constraints forbidding conflicts are excluded and added iteratively only after a conflict in the solution is detected. Despite the name, no SMT solver is used. Rather, an SMT-like framework ‘generate and test‘ is created. We did a preliminary experiment with the lazy conflict clause generation, but we were unable to confirm the benefit claimed by the initial study. Hence, we did not make a direct comparison.

None of the solvers was used in incremental mode (Kissat currently does not include incremental solving). Incremental solving for increasing the cost function is not practical, since increasing cost makes some extra vertices reachable, extending the right-hand side of formulas (4) and (8). Excluding the variables of unreachable vertices in preprocessing generally outweighs the benefits of incremental solving. Further study on incremental solving in MAPF (both lazy clause generation and cost increment) would be beneficial.

Acknowledgments

[Hidden]

References

Achá, R. A.; López, R.; Hagedorn, S.; and Baier, J. 2021. A New Boolean Encoding for MAPF and its Performance with ASP and MaxSAT Solvers. In *Proceedings of the Fourteenth International Symposium on Combinatorial Search (SOCS'21)*, 11–19. AAAI Press.

Achá, R. A.; López, R.; Hagedorn, S.; and Baier, J. 2022. Multi-Agent Path Finding: A New Boolean Encoding. *Journal of Artificial Intelligence Research*, 75: 323–350.

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

Barták, R.; and Švancara, J. 2019. On SAT-Based Approaches for Multi-Agent Path Finding with the Sum-of-Costs Objective. In *Proceedings of the Twelfth International Symposium on Combinatorial Search (SOCS'19)*, 10–17. AAAI Press.

Barták, R.; Zhou, N.; Stern, R.; Boyarski, E.; and Surynek, P. 2017. Modeling and Solving the Multi-agent Pathfinding Problem in Picat. In *Proceedings of the Twenty-ninth IEEE International Conference on Tools with Artificial Intelligence (ICTAI'17)*, 959–966. IEEE Computer Society Press.

Bayless, S.; Bayless, N.; Hoos, H.; and Hu, A. 2015. SAT Modulo Monotonic Theories. In *Proceedings of the Twenty-ninth National Conference on Artificial Intelligence (AAAI'15)*, 3702–3709. AAAI Press.

Biere, A.; Faller, T.; Fazekas, K.; Fleury, M.; Froleyks, N.; and Pollitt, F. 2024. CaDiCaL, Gimsatul, IsaSAT and Kissat Entering the SAT Competition 2024. In *Proc. of SAT Competition 2024 – Solver, Benchmark and Proof Checker Descriptions*, volume B-2024-1 of *Department of Computer Science Report Series B*, 8–10. University of Helsinki.

Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds. 2021. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press. ISBN 978-1-64368-160-3.

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann Publishers.

Dresner, K.; and Stone, P. 2008. A Multiagent Approach to Autonomous Intersection Management. *Journal of Artificial Intelligence Research*, 31: 591–656.

Gebser, M.; Janhunen, T.; and Rintanen, J. 2014. SAT Modulo Graphs: Acyclicity. In *Proceedings of the Fourteenth European Conference on Logics in Artificial Intelligence (JELIA'14)*, volume 8761 of *Lecture Notes in Artificial Intelligence*, 137–151. Springer-Verlag.

Kaduri, O.; Boyarski, E.; and Stern, R. 2020. Algorithm Selection for Optimal Multi-Agent Pathfinding. In *Proceedings of the Thirty-fourth National Conference on Artificial Intelligence (AAAI'20)*, 161–165. AAAI Press.

Morris, R.; Pasareanu, C.; Luckow, K.; Malik, W.; Ma, H.; Kumar, T.; and Koenig, S. 2016. Planning, Scheduling and Monitoring for Airport Surface Operations. In *The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence: Planning for Hybrid Systems*, 608–614.

Philipp, T.; and Steinke, P. 2015. PBLib – A Library for Encoding Pseudo-Boolean Constraints into CNF. In *Proceedings of the Eighteenth International Conference on Theory and Applications of Satisfiability Testing (SAT'15)*, volume 9340 of *Lecture Notes in Computer Science*, 9–16. Springer-Verlag.

Rankooh, M. F.; and Rintanen, J. 2022. Propositional Encodings of Acyclicity and Reachability by Using Vertex Elimination. In *Proceedings of the Thirty-sixth National Conference on Artificial Intelligence (AAAI'22)*, 5861–5868. AAAI Press.

Schrijver, A. 1999. *Theory of linear and integer programming*. Discrete mathematics and optimization. John Wiley & sons.

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

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2011. The Increasing Cost Tree Search for Optimal Multi-Agent Pathfinding. In *Proceedings of the Twenty-second International Joint Conference on Artificial Intelligence (IJCAI'11)*, 662–667. IJCAI/AAAI Press.

Silver, D. 2005. Cooperative Pathfinding. In *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE'05)*, 117–122. AAAI Press.

Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T.; Barták, R.; and Boyarski, E. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Proceedings of the Twelfth International Symposium on Combinatorial Search (SOCS'19)*, 151–159. AAAI Press.

Surynek, P. 2014. Compact Representations of Cooperative Path-Finding as SAT Based on Matchings in Bipartite Graphs. In *Proceedings of the Twenty-sixth IEEE International Conference on Tools with Artificial Intelligence (ICTAI'14)*, 875–882. IEEE Computer Society Press.

Surynek, P. 2015. On the Complexity of Optimal Parallel Cooperative Path-Finding. *Fundamenta Informaticae*, 137(4): 517–548.

Surynek, P. 2016. Makespan Optimal Solving of Cooperative Path-Finding via Reductions to Propositional Satisfiability. *CoRR*, abs/1610.05452.

Surynek, P. 2019a. Multi-Agent Path Finding with Continuous Time and Geometric Agents Viewed through Satisfiability Modulo Theories (SMT). In *Proceedings of the Twelfth International Symposium on Combinatorial Search (SOCS'19)*, 200–201. AAAI Press.

Surynek, P. 2019b. 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)*, 1177–1183. ijcai.org.

Surynek, P. 2022. Problem Compilation for Multi-Agent Path Finding: a Survey. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI'22)*, 5615–5622. ijcai.org.

Surynek, P.; Felner, A.; Stern, R.; and Boyarski, E. 2016. 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)*, 810–818. IOS Press.

Švancara, J.; Atzmon, D.; Strauch, K.; Kaminski, R.; and Schaub, T. 2024. Which Objective Function is Solved Faster in Multi-Agent Pathfinding? It Depends. In *Proceedings of the Sixteenth International Conference on Agents and Artificial Intelligence (ICAART'24)*, 23–33. SciTePress.

Wurman, P.; D'Andrea, R.; and Mountz, M. 2008. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine*, 29(1): 9–20.

Yu, J.; and LaValle, S. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *Proceedings of the Twenty-seventh National Conference on Artificial Intelligence (AAAI'13)*, 1443–1449. AAAI Press.