A New Boolean Encoding for MAPF and Its Performance with ASP and MaxSAT Solvers

Roberto Asín Achá,[1] **Rodrigo López,**[2] **Sebastián Hagedorn,**[2] **Jorge A. Baier**[2,3]

[1] Universidad de Concepción, Concepción, Chile
[2] Pontificia Universidad Católica de Chile, Santiago, Chile
[3] Instituto Milenio Fundamentos de los Datos, Santiago, Chile
rasin@udec.cl, {rilopez3, shagedorn}@uc.cl, jabaier@ing.puc.cl

Abstract

Multi-agent pathfinding (MAPF) is an NP-hard problem. As such, dense maps may be very hard to solve optimally. In such scenarios, compilation-based approaches, via Boolean satisfiability (SAT) and answer set programming (ASP), have proven to be most effective. In this paper, we propose a new encoding for MAPF, which we implement and solve using both ASP and MaxSAT solvers. Our encoding builds on a recent ASP encoding for MAPF but changes the way agent moves are encoded. This allows to represent swap and follow conflicts with binary clauses, which are known to work well along with conflict-based clause learning. For MaxSAT, we study different ways in which we may combine the MSU3 and LSU algorithms for maximum performance. Our results, over grid and warehouse maps, show that the ASP solver scales better when the number of agents is increased on grids with few obstacles, while the MaxSAT solver performs better in scenarios with more obstacles and fewer agents.

Introduction

Given a graph G and K agents, each of which is associated with a start and a goal vertex of G, multi-agent pathfinding (MAPF) is the problem of finding k non-conflicting paths $\pi_1, \ldots, \pi_K$, such that π_i connects the start and goal vertex associated with agent i. MAPF has many applications. It is key for the implementation of automated warehouses, multi-agent videogames (Wang and Botea 2008), and has become increasingly relevant in other important applications such as airport ground control (e.g., Li et al. 2019b).

Optimal MAPF solving is NP-hard (Surynek 2010; Yu and LaValle 2013; Nebel 2020). Not surprisingly, optimal solvers have scalability issues. Consequently, solving dense MAPF instances, that is, instances with a high proportion of space occupied by either agents or obstacles, can be challenging even when maps are relatively small.

Most MAPF solvers can be classified as either search- or compilation-based or hybridized. While the former (e.g., Sharon et al. 2012; Felner et al. 2018; Li et al. 2019a) can scale to large maps, they do not perform very well in relatively small, dense maps. Compilation-based techniques, instead, translate the MAPF instance to an instance of another problem, for example Boolean satisfiability (SAT) (e.g., Surynek et al. 2016; Barták et al. 2017; Barták and Svancara 2019), answer set programming (ASP) (e.g., Erdem et al. 2013; Gebser et al. 2018; Nguyen et al. 2017; Gómez, Hernández, and Baier 2021), or Mixed-Integer Programming (MIP) (e.g., Barták et al. 2017). They do perform better than search-based solvers in dense, rather small maps, but do not scale to large maps. Hybridized methods (e.g., Lam et al. 2019) incorpporate elements of both.

The efficiency of compilation-based approaches depends on a number of factors, including the base SAT/ASP/MIP solver, but also on the encoding used. Recently, Gómez, Hernández, and Baier (2021) showed that focusing on generating a smaller encoding, linear on the number of agents rather than quadratic, yielded significant benefits in practice.

In this paper, we propose a new Boolean encoding for sum-of-costs MAPF. The encoding shares common aspects with the recent encoding introduced in ASP-MAPF (Gómez, Hernández, and Baier 2021), but encodes the action decisions in a different way, allowing to represent swap and follow conflicts using binary clauses involving two decision variables. We design our encoding with a focus on MaxSAT solvers and show briefly how to adapt it for ASP.

Furthermore, this paper presents the first study and evaluation of MAPF encoding using MaxSAT technology. For this reason, even though the encoding can be represented in ASP, our presentation and experimental section focuses primarily on MaxSAT solving. As such we investigate the effectiveness of using different optimization algorithms (e.g., MSU3/LSU), and a number of encodings for cardinality constraints, and report the configurations that work best in practice. In addition, we investigate the benefits of including redundant clauses, which allow the MaxSAT solver to reduce the search space, resulting in better scaling.

In our experimental evaluation, we compare our MaxSAT and our ASP MAPF solvers with ASP-MAPF, MDD-SAT (Surynek et al. 2016), a SAT-based MAPF solver, and CBS-H2 (Li et al. 2019a), a state-of-the-art search-based solver. We use a number of grid and warehouse benchmarks. We observe that our ASP solver scales better when the number of agents is increased on grids with few obstacles, while our MaxSAT solver performs better in scenarios with more obstacles and fewer agents. Our results support the fact that our encoding is superior to previous ones, independently of the technology used.

Background

We start off with background on MAPF, following Stern et al. (2019) rather closely. Then we present background on MaxSAT and MAPF compilations relevant for our work.

Multi-Agent Pathfinding

A MAPF instance is defined by a tuple $(G, K, start, goal)$, where $G = (V, E)$ is a directed graph, K is the number of agents, and $start : \{1, \ldots, K\} \rightarrow V$ and $goal : \{1, \ldots, K\} \rightarrow V$ specify the start and goal vertex for each of the agents. In addition, E contains the edge (u, u), for every $u \in V$. This constraint is important, since it allows agents to 'wait' at their current location.

A *path over graph* G is a sequence of edges $\pi = (u_0, u_1)(u_1, u_2) \ldots (u_{n-1}, u_n)$, for every $i \in \{0, \ldots, n-1\}$. Given a path $\pi = (u_0, u_1) \ldots (u_{n-1}, u_n)$, we denote the vertex reached by traversing the first t edges of π by $\pi[t]$; formally $\pi[t] = u_t$, where $0 \leq t \leq |\pi|$. If $\pi[t] = v$ we say that the path *visits vertex* v *at time step* t. In this paper, like in most of the literature on MAPF (Stern et al. 2019), we consider that agents stay at their target after reaching it. Thus, for any $t > |\pi|$, we define $\pi[t] = \pi[|\pi|]$. A path π *connects* u *with* v iff $\pi[0] = u$ and $\pi[|\pi|] = v$. Given two nodes $u, v \in V$ we denote by $d(u, v)$ the length of the shortest path connecting u and v.

A *solution* to a MAPF instance $(G, K, start, goal)$ is a tuple of K paths over G, $\Pi = (\pi_1, \pi_2, \ldots, \pi_K)$, where:

1. The last element of π_k is not of the form (u, u), for every $k \in \{1, \ldots, K\}$.
2. π_k connects $start(k)$ with $goal(k)$, for every $k \in \{1, \ldots, K\}$.
3. Π is *conflict-free*.

Intuitively, Π has a conflict if the paths of two or more agents interfere with each other. A relevant observation is that condition 1 is necessary for the definition of the cost of a path, which we introduce below. A non-solution that satisfies conditions 2 and 3 but not 1 can always be transformed into a solution by removing the largest suffix of wait actions from every path.

Different types of conflicts have been considered in the MAPF literature. Those relevant to this paper follow.

- **Vertex Conflicts**. Π has a *vertex conflict* if two paths π_i, π_j in Π, where $i \neq j$, are such that $\pi_i[t] = \pi_j[t]$, for some $t \geq 0$. Intuitively, a vertex conflict occurs when two agents visit the same vertex at the same time step.
- **Swap Conflicts**. Π has an *swap conflict* if two paths π_i, π_j in Π, where $i \neq j$, are such that $(\pi_i[t], \pi_i[t+1]) = (\pi_j[t+1], \pi_j[t])$, for some $t \geq 0$. Intuitively, a vertex conflict occurs when two agents traverse the same edge in opposite direction at the same time instant.
- **Follow Conflicts**. Π has a *follow conflict* if two paths π_i, π_j in Π, where $i \neq j$, are such that $\pi_i[t] = \pi_j[t+1]$, for some $t \geq 0$. Intuitively, a follow conflict occurs when at time $t+1$ an agent occupies the position another agent had at time step t.

Most literature on MAPF (e.g., Sharon et al. 2012) has focused on solvers for solutions free of vertex and swap conflict. The most relevant compilation-based solvers that we consider in this paper (Surynek et al. 2016; Surynek 2019), compute solutions free of vertex, swap, and follow conflicts.

The *sum-of-costs* (SOC) of a solution $\Pi = (\pi_1, \pi_2, \ldots, \pi_K)$ is $c(\Pi) = \sum_{i=1}^{K} |\pi_i|$, whereas its *makespan* is $makespan(\Pi) = \max_{i \in \{1,\ldots,K\}} |\pi_i|$. A solution Π to an instance of MAPF is SOC-optimal iff the SOC of every other solution is equal to or greater than $c(\Pi)$. A solution Π to an instance of MAPF is makespan-optimal iff the makespan of every other solution is equal to or greater than $makespan(\Pi)$.

Maximum Boolean Satisfiability

Given a set X of *variables*, ℓ is a *literal* over X iff $\ell = x$ or $\ell = \neg x$, for some $x \in X$. A *clause* over X is a set of literals over X. Given a literal ℓ over X, its *complement* $\overline{\ell}$ is defined as x if $\ell = \neg x$, and as $\neg x$ if $\ell = x$, for some variable x. A *boolean assignment* for X is a function $\sigma : X \rightarrow \{true, false\}$. An assignment *satisfies* Y iff $\sigma \models Y$, where the binary relation $\models$ is such that $\sigma \models x$ iff $\sigma(x) = true$, and $\sigma \models \neg x$ iff $\sigma(x) \not\models x$, for every $x \in X$. If C is a clause $\sigma \models C$ iff $\sigma \models \ell$, for some $\ell \in C$. If S is a set of clauses $\sigma \models S$ iff $\sigma \models C$, for every $C \in S$.

Given a set of clauses S over variables X, the *boolean satisfiability* problem (SAT) consists of computing an assignment σ over X such that $\sigma \models S$. Given a pair of two sets of clauses (H, S), where H corresponds to a set of *hard* clauses, and S is a set of *soft* clauses, the *(partial) maximum boolean satisfiability problem* (MaxSAT), consists of finding an assignment σ that satisfies H and maximizes the number of clauses of S which it satisfies.

Given a set of literals $L = \{\ell_1, \ldots, \ell_n\}$, $\bigwedge L$ stands for $\ell_1 \wedge \ldots \wedge \ell_n$, $\bigvee L$ stands for $\ell_1 \vee \ldots \vee \ell_n$, and $\overline{C}$ stands for $\{\overline{\ell_1}, \ldots, \overline{\ell_n}\}$. A clause C logically corresponds to the disjunction $\bigvee C$, which in turn is logically equivalent to the implication $\bigwedge \overline{B} \rightarrow \bigvee T$, where B and T are disjoint and such that $C = B \cup T$.

For our experimental evaluation, it is relevant to review the state-of-the-art approaches to MaxSAT. The following are approaches that have been used by solvers participating in recent MaxSAT competitions (e.g., Bacchus et al. 2020).

SAT-based These algorithms iteratively call an underlying SAT solver where each iteration imposes a stronger bound for the number of unsatisfied clauses in S. Between consecutive calls, the bound is incremented by 1. The the first call that returns UNSAT yields a MaxSAT assignment. The most used SAT-based algorithm is known as LSU (Biere, Heule, and van Maaren 2009). Although not usually competitive in the MaxSAT competitions, a strength of this approach is its anytime property which allows it to output a sequence of assignments of increasing quality.

UNSAT-based Also an iterative approach. In the first iteration all clauses are treated as hard. In subsequent iterations, depending on the reasoning of the algorithm, some or all original soft clauses are relaxed by adding cardinality constraints imposing that at most a certain number of

soft clauses may be unsatisfied. Bounds imposed on the relaxation are sound, which guarantees that the first SAT call returns an optimal assignment. MSU3 (Morgado, Liffiton, and Marques-Silva 2012), RC2 (Ignatiev, Morgado, and Marques-Silva 2019) and EvalMaxSAT (Avellaneda 2020) are examples of UNSAT-based solvers.

SAT-UNSAT-based Algorithms belonging to this category mix the two strategies from above. Pacose (Paxian, Reimer, and Becker 2019), QMaxSAT (Koshimura et al. 2012), UWrMaxSAT (Piotrów 2019), are examples of solvers based on this paradigm.

Hitting-set-based The solvers belonging to this category separate the problem in two: core extraction, performed by a SAT solver and optimisation, performed by an Integer Linear Programming (ILP) solver. By doing so, this kind of solvers avoid increasing the complexity of the SAT problem, but instead they use the ILP optimizer for doing the numerical reasoning. The main representative of this paradigm is MaxHS (Berg, Bacchus, and Poole 2020).

Existing Compilation-Based Approaches

We briefly review relevant aspects of the two most related compilation-based approaches: MDD-SAT (Surynek et al. 2016) and ASP-MAPF (Gómez, Hernández, and Baier 2021). Both approaches encode the problem using propositional variables relative to a certain time instant. For example, MDD-SAT uses variable $\mathcal{X}_j^t(a_i)$ to express that agent a_i is at vertex v_j at time t, and variable $\mathcal{E}_{j,k}^t(a_i)$ to express the fact that agent a_i moves from vertex v_j to vertex v_k at time t. The way variables are chosen is relevant since this determines the way constraints are written. For example, to encode follow and swap conflicts simultaneously, MDD-SAT uses the constraint:

$$\mathcal{E}_{j,k} \rightarrow \bigwedge_{a_\ell \in A, a_\ell \neq a_i, u_k^{t+1} \in V_\ell} \neg \mathcal{X}_k^{t+1}(a_\ell),$$

which expands to a number of clauses that is quadratic on the number of agents, and linear on the number of edges of the graph. The following table shows the number of clauses needed by MDD-SAT and ASP-MAPF to encode the different types of conflicts, where T is the time bound of the encoding, and whether or not the full encoding is redundant.

	MDD-SAT	ASP-MAPF
vertex conflicts	$O(K^2 \cdot \lvert V\rvert \cdot T)$	$O(K \cdot \lvert V\rvert \cdot T)$
follow conflicts	$O(K^2 \cdot \lvert E\rvert \cdot T)$	N/A
swap conflicts	$O(K^2 \cdot \lvert E\rvert \cdot T)$	$O(K \cdot \lvert E\rvert \cdot T)$
redundant?	non-redundant	non-redundant

MAPF to MaxSAT

We follow the same principle of previous compilation-based approaches to MAPF, encoding the position of the agents using propositional variables, for different time steps in $\{0, \ldots, T\}$. As previous approaches (e.g., Surynek et al. 2016; Gómez, Hernández, and Baier 2021), we optimize the number of variables by, in practice, removing those variables that would encode an agent a being at a vertex v that is not reachable by a given the constraints imposed by the bound T. To that end, we define $\mathit{Feasible}(a, t)$ as the set of vertices that can be *feasibly reached* by a. An agent can feasibly reach a vertex v at time t if it can reach v in t steps or less and the goal is reachable from v by time step T. Formally,

$$\mathit{Feasible}(a, t) = \{u \in V : d(\mathit{start}(a), u) \leq t, d(u, \mathit{goal}(a)) \leq T - t\}.$$

To compute set $\mathit{Feasible}(a, t)$ efficiently, before encoding our problem we run Dijkstra's algorithm to compute the d function, for each start and goal state as a source.

Below we use $\mathcal{A}$ to abbreviate $\{1, \ldots, K\}$. The variables we use in our encoding are as follows.

Variables

1. $at_{a,u,t}$: agent a is at vertex u at time step t, where $a \in \mathcal{A}$, $u \in \mathit{Feasible}(a, t)$, and $t \in \{0, \ldots, T\}$.
2. $\mathit{shift}_{u,v,t}$: vertex u *shifts towards* v at time step t, where $(u, v) \in E$, and $t \in \{0, \ldots, T-1\}$. Specifically, this means that if an agent is at u, then it should move towards v. Vertices shift regardless of whether or not there is an agent at them.
3. $\mathit{finalState}_{a,t}$: agent a is at its goal vertex, $\mathit{goal}(a)$, at time t, and will not move in the future. This variable is defined for every $a \in \mathcal{A}$, and every $t \in \{0, \ldots, T\}$. These variables allow us to define our optimization function, are inspired by ASP-MAPF and defined in an analogous way.

It is important to note here that the use of the $\mathit{shift}_{u,v,t}$, which is a variable that does not refer to the agent, is an important difference with the encoding of MDD-SAT and of ASP-MAPF. Indeed, both of these approaches use variables that involve the agents for action decisions. As mentioned above, MDD-SAT uses a variable $\mathcal{E}_{j,k}^t(a_i)$, to express that agent a_i moves from vertex j to vertex k at time t, while ASP-MAPF uses a variable `exec(A,M,T)` to express that agent `A` performs move `M` at time `T`. As we see later in our experimental section, this rather simple change yields important speedups experimentally.

The Encoding

A constraint on *shift* variables We start off by expressing that every vertex u shifts to exactly one of its neighbors at every time interval, by including the cardinality constraint:

$$\sum_{v:(u,v)\in E} \mathit{shift}_{u,v,t} = 1, \qquad \text{(C1)}$$

for every $u \in V$, and every $t \in \{0, \ldots, T-1\}$.

Encoding the Agents' locations Now we define the relationship between the at variables and the shift variables. If one understands the at variable actually as a *fluent*, as one may do when understanding our model as a theory of action (e.g., Reiter 2001), we would generate a so-called *successor state axiom*, which, in the context of (C1) is as follows:

$$at_{a,v,t+1} \leftrightarrow \bigvee_{u:(u,v)\in E} at_{a,u,t} \wedge \mathit{shift}_{u,v,t}, \qquad \text{(SSA)}$$

for every $v \in V$. However, translating (SSA) results in many clauses; indeed, it results on average in $\Theta(K \cdot |V| \cdot 2^n)$ clauses, where n is the average number of successors for a node in the graph. Most of such clauses, however, are redundant with each other. Moreover (SSA) does not take into account that $at_{a,u,t}$ is only defined when $u \in \mathit{Feasible}(a,t)$. Instead of (SSA), we use the clauses corresponding to the following four families of formulas.

$$at_{a,u,t} \wedge \mathit{shift}_{u,v,t} \rightarrow at_{a,v,t+1}, \quad \text{(H1)}$$

$$at_{a,u,t} \wedge at_{a,v,t+1} \rightarrow \mathit{shift}_{u,v,t}, \quad \text{(H2)}$$

for every $a \in \mathcal{A}$, every $u \in \mathit{Feasible}(a,t)$ every v such that $(u,v) \in E$ which is such that $v \in \mathit{Feasible}(a,t+1)$, and every $t \in \{0,\ldots,T-1\}$,

$$at_{a,v,t+1} \rightarrow \bigvee_{u:(u,v)\in E, u\in \mathit{Feasible}(a,t)} at_{a,u,t}, \quad \text{(H3)}$$

and:

$$at_{a,u,t} \rightarrow \overline{\mathit{shift}_{u,v,t}}, \quad \text{(H4)}$$

for every $u \in \mathit{Feasible}(a,t)$, every $t \in \{0,\ldots,T\}$, and every v such that $(u,v) \in E$ and such that $v \in V \setminus \mathit{Feasible}(a,t+1)$.

Formula (H1) can be regarded as a *positive effect axiom* (Reiter 1991, 2001), which establishes that when agent a is at vertex u and a shift from u to v occurs at time t, then a is at v at time $t+1$. Formula (H2) could be viewed as an explanation axiom: if an agent has moved from u to v between time t and time $t+1$, then such a move is explained by a shift from u to v at time t. Formula (H3) establishes that if an agent is at a certain vertex u at time $t+1$, then at the previous time instant, it must have been at one of the predecessors of u. Finally, formula (H4) could be viewed as a special case of (H1) had (H1) been defined for any $(u,v) \in E$. In such a case, we would have needed to account for the fact that $at_{a,v,t+1}$ is not defined—and thus should be considered equivalent to *false*—when $v \notin \mathit{Feasible}(a,t+1)$. In words, (H4) says it is not possible to shift an agent to an unfeasible vertex.

The following result provides a formal account of the correctness of our approach to encoding the agents' locations, by identifying the relationship between formulas (H1)–(H4) and the successor state axiom of (SSA).

Proposition 1 *The conjunction of:*

F1. any formula logically equivalent to (C1)*,*

F2. a formula specifying that every agent is in exactly one location at $t = 0$, and

F3. clauses of the form $\overline{at_{a,u,t}}$, for every $a \in \mathcal{A}$, $u \in V \setminus \mathit{Feasible}(a,t)$, and $t \in \{0,\ldots,T\}$,

implies that (SSA) *is equivalent to the conjunction of* (H1)–(H4).

Proof sketch: As a first step of our proof, we prove a lemma (LEM) that establishes that F1, F2, and (SSA) imply that every agent is at single location, for every $t \in \{1,\ldots,T\}$. The proof is by induction on t. Then, for the $\Rightarrow$ side of the proof, we obtain (H1) and (H3) directly from (SSA) by syntactic manipulation. We use LEM to simplify formulas of the form $\bigvee_{j:(j,v)\in E; j\neq u} at_{a,j,t}$ into $\overline{at_{a,u,t}}$. This allows us to generate (H2). Then we obtain (H4) by simplifying (H2) with F3. For the $\Leftarrow$ direction, we use resolution between clauses in (H2), (H4) and (H3) to obtain clauses, which, conjoined with (H1)–(H4) result in (SSA). ■

Encoding *finalState* variables We continue defining the dynamics of the *finalState* variables. For every $a \in \mathcal{A}$ and every $t \in \{d(start(a), goal(a)),\ldots,T-1\}$ we define:

$$\mathit{finalState}_{a,T}, \quad \text{(H5)}$$

$$at_{a,goal(a),t} \wedge \mathit{finalState}_{a,t+1} \leftrightarrow \mathit{finalState}_{a,t}. \quad \text{(H6)}$$

The first formula, (H5), expresses that, at time T, agent a is at its final state. Indeed, as required below by (H8), each agent should reach its goal at time T. Formula (H6) expresses that in t, a is at its final state if and only if it was already at its final state at $t+1$ and a is still at its goal location at time instant t.

Initial and Goal Conditions To define the start location of the agents, we add, for every $a \in \mathcal{A}$:

$$at_{a,start(a),0}. \quad \text{(H7)}$$

Note that it is not necessary to explicitly specify that the agent is *not* at other locations different from $start(a)$ at $t = 0$, since $start(a)$ is the only location that is in $\mathit{Feasible}(a,0)$.

For the goal condition, we add, for every $a \in \mathcal{A}$:

$$at_{a,goal(a),T}. \quad \text{(H8)}$$

Vertex Conflicts To make solutions comply with vertex conflicts, we prohibit two or more agents occupying the same vertex at a particular time, using the following cardinality constraint:

$$\sum_{a:u\in \mathit{Feasible}(a,t)} at_{a,u,t} \leq 1, \quad \text{(C2)}$$

for every $t \in \{0,\ldots,T\}$.

Swap Conflicts To enforce solutions respect swap conflicts, we forbid one edge being used in two different directions at the same time step, using:

$$\mathit{shift}_{u,v,t} \rightarrow \overline{\mathit{shift}_{v,u,t}}, \quad \text{(H9)}$$

for every $(u,v) \in E$, and every $t \in \{0,\ldots,T-1\}$.

Follow Conflicts To get solutions to respect follow conflicts, if there is a shift towards a specific vertex v, we force that such a vertex to perform a shift towards itself, using the following formula,

$$\mathit{shift}_{u,v,t} \rightarrow \mathit{shift}_{v,v,t}. \quad \text{(H10)}$$

Cost Minimization To minimize the sum-of-costs, like Gómez, Hernández, and Baier (2021), we maximize the number of time intervals for which each agent has been at its goal state. To that end, we add the following *soft* clauses:

$$\mathit{finalState}(a,t), \quad \text{(S1)}$$

for each $a \in \mathcal{A}$, and each $t \in \{d(start(a), goal(a)), \ldots, T\}$. We associate the same violation cost, equal to 1, with each clause.

Additional Constraints Up to this point our encoding does not include redundant clauses. However, as we see later in our experimental section, the MaxSAT solver benefits from the inclusion of redundant clauses. This is because these may trigger propagations that are helpful for the solver. Below, we consider two redundant constraints, the first of which is given by the following formula:

$$at_{a,u,t} \rightarrow \bigvee_{v:(u,v)\in E, v\in Feasible(a,t)} at_{a,v,t+1}, \qquad \text{(H11)}$$

for every $a \in \mathcal{A}$, every $u \in Feasible(a,t)$ and every $t \in \{0,\ldots,T-1\}$, establishes that if an agent is at certain vertex u at time t, then at time $t+1$ it will be at a neighbor v of u. The second constraint establishes that each agent is at some vertex of the graph at any given time. For every $a \in \mathcal{A}$ and $t \in \{0,\ldots,T\}$:

$$\sum_{u\in Feasible(a,t)} at_{a,u,t} = 1 \qquad \text{(C3)}$$

The following results show in what sense these constraints are redundant with the rest of the encoding.

Proposition 2 *Formula* (H11) *is implied by the conjunction of* (H1) *and any formula logically equivalent to* (C1)*.*

Proof: (C1) implies $\bigvee_{v:(u,v)\in E} shift_{u,v,t}$, for every $u \in V$, which together with (H1) implies (H11). ■

Proposition 3 *The conjunction of* (H1)–(H4)*, and* (H7) *and any formula logically equivalent to* (C1) *entails any formula logically equivalent to* (C3)*.*

Proof sketch: By induction on the total number of time steps of the encoding. In the base case ($t = 0$) (C3) holds by definition. For the induction, if the property holds at t, using (SSA) and (C1) we prove that, in $t+1$, (C3) holds. ■

Another observation that is important to make at this point is that the inclusion of (C3) in the encoding allows us to remove (H3), resulting in a more *compact* encoding. Indeed, this is because the number of clauses produced by (H3) is $O(K \cdot |E| \cdot T)$, whereas the number of clauses produced by (C3) is $O(S \cdot |V| \cdot T)$, where S is a function that depends on the way we encode cardinality constraints. Thus, substituting H3 may actually generate a more compact encoding. Such a substitution is justified by the following result.

Proposition 4 *The conjunction of* (H1)*,* (H2)*,* (H4)*,* (H7)*, and* (C1) *implies* (H3)*.*

Proof sketch: We prove that the conjunction of (H1), (H2), (H4), (H7), and the negation of (H3) implies the negation of (C1), since it require agents to be at certain location in $t+1$ which is not a neighbor of a location in t, effectively 'duplicating' agents on the graph. ■

Size of the Encoding

Any encoding constructed using the formulas above is at least linear in the size of the graph, number of agents, and time bound T, but the final size of the encoding depends on the particular way we encode cardinality constraints:

Proposition 5 *Let (ALL) denote the set that results from taking the union of the clauses generated by* (H1)–(H11) *and* (C1)–(C3)*. The size of (ALL) is* $O(K \cdot |E| \cdot T) + O(S \cdot (|V| + |E|) \cdot T)$*, where S is the size of the encoding for the cardinality constraints.*

Proof sketch: To prove the result, we observe that the number of clauses for (H1)-(H11) is bounded from above by the number of edges of the graph, $|E|$, similarly, the number of constraints of the form (C1). The number of constraints of the form (C2) and (C3), however, are bounded from above by $|V|$. ■

SOC-optimal MAPF via MaxSAT

With the encoding in hand, now we describe how we find SOC-optimal solutions to MAPF. A standard approach to find a plan in SAT-based planning (e.g., Kautz and Selman 1992) is to iteratively increase the time bound of the encoding until a solution is found. If one aims at cost optimality, this approach does not necessarily guarantees finding a SOC-optimal solution since SOC-optimal solutions are not necessarily makespan-minimal.

We follow a two-phase approach similar to the one described by Barták and Svancara (2019) and Gómez, Hernández, and Baier (2021). Phase 1 finds a Makespan-minimal with minimum cost. Let such a solution be Π_{min}, and its makespan be T_{min}. To compute it, we iteratively increment the bound of the encoding until a solution is found. The lower bound of the iteration is set to the makespan of a *relaxed* solution Π_{rel}, in which no conflicts are considered, and which is efficiently computed using Dijkstra's algorithm. In phase 2, we call the MaxSAT solver with an encoding whose time bound is set to $T_{min} + c(\Pi_{min}) - c(\Pi_{rel}) - 1$, which provably (Surynek et al. 2016; Gómez, Hernández, and Baier 2021) allows finding a SOC-optimal solution.

In our experimental evaluation we exploit the following two observations, which despite their simplicity have not been exploited by previous compilation-based approaches to MAPF. First, there is no need to run the same solver configuration on both phases. Second, the solution found on the first phase is a feasible (possibly non-optimal) solution for the second phase call. This allows us to pass such solution together with the input encoding to the MaxSAT call of the second phase. As a result, the solver may perform better, especially when using LSU as the optimization algorithm.

An ASP Encoding

For space restrictions, we do not provide a complete background of ASP. Instead, here we show how we can incorporate our new encoding into the encoding of ASP-MAPF (Gómez, Hernández, and Baier 2021). As we have explained

above, the main difference between our encoding and ASP-MAPF's is that our decision variable. This implies that instead of writing:

```
at(A,X,Y,T) :- exec(A,M,T-1),
               at(A,X',Y',T-1),
               delta(M,X',Y',X,Y).
```

we write:

```
at(R,X,Y,T) :- shift(X',Y',A,T-1),
               at(R,X',Y',T-1),
               delta(A,X',Y',X,Y).
```

To express only one action can be performed at each vertex we use the cardinality constraint:

```
1 {shift(X,Y,A,T-1) : action(A)} 1 :-
                        free(X,Y), time(T).
```

Swap conflicts are encoded with the two restrictions, analogous to (H9):

```
:- shift(X,Y,east,T),shift(X+1,Y,west,T).
:- shift(X,Y,north,T),shift(X,Y+1,south,T).
```

Follow conflicts are expressed by translating (H10) directly. The resulting encoding is $O(K \cdot |V| \cdot T)$.

Experimental Evaluation

Our empirical evaluation had the following objectives.

1. Our encoding includes redundant constraints. We wanted to evaluate whether or not redundancy provides any practical benefit.
2. As pointed out above, MaxSAT solvers can be run in many configurations (e.g., LSU, MSU3, etc.). We wanted to determine the best configuration.
3. Our encoding includes cardinality constraints, which can be encoded in different ways. We wanted to evaluate the relative performance of the resulting encodings.
4. We wanted to evaluate our encoding using ASP solvers and compare to compilation- and search-based state-of-the-art solvers.

Benchmarks Like other compilation-based approaches, ours does not scale to large maps. Thus we use the benchmarks used by Gómez, Hernández, and Baier (2021) which are medium-scale maps, which are challenging for both types of solvers.

AG: Instances in this set are 20×20 grids with 40 (10%) randomly placed obstacles with a number of agents that varies between 20 and 70, randomly distributed. Densest instances have around 27.5% of their cells occupied. This benchmark has 260 instances.

OBS: Instances in this are 20×20 with 20 randomly placed agents. These instances have between 0 and 70 obstacles, randomly placed on the grid. The densest instances have around 22.5% of cells occupied. This benchmark has 150 instances.

WH: All the instances in this set correspond to warehouse layouts of size 9×21 with aisles of width 1. These instances vary in the number of agents, randomly distributed on the grid, which can be between 4 and 30. The densest instances on the set have around 63.5% of cells occupied. This benchmark has 140 instances.

AMO Encoding	Clauses	Aux. Vars.
Pairwise (naive) **(pw)**	$O(n^2)$	0
Bitwise (Frisch and Peugniez 2001) **(bw)**	$O(n \log(n))$	$O(\log(n))$
Cardinality Nets (Asín et al. 2011) **(cn)**	$O(n)$	$O(n)$
Totalizer (Joshi et al. 2015) **(to)**	$O(n)$	$O(n)$
Km-totalizer (Joshi et al. 2015) **(kt)**	$O(n)$	$O(n)$
Sequential Counter (Sinz 2005) **(sc)**	$O(n)$	$O(n)$

Table 1: At-most-one encodings and number of auxiliary variables and clauses introduced to the encoding.

Our system, MtMS (MAPF through MaxSAT Solving)[1], is coded in C++ and integrated with the OpenWBO solver (Martins, Manquinho, and Lynce 2014). All experiments were run on a computer running Linux Mint 19.3 with an Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz and 16 Gbytes of RAM memory.

Algorithm Configuration

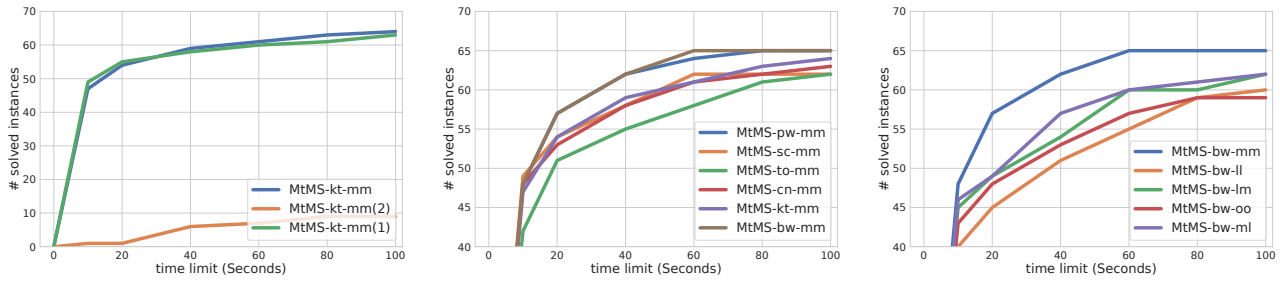
In this subsection we focus on the first three experimental objectives. We use a reduced set of hard instances of the AG Benchmark set, with 78 out of the 260 original instances, and a time limit of 100 seconds. To find the best-performing configuration, we followed a fix-all-but-one dimension search. Henceforth the name of our solver is followed by up to three suffixes. The first suffix abbreviates the algorithm to encode at-most-one cardinality constraints. We considered 6 different state-of-the-art algorithms for encoding these. Both the suffix we use and the number of clauses and auxiliary variables they employ are listed in Table 1.The second suffix corresponds to the MaxSAT algorithm used in each phase. Here we considered MSU3 (m) and LSU (l). Hence, if the suffix is ml, it means we use MSU3 on the first phase, and LSU on the second phase. OpenWBO implements other algorithms such as Part-MSU3, and OLL. We do not include them in this evaluation since they showed poor performance. In particular, Part-MSU3 cannot handle the size of our encoding. Finally, the third suffix is an optional suffix indicating the encoding used. We describe the encodings below.

Choosing an Encoding First we consider varying the encoding, while fixing the solving algorithm to MSU3 in both phases and the at-most-one encoding to Km-totalizer. We considered three encodings for testing:

- *Encoding* (0): Contains all constraints but (C3).
- *Encoding* (1): Contains all constraints.
- *Encoding* (2): A minimal encoding, which uses all constraints but (H3) and (H11).

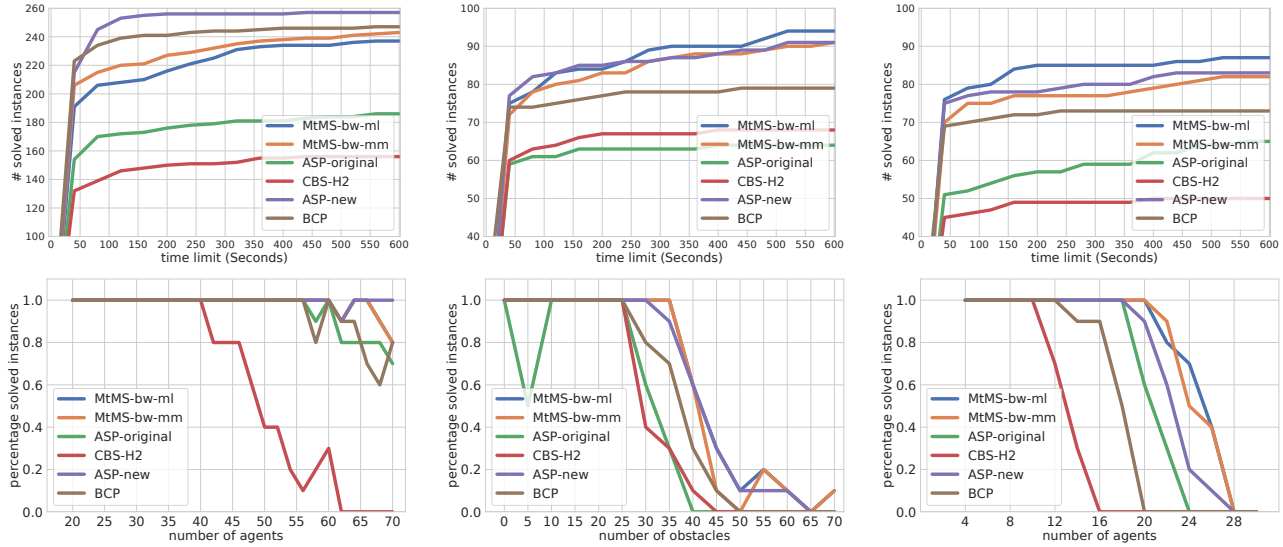
Figure 1 (a) shows the performance of the encodings on the reduced dataset. Encodings (0) and (1) have a similar performance, and outperform (2) by a great margin. This suggests that (H3) and (H11) help the conflict-driven clause learning (CDCL) procedure of the SAT solver to guide

[1]Source code and benchmarks available at https://github.com/robertoasin/MtMF



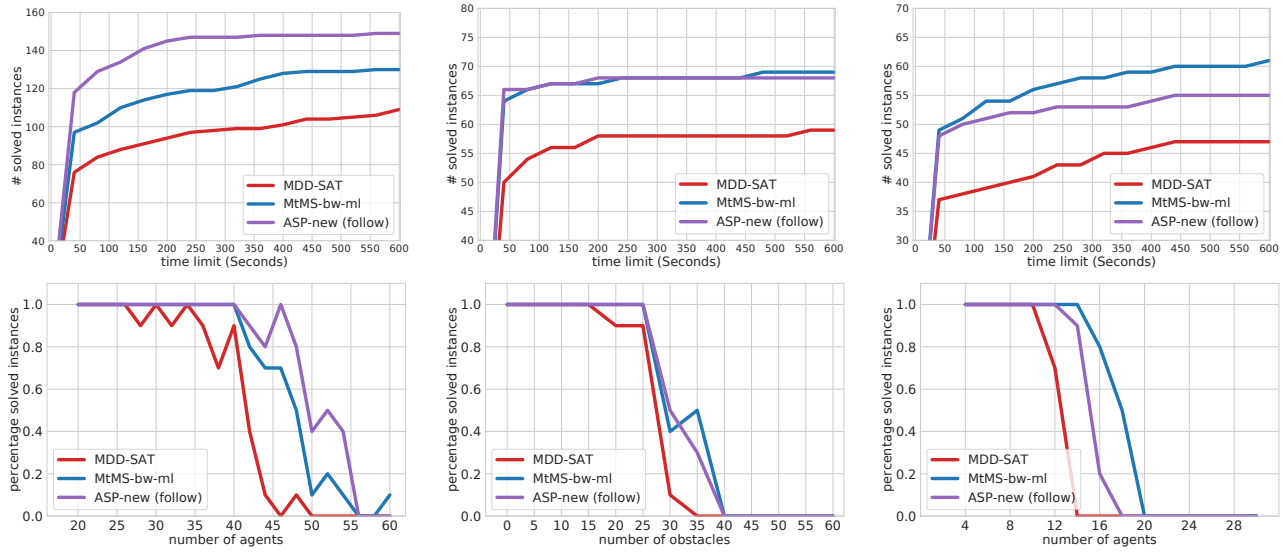
(a) (b) (c)

Figure 1: Number of MAPF instances solved by MtMS using (a) 3 different MAPF encodings (b) 6 different AMO encodings, and (c) 4 combinations of MaxSAT algorithms (MSU3:m, LSU:l) on phases 1 and 2.



AG: 20 × 20, increasing agents OBS: 20 × 20, increasing obstacles WH: Warehouse, increasing agents

Figure 2: Comparison with ASP-MAPF and CBS-H2, without follow conflicts. (AG: left; OBS: center; WH: right.)



AG: 20 × 20, increasing agents OBS: 20 × 20, increasing obstacles WH: Warehouse, increasing agents

Figure 3: Comparison with MDD-SAT, using follow conflicts. (AG: left; OBS: center; WH: right.)

search, most likely because they strengthen propagation, resulting in a reduction of the search space. We use the (0) encoding in all remaining experiments.

At-most-one (AMO) encoding We continue investigating the impact of encoding the cardinality constraints, leaving the encoding fixed to (0) and the algorithm configuration as fixed to "mm". Figure 1 (b) shows that AMO encodings to, cn, sc and kt are outperformed by the pairwise and bitwise. Henceforth we use bitwise to encode cardinality constraints due to its good performance on this benchmark and its good scalability properties.

Algorithm for Each Phase Figure 1 (c) shows the performance of MSU3 and LSU when used on phases 1 and 2. The MSU3-MSU3 achieves better results in this reduced dataset; nevertheless, we keep the MSU3-LSU combination for the following experiments, since we observed that on harder instances of the extended benchmark set, the latter yielded better results. We think the reason that explains this is as follows. When approaching the optimal value, LSU makes many hard SAT calls. For harder benchmarks, the number of hard SAT calls would be expected to increase, but by feeding the solution of phase 1 to LSU on phase 2, LSU seems to skip many hard SAT calls, guiding search more effectively, outperforming its MSU3 counterpart.

ASP Encoding and State of the Art Solvers

We implemented our encoding in ASP (ASP-new, below) and for our comparison with the state-of-the-art solvers, we chose to compare to ASP-MAPF and MDD-SAT, since they are both good representatives of compilation-based SOC-optimal solvers. Moreover, we chose CBS-H2 (Li et al. 2019a) as a reference for search-based approaches. For all solvers we use versions provided by their authors. ASP-new uses the same codebase as ASP-MAPF. In addition, we include the BCP (Lam et al. 2019), an optimal MAPF solver which is a hybrid solver integrating search-based MAPF techniques with and integer programming compilation.

Since MDD-SAT produces solutions respecting follow conflicts, but BCP, ASP-MAPF and CBS-H2 do not, we separate our experimental comparison in two parts, comparing only against MDD-SAT with follow conflicts (Figure 3) and without follow conflicts against BCP, ASP-MAPF and CBS-H2 (Figure 2). We compare on all three benchmarks, with the time limit set to 600 seconds per instance. We can make the observations that follow.

Without Follow Conflicts. Both variants of our MaxSAT approach: MtMS-bw-ml and MtMS-bw-mm outperform, and our ASP-new encoding outperform ASP-MAPF and CBS-H2 in all benchmark sets by a substantial margin (cf. Table 3). MtMS-bw-ml outperforms BCP. ASP-new scales better in domains with fewer obstacles and an increasing number of agents, while the MaxSAT approach scales better in domains with more obstacles.

With Follow Conflicts. For this comparison, we only use MtMS-bw-ml, since this solver seemed more robust in the previous setting. Our solver outperforms MDD-SAT by a substantial margin (cf. Table 2). Like in the previous benchmark, ASP-new scales better in domains with fewer obstacles and an increasing number of agents, while the MaxSAT approach scales better in domains with more obstacles.

	MtMS -bw-ml	MtMS -bw-mm	ASP -original	CBS -H2	ASP -new
AG	237	243	186	156	**257**
OBS	**94**	91	64	68	91
WH	**87**	82	65	50	83
Total	418	416	315	274	**431**

Table 2: # of solved instances at time limit (600 seconds).

	MDD-SAT	MtMS-bw-ml	ASP-new (follow)
AG	109	130	**150**
OBS	59	**69**	68
WH	47	**61**	55
Total	215	260	**273**

Table 3: # of solved instances using follow conflicts, at time limit (600 seconds).

Summary and Future Work

We presented a new Boolean encoding for SOC-optimal MAPF. We studied extensively how this encoding can be exploited within MaxSAT solvers. The encoding is compact. When implemented in ASP, it is linear on the number of agents, size of the map, and makespan. When implemented in MaxSAT the size depends on the encoding used for cardinality constraints. Specifically, our encoding for vertex conflicts, done via a cardinality constraint, may result in a number of clauses that may be linear, quadratic or superlinear on the number of agents, depending on the algorithm used to encode AMO constraints. Our encoding contains redundant clauses which, we show, benefit the solving process.

An important advantage of our encoding is that it can be exploited with the latest state-of-the-art ASP and MaxSAT technology, which has been designed for optimization. In our experimental evaluation, we study different solving configurations for MaxSAT, and determine that the bitwise encoding for AMO constraints together with a combination of the MSU3 algorithm for phase 1, plus the LSU algorithm for phase 2, results in the most effective configuration. Finally, we show that this configuration is superior to other state-of-the-art algorithms both search- and compilation-based, in dense, relatively small benchmarks.

As future work we plan to study the efficacy of other MaxSAT solvers and configuration. We also want to leverage this technology to scale to larger maps. Finally, it is not hard to extend our approach to the combined target-assignment and path-finding problem (TAPF). This is a topic of ongoing and future work.

References

Asín, R.; Nieuwenhuis, R.; Oliveras, A.; and Rodríguez-Carbonell, E. 2011. Cardinality networks: a theoretical and empirical study. *Constraints* 16(2): 195–221.

Avellaneda, F. 2020. A short description of the solver EvalMaxSAT. *MaxSAT Evaluation 2020* 8.

Bacchus, F.; Berg, J.; Järvisalo, M.; and Martins, R. 2020. *MaxSAT Evaluation 2020: Solver and Benchmark Descriptions*. University of Helsinki, Department of Computer Science.

Barták, R.; and Svancara, J. 2019. On SAT-Based Approaches for Multi-Agent Path Finding with the Sum-of-Costs Objective. In Surynek, P.; and Yeoh, W., eds., *SoCS*, 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 *ICTAI*, 959–966. Boston, MA, USA: IEEE Computer Society.

Berg, J.; Bacchus, F.; and Poole, A. 2020. Abstract Cores in Implicit Hitting Set MaxSat Solving. In *SAT*, 277–294. Springer.

Biere, A.; Heule, M.; and van Maaren, H. 2009. *Handbook of satisfiability*, volume 185. IOS press.

Erdem, E.; Kisa, D. G.; Öztok, U.; and Schüller, P. 2013. A General Formal Framework for Pathfinding Problems with Multiple Agents. In *AAAI*. AAAI Press.

Felner, A.; Li, J.; Boyarski, E.; Ma, H.; Cohen, L.; Kumar, T. K. S.; and Koenig, S. 2018. Adding Heuristics to Conflict-Based Search for Multi-Agent Path Finding. In *ICAPS*, 83–87. Delft, The Netherlands.

Frisch, A. M.; and Peugniez, T. J. 2001. Solving non-boolean satisfiability problems with stochastic local search. In *IJCAI*, volume 2001, 282–290.

Gebser, M.; Obermeier, P.; Otto, T.; Schaub, T.; Sabuncu, O.; Nguyen, V.; and Son, T. C. 2018. Experimenting with robotic intra-logistics domains. *TPLP* 18(3-4): 502–519.

Gómez, R. N.; Hernández, C.; and Baier, J. A. 2021. A Compact Answer Set Programming Encoding of Multi-Agent Pathfinding. *IEEE Access* 9: 26886–26901.

Ignatiev, A.; Morgado, A.; and Marques-Silva, J. 2019. RC2: An efficient MaxSAT solver. *Journal on Satisfiability, Boolean Modeling and Computation* 11(1): 53–64.

Joshi, S.; Martins, R.; and Manquinho, V. 2015. Generalized totalizer encoding for pseudo-boolean constraints. In *CP*, 200–209. Springer.

Kautz, H. A.; and Selman, B. 1992. Planning as Satisfiability. In Neumann, B., ed., *ECAI*, 359–363.

Koshimura, M.; Zhang, T.; Fujita, H.; and Hasegawa, R. 2012. QMaxSAT: A partial Max-SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation* 8(1-2): 95–100.

Lam, E.; Bodic, P. L.; Harabor, D. D.; and Stuckey, P. J. 2019. Branch-and-Cut-and-Price for Multi-Agent Pathfinding. In *IJCAI*, 1289–1296.

Li, J.; Felner, A.; Boyarski, E.; Ma, H.; and Koenig, S. 2019a. Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search. In *IJCAI*, 442–449.

Li, J.; Gong, M.; Liang, Z.; Liu, W.; Tong, Z.; Yi, L.; Morris, R.; Pasearanu, C.; and Koenig, S. 2019b. Departure Scheduling and Taxiway Path Planning under Uncertainty. In *AIAA*.

Martins, R.; Manquinho, V.; and Lynce, I. 2014. Open-WBO: A modular MaxSAT solver. In *SAT*, 438–445. Springer.

Morgado, A.; Liffiton, M.; and Marques-Silva, J. 2012. MaxSAT-based MCS enumeration. In *Haifa Verification Conference*, 86–101. Springer.

Nebel, B. 2020. On the Computational Complexity of Multi-Agent Pathfinding on Directed Graphs. In *ICAPS*, 212–216.

Nguyen, V.; Obermeier, P.; Son, T. C.; Schaub, T.; and Yeoh, W. 2017. Generalized Target Assignment and Path Finding Using Answer Set Programming. In Sierra, C., ed., *IJCAI*, 1216–1223. Melbourne, Australia: ijcai.org.

Paxian, T.; Reimer, S.; and Becker, B. 2019. Pacose: An Iterative SAT-based MaxSAT Solver. *MaxSAT Evaluation 2019* 9.

Piotrów, M. 2019. UWrMaxSat-a new MiniSat+-based Solver in MaxSAT Evaluation 2019. *MaxSAT Evaluation 2019* 11.

Reiter, R. 1991. *The Frame Problem in the Situation Calculus: A Simple Solution (sometimes) and a completeness result for goal regression*, 359–380. Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy. San Diego, CA: Academic Press.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. Cambridge, MA: MIT Press.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2012. Conflict-Based Search For Optimal Multi-Agent Path Finding. In *AAAI*. Toronto, Canada.

Sinz, C. 2005. Towards an optimal CNF encoding of boolean cardinality constraints. In *CP*, 827–831. Springer.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Barták, R.; and Boyarski, E. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In Surynek, P.; and Yeoh, W., eds., *SoCS*, 151–159. AAAI Press.

Surynek, P. 2010. An Optimization Variant of Multi-Robot Path Planning Is Intractable. In Fox, M.; and Poole, D., eds., *AAAI*, 1261–1263. AAAI Press.

Surynek, P. 2019. Unifying Search-based and Compilation-based Approaches to Multi-agent Path Finding through Satisfiability Modulo Theories. In Kraus, S., ed., *IJCAI*, 1177–1183.

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 *ECAI*, 810–818. IOS Press.

Wang, K. C.; and Botea, A. 2008. Fast and Memory-Efficient Multi-Agent Pathfinding. In *ICAPS*, 380–387.

Yu, J.; and LaValle, S. M. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *AAAI*. Bellevue, Washington.