Efficient SAT Approach to Multi-Agent Path Finding under the Sum of Costs Objective

Pavel Surynek
Charles University Prague
Malostranské náměstí 25
11800, Praha, Czech Republic
pavel.surynek@mff.cuni.cz

Ariel Felner and **Roni Stern**
Ben Gurion University
Beer-Sheva, Israel 84105
felner,sternron@bgu.ac.il

Eli Boyarski
Bar-Ilan University
Ramat-Gan, Israel
eli.boyarski@gmail.com

Abstract

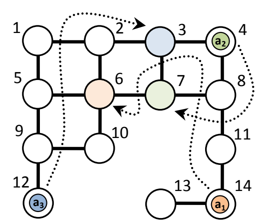
In the *multi-agent path finding* (MAPF) the task is to find non-conflicting paths for multiple agents. In this paper we present the first SAT-solver for the *sum-of-costs* variant of MAPF which was previously only solved by search-based methods. Using both a lower bound on the sum-of-costs and an upper bound on the makespan, we are able to have a reasonable number of variables in our SAT encoding. We then further improve the encoding by borrowing ideas from ICTS, a search-based solver. Experimental evaluation on several domains shown that there are many scenarios where the new SAT-based method outperforms the best variants of previous sum-of-costs search solvers - the ICTS and ICBS algorithms.

1 Introduction and Background

The *multi-agent path finding* (MAPF) problem consists a graph, $G = (V, E)$ and a set $A = \{a_1, a_2, \dots a_m\}$ of m agents. Time is discretized into time steps. The arrangement of agents at time-step t is denoted as α_t. Each agent a_i has a start position $\alpha_0(a_i) \in V$ and a goal position $\alpha_+(a_i) \in V$. At each time step an agent can either *move* to an adjacent empty location[1] or *wait* in its current location. The task is to find a sequence of move/wait actions for each agent a_i, moving it from $\alpha_0(a_i)$ to $\alpha_+(a_i)$ such that agents do not *conflict*, i.e., do not occupy the same location at the same time. Formally, an MAPF instance is a tuple $\Sigma = (G = (V, E), A, \alpha_0, \alpha_+)$. A *solution* for Σ is a sequence of arrangements $\mathcal{S}(\Sigma) = [\alpha_0, \alpha_1, ..., \alpha_\mu]$ such that $\alpha_\mu = \alpha_+$ where α_{t+1} results from valid movements from α_t for $t = 1, 2, ..., \mu - 1$. An example of MAPF and its solution are shown in Figure 1.

MAPF has practical applications in video games, traffic control, robotics etc. (see Sharon et al. (2015) for a survey). The scope of this paper is limited to the setting of *fully cooperative* agents that are centrally controlled. MAPF is usually solved aiming to minimize one of the two commonly-used global cumulative cost functions:

MAPF Σ=(G, {a_1,a_2,a_3}, α_0, α_+)



(Σ)	α_0	α_1	α_2	α_3	α_4	$\alpha_5 = \alpha_+$
a_1	14	11	8	7	6	6
a_2	4	4	4	4	8	7
a_3	12	9	5	1	2	3

Figure 1: Example of MAPF for agents a_1, a_2, and a_3 over a 4-connected grid (left) and its optimal solution (right)

(1) sum-of-costs (denoted ξ) is the summation, over all agents, of the number of time steps required to reach the goal location Dresner and Stone (2008); Standley (2010); Sharon et al. (2013, 2015). Formally, $\xi = \sum_{i=1}^{m} \xi(a_i)$, where $\xi(a_i)$ is an *individual path cost* of agent a_i.
(2) makespan: (denoted μ) is the total time until the last agent reaches its destination (i.e., the maximum of the individual costs) Surynek (2010, 2014a, 2015).

It is important to note that in any solution $\mathcal{S}(\Sigma)$ it holds that $\mu \leq \xi \leq m \cdot \mu$ Thus the optimal *makespan* is usually smaller than the optimal *sum-of-costs*.

Finding optimal solutions for both variants is NP-Hard Yu and LaValle (2013b); Surynek (2010). Therefore, many sub-optimal solvers were developed and are usually used when m is large Ryan (2010); Cohen, Uras, and Koenig (2015); Silver (2005); Röger and Helmert (2012); Khorshid, Holte, and Sturtevant (2011); Wang and Botea (2011)

1.1 Optimal MAPF Solvers

The focus of this paper is on optimal solvers which are divided into two main classes:
(1) Reduction-based solvers. Many recent optimal solvers reduce MAPF to known problems such as CSP Ryan (2010), SAT Surynek (2012), Inductive Logic Programming Yu and LaValle (2013a) and Answer Set Programming Erdem et al. (2013). These papers mostly prove a polynomial-time reduction from MAPF to these problems. These reductions are usually designed for the *makespan* variant of MAPF; they are not applicable for the sum-of-costsvariant.
(2) Search-based solvers. By contrast, many recent opti-

[1]Some variants of MAPF relax the empty location requirement by allowing a chain of neighboring agents to move, given that the head of the chain enters an empty locations. Most MAPF algorithms are robust (or at least easily modified) across these variants.

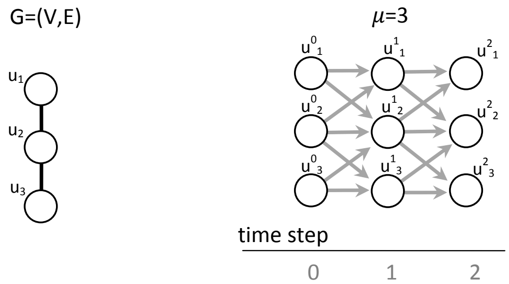


Figure 2: An example of time expansion graph.

mal MAPF solvers are search-based. Some are variants of the A* algorithm on a global *search space* – all different ways to place m agents into V vertices, one agent per vertex Standley (2010); Wagner and Choset (2015). Other employ novel search trees Sharon et al. (2013, 2015); Boyarski et al. (2015). These search-based solvers are usually designed for the *sum-of-costs* MAPF variant.

A major weaknesses is that connection/comparison between different algorithms was usually done only within a given class of algorithms and cost variant but not across these two classes.

1.2 Contributions

This paper aims to start and close the gap. Most of the search-based algorithms can be easily modified to the makespan variant by modifying the cost function and the way the state-space is represented. Some initial directions are given by Sharon et al. (2015). By contrast, the reduction-based algorithms are not trivially modified to the sum-of-costs variant and sometimes a completely new reduction is needed.

In this paper we develop the first SAT-based solvers for the sum-of-costs variant which is based on adding *cardinality constraints* Bailleux and Boufkhad (2003); Silva and Lynce (2007) for bounding the sum-of-costs. We show how to use known lower bounds on the sum-of-costs to reduce the number of variables that encode these cardinality constraints so as to be practicle for current SAT solvers. We then present an *enhanced SAT-solver* which adapts ideas from the ICTS algorithm Sharon et al. (2013) and uses *multi-value decision diagrams* (MDDs) Srinivasan et al. (1990) to further reduce the encoding. Experimental results show that our SAT solver outperforms the best existing search-based solvers for the sum-of-costs variant on many scenarios.

2 SAT Encoding for Optimal Makespan

SAT solvers encompass boolean variables and answer binary questions. The challenge is to apply SAT for MAPF where there is a cumulative cost function. This challenge is stronger for the sum-of-costs variant where each agent has its own cost. We first describe existing SAT encodings for makespan. Then, we present our SAT encoding for sum-of-costs.

A *time expansion graph* (denoted TEG) is a basic concept used in SAT solvers for makespan Surynek (2014a). We use it too in the sum-of-costs variant below. A TEG is a directed acyclic graph (DAG). First, the set of vertices of the underlaying graph G are duplicated for all time-steps from 0 up to the given bound μ. Then, possible actions (move along edges or wait) are represented as directed edges between successive time steps. Figure 2 shows a graph and its TEG for time steps 0, 1 and 2 (vertical layouts). It is important to note that in this example (1) horizonal edges in TEG correspond to *wait* actions. (2) diagonal moves in TEG correspond to real moves. Formally a TEG is defined as follows:

Definition 1. *Time expansion graph of depth μ is a digraph (V, E) where $V = \{u_j^t | t = 0, 1, ..., \mu \wedge u_j \in V\}$ and $E \subseteq \{(u_j^t, u_k^{t+1}) | t = 0, 1, ..., \mu - 1 \wedge (\{u_j, u_k\} \in E \vee j = k)\}$.*

The encoding for MAPF introduces propositional variables and constraints for a single time-step t in order to represent any possible arrangement of agents at time t. Given a desired makespan μ, the formula represents the question of whether there is a solution in the TEG of μ time steps. The search for optimal makespan is done by iteratively incrementing μ (=0, 1, 2...) until a satisfiable formula is obtained. This ensures optimality in case of a solvable MAPF instance. More information on SAT encoding for the makespan variant can be found, e.g. in Surynek (2014a,b,c)

3 Basic-SAT for Optimal Sum-of-costs

The general scheme described above for finding optimal makespan is to convert the optimization problem (finding minimal makespan) to a sequence of decision problems (is there a solution of a given makespan μ). We apply the same scheme for finding optimal sum-of-costs, converting it to a sequence of decision problems – is there a solution of a given sum-of-costs ξ. However, encoding this decision problem is more challenging than the makespan case, because one needs to both bound the sum-of-costs, but also to predict how many time expansions are needed. We address this challenge by using two key techniques descried next: **(1)** Cardinality constraint for bounding ξ and **(2)** Bounding the Makespan.

3.1 Cardinality Constraint for Bounding ξ

The SAT literature offers a technique for encoding a *cardinality constraint* Bailleux and Boufkhad (2003); Silva and Lynce (2007), which allows calculating and bounding a numeric cost within the formula. Formally, for a bound $\lambda \in \mathbb{N}$ and a set of propositional variables $X = \{x_1, x_2, ..., x_k\}$ the *cardinality constraint* $\leq_\lambda \{x_1, x_2, ..., x_k\}$ is satisfied iff the number of variables from the set X that are set to `TRUE` is $\leq \lambda$.

In our SAT encoding, we bound the sum-of-costs by mapping every agent's action to a propositional variable, and then encoding a cardinality constraint on these variables. Thus, one can use the general structure of the makespan SAT encoding (which iterates over possible makespans), and add such a cardinality constraint on top. Next we address the challenge of how to connect these two factors together.

3.2 Bounding the Makespan for the Sum of Costs

Next, we compute how many time expansions (μ) are needed to guarantee that if a solution with sum-of-costs ξ exists then

Algorithm 1: SAT consult

```
1  MAPF-SAT(MAPF Σ = (G = (V, E), A, α0, α+))
2      μ0 = max_{ai∈A} ξ0(ai) ; Δ ← 0
3      while Solution not found do
4          μ ← μ0 + Δ;
5          for each agent ai do
6              build TEG_i(μ);
7          end
8          Solution=Consult-SAT-SOLVER(Σ, μ,Δ);
9          if Solution not found then
10             Δ++;
11         end
12     end
13     return (Solution);
14 end
```

it will be found. In other words, in our encoding, the values we give to ξ and μ must fulfill the following requirement:

R1: *all possible solutions with sum-of-costs ξ must be possible for a makespan of at most μ.*

To find a μ value that meets R1, we require the following definitions. Let $\xi_0(a_i)$ be the shortest individual path for agent a_i, and let $\xi_0 = \sum_{a_i \in A} \xi_0(a_i)$. ξ_0 was called the *sum of individual costs* (SIC) Sharon et al. (2013). ξ_0 is an admissible heuristic for optimal sum-of-costs search algorithms, since ξ_0 is a lower bound on the minimal sum-of-costs. ξ_0 is calculated by relaxing the problem by omitting the other agents. Similarly, we define $\mu_0 = \max_{a_i \in A} \xi_0(a_i)$. μ_0 is length of the *longest* of the shortest individual paths and is thus a lower bound on the minimal makespan. Finally, let Δ be the extra cost over SIC (as done in Sharon et al. (2013)). That is, let $\Delta = \xi - \xi_0$.

Proposition 1. *For makespan μ of any solution with sum-of-costs ξ, R1 holds for $\mu \leq \mu_0 + \Delta$.*

Proof outline: The worst-case scenario, in terms of makespan, is that all the Δ extra moves belong to a single agent. Given this scenario, in the worst case, Δ is assigned to the agent with the largest shortest-path. Thus, the resulting path of that agent would be $\mu_0 + \Delta$, as required. □

Using Proposition 1, we can safely encode the decision problem of whether there is a solution with sum-of-costs ξ by using $\mu = \mu_0 + \Delta$ time expansions, knowing that if a solution of cost ξ exists then it will be found within $\mu = \mu_0 + \Delta$ time expansions. Algorithm 1 summarizes our optimal sum-of-costs algorithm. In every iteration, μ is set to $\mu_0 + \Delta$ (Line 4) and the relevant TEGs (described below) for the various agents are built. Next a decision problem asking whether there is a solution with sum-of-costs ξ and makespan μ is queried (Line 8). The first iteration starts with $\Delta = 0$. If such a solution exists, it is returned. Otherwise ξ is incremented by one, Δ and consequently μ are modified accordingly and another iteration of SAT consulting is activated.

This algorithm clearly terminates for solvable MAPF instances as we start seeking a solution of $\xi = \xi_0$ ($\Delta = 0$) and increment ξ (and Δ) to all possible values. The unsolvability of an MAPF instance can be checked separately by a polynomial-time complete sub-optimal algorithm such as PUSH-AND-ROTATE de Wilde, ter Mors, and Witteveen (2014).

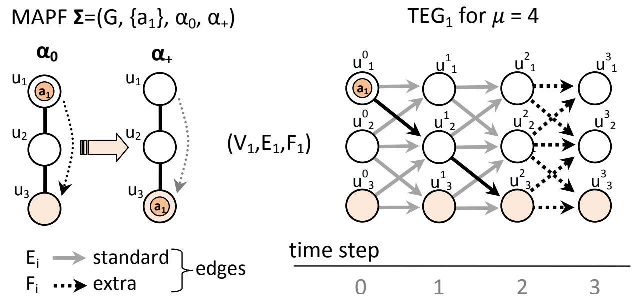


Figure 3: A TEG for an agent that need to go from u_1 to u_3.

3.3 Efficient Use of the Cardinality Constraint

The complexity of encoding a cardinality constraint depends linearly in the number of constrained variables Silva and Lynce (2007); Sinz (2005). Since each agent a_i must move at least $\xi_0(a_i)$, we can reduce the number of variables counted by the cardinality constraint by only counting the variables corresponding to extra movements over the first $\xi_0(a_i)$ movement a_i makes. We implement this by introducing a TEG for a given agent a_i (labeled TEG_i).

TEG_i differs from TEG (Definition 1) in that it distinguishes between two types of edges: E_i and F_i. E_i are (directed) edges whose destination is at time step $\leq \xi_0(a_i)$. These are called *standard edges*. F_i denoted by *extra edges* are directed edges whose destination is at time step $> \xi_0(a_i)$. Figure 3 shows an underlying graph for agent a_1 (left) and the corresponding TEG_1. Note that the optimal solution of cost 2 is denoted by the diagonal path of the TEG. Edges that belong to F_i are those that their destination is time step 3 (dotted lines). The key in this definition is that the cardinality constraint would only be applied to the extra edges, that is, we will only bound the number of extra edges (they sum up to Δ) making it more efficient.

3.4 Detailed Description of the SAT Encoding

Agent a_i must go from its initial position to its goal within TEG_i. This simulates its location in time in the underlying graph G. That is, the task is to find a path from $\alpha_0^0(a_i)$ to $\alpha_+^\mu(a_i)$ in TEG_i. The search for such a path will be encoded within the Boolean formula. Additional constraints will be added to capture all movement constraints such as *collision avoidance* etc. And, of course, we will encode the cardinality constraint that the number of extra edges must be exactly Δ.

We want to ask whether a sum-of-costs solution of ξ exist. For this we build TEG_i for each agent $a_i \in A$ of depth $\mu_0 + \Delta$. We use V_i to denote the set of vertices in TEG_i that agent a_i might occupy during the time steps. Next we introduce the Boolean encoding (denoted BASIC-SAT)which has the following Boolean variables:

1:) $\mathcal{X}_j^t(a_i)$ for every $t \in \{0, 1, ..., \mu\}$ and $u_j^t \in V_i$ – Boolean variable of whether agent a_i is in vertex v_j at time step t.

2:) $\mathcal{E}^t_{j,k}(a_i)$ for every $t \in \{0,1,...,\mu-1\}$ and $(u^t_j, u^{t+1}_k) \in (E_i \cup_i)$ — Boolean variables that model transition of agent a_i from vertex v_j to vertex v_k through any edge (standard or extra) between time steps t and $t+1$ respectively.

3:) $\mathcal{C}^t(a_i)$ for every $t \in \{0,1,...,\mu-1\}$ such that there exist $u^t_j \in V_i$ and $u^{t+1}_k \in V_i$ with $(u^t_j, u^{t+1}_k) \in F_i$ — Boolean variables that model cost of movements along **extra edges** (from F_i) between time steps t and $t+1$.

We now introduce constraints on these variables to restrict illegal values as defined by our variant of MAPF. Other variants may use a slightly different encoding but the principle is the same. Let $T_\mu = \{0,1,...,\mu-1\}$. Several groups of constraints are introduced for each agent $a_i \in A$ as follows:

C1: If an agent appears in a vertex at a given time step, then it must follow through exactly one adjacent edge into the next time step. This is encoded by the following two constraints, which are posted for every $t \in T_\mu$ and $u^t_j \in V_i$

$$\mathcal{X}^t_j(a_i) \Rightarrow \bigvee_{(u^t_j, u^{t+1}_k) \in E_i \cup F_i} \mathcal{E}^t_{j,k}(a_i) \tag{1}$$

$$\bigwedge_{(u^t_j, u^{t+1}_k),(u^t_j, u^{t+1}_l) \in E_i \cup F_i \wedge k<l} \neg\mathcal{E}^t_{j,k}(a_i) \vee \neg\mathcal{E}^t_{j,l}(a_i) \tag{2}$$

C2: Whenever an agent occupies an edge it must also enter it before and leave it at the next time-step. This is ensured by the following constraint introduced for every $t \in T_\mu$ and $(u^t_j, u^{t+1}_k) \in E_i \cup F_i$:

$$\mathcal{E}^t_{j,k}(a_i) \Rightarrow \mathcal{X}^t_j(a_i) \wedge \mathcal{X}^{t+1}_k(a_i) \tag{3}$$

C3: The target vertex of any movement except wait action must be empty. This is ensured by the following constraint introduced for every $t \in T_\mu$ and $(u^t_j, u^{t+1}_k) \in E_i \cup F_i$ such that $j \neq k$.

$$\mathcal{E}^t_{j,k}(a_i) \Rightarrow \bigwedge_{a_l \in A \wedge a_l \neq a_i \wedge u^t_j \in V_l} \neg\mathcal{X}^t_j(a_l) \tag{4}$$

C4: No two agents can appear in the same vertex at the same time step. That is the following constraint is added for every $t \in T_\mu$ and pair of of agents $a_i, a_l \in A$ such that $i \neq l$:

$$\bigwedge_{u^t_j \in V_i \cap V_l} \neg\mathcal{X}^t_j(a_i) \vee \neg\mathcal{X}^t_j(a_l) \tag{5}$$

C5: Whenever an extra edge is traversed the cost needs to be accumulated. In fact, this is the only cost that we accumulate as discussed above. This is done by the following constraint for every $t \in T_\mu$ and extra edge $(u^t_j, u^{t+1}_k) \in F_i$.

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

C6: Cardinality constraint. Finally the bound on the total cost needs to be introduced. Reaching the sum-of-costs of ξ corresponds to traversing exactly Δ extra edges from F_i. The following cardinality constrains ensures this:

$$\leq_\Delta \left\{ \begin{array}{c} \mathcal{C}^t(a_i) | i = 1,2,...,n \wedge t = 0,1,...,\mu-1 \\ \wedge\{(u^t_j, u^{t+1}_k) \in F_i\} \neq \emptyset \end{array} \right\} \tag{7}$$

Final formula. The resulting Boolean formula that is a conjunction of $C1 \ldots C6$ will be denoted as $\mathcal{F}_{BASIC}(\Sigma,\mu,\Delta)$ and is the one that is consulted by Algorithm 1 (line 4).

The following proposition summarizes the correctness of our encoding.

Proposition 2. *MAPF $\Sigma = (G = (V,E), A, \alpha_0, \alpha_+)$ has a sum-of-costs solution of ξ if and only if $\mathcal{F}_{BASIC}(\Sigma,\mu,\Delta)$ is satisfiable. Moreover, a solution of MAPF Σ with the sum-of-costs of ξ can be extracted from the satisfying valuation of $\mathcal{F}_{BASIC}(\Sigma,\mu,\Delta)$ by reading its $\mathcal{X}^t_j(a_i)$ variables.*

Proof: The direct consequence of the above definitions is that a valid solution of a given MAPF Σ corresponds to non-conflicting paths in the TEGs of the individual agents. These non-conflicting paths further correspond to satisfying the variable assignment of $\mathcal{F}_{BASIC}(\Sigma,\mu,\Delta)$, i.e., that there are Δ extra edges in TEGs of depth $\mu = \mu_0 + \Delta$. □

Proposition 3. *Let D be the maximal degree of any vertex in G and let m be the number of agents. If $m \cdot |E| \geq \Delta$ and $m \geq D$ then the number of clauses in $\mathcal{F}_{BASIC}(\Sigma,\mu,\Delta)$ is $O(\mu \cdot m^2 \cdot |E|)$, and the number of variables is $O(\mu \cdot |E| \cdot m)$.*

Proof: The components of $\mathcal{F}_{BASIC}(\Sigma,\mu,\Delta)$ is described in equations 1– 7. Equation 1 introduces at most $O(m \cdot \mu \cdot |E|)$ clauses. Equation 2 introduces at most $O(m \cdot \mu|E| \cdot D)$ clauses. Equation 3 introduces at most $O(m \cdot \mu \cdot |E|)$ clauses. Equation 4 introduces at most $O(m^2 \cdot \mu \cdot |E|)$ Equation 5 introduces at most $O(m^2 \cdot \mu \cdot |V|)$ clauses. Equation 6 introduces at most $O(m \cdot \mu \cdot |E|)$ clauses. Equation 7 introduces at most $O(m \cdot \mu \cdot (\xi - \xi_0))$ clauses, since a cardinality constraint checking that n variables has a cardinality constraint of m requires $O(n \cdot m)$ clauses Sinz (2005). Summing all the above results in a total of $O(\mu \cdot m \cdot (|E| \cdot (D+m) + (\xi - \xi_0)))$. If we assume that $m > D$ and that $m \cdot |E| > (\xi - \xi_0)$ then the number of clauses is $O(\mu \cdot m^2 \cdot |E|)$. The number of variables is easily computed in a similar way. □

4 Improving Basic SAT by Adding MDDs

A major parameter that affects the speed of solving of Boolean formulae is their size Petke (2015). The size of formulae in the BASIC-SAT encoding is affected mostly by the size of the TEGs (this is embodied in the $|E|$ factor in the encoding size). To obtain a significant speedup we reduce the size of TEG_i for agent a_i in terms of number of vertices while the soundness of encoding is preserved.

Let TEG^μ_i denote TEG_i for μ time expansions. We set $\mu = \mu_0 + \Delta$ in our solution. The data structure we use for reducing TEG^μ_i is a *multi-value Decision Diagram* (MDD). MDDs were already used in the search-based MAPF algorithm ICTS Sharon et al. (2013). In our context, MDD^μ_i is a digraph that represents all possible valid paths from $\alpha_0(a_i)$ to $\alpha_+(a_i)$ of cost μ for agent a_i. MDD^μ_i has a single *source node* at level 0 and a single *sink node* at level μ. Every node at depth t of MDD^μ_i corresponds to a possible location of a_i at time t, that is on a path of cost μ from $\alpha_0(a_i)$ to $\alpha_+(a_i)$. It is easy to see that MDD^μ_i is subgraph of TEG_i. While TEG^μ_i includes all vertices of G at each time step, MDD^μ_i includes only those vertices and edges that represent possible valid paths, and thus vertices not in MDD^μ_i can be ignored.

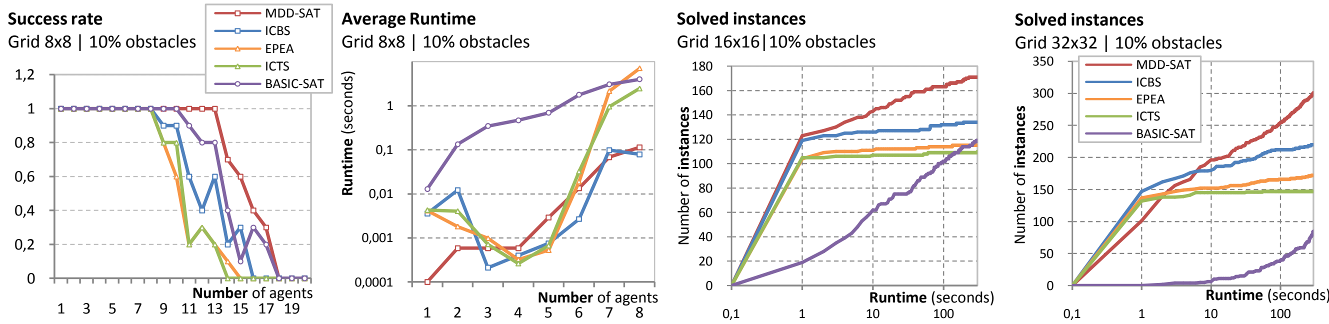


Figure 5: Results on 8×8 grid (left). Number of solved instances in the given runtime on 16×16 and 32×32 grids. (right)

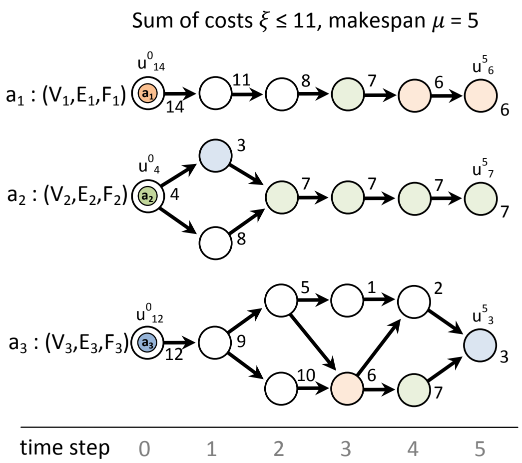


Figure 4: MDDs for agents a_1, a_2, and a_3 for the MAPF from Figure 1 for *sum of individual cost* $\xi \leq 11$

Moreover, the maximum cost that can be consumed by single agent a_i under given sum-of-costs bound ξ is $\xi_0(a_i) + \Delta$ where, as defined above, $\xi_0(a_i)$ is the shortest path connecting $\alpha_0(a_i)$ with $\alpha_+(a_i)$ in G (assuming no other agent exist). Thus, it is sufficient to replace TEG_i^{μ} with $MDD_i^{\xi_0(a_i)+\Delta}$, which is useful since $\xi_0(a_i) + \Delta \leq \mu_0 + \Delta = \mu$.

MDDs for the agents of Figure 1 are shown in Figure 4. Indeed, the size of the MDDs is much smaller than the corresponding TEGs which include all states for all time steps.

The encoding that uses MDD-based time expansion will be called MDD-SAT and the corresponding formulae will be denoted as $\mathcal{F}_{MDD}(\Sigma, \mu, \Delta)$. $\mathcal{F}_{MDD}(\Sigma, \mu, \Delta)$ are similar to BASIC-SAT. The only different is that in BASIC-SAT there is a variable for all vertices and edges of the TEGs while in MDD-SAT, only variables for the vertices and edges of the MDDs are needed. This difference can be significant. Table 1 presents the number of propositional variables and clauses accumulated over all the constructed formulae for a given MAPF instance for BASIC-SAT and for MDD-SAT over 8×8 grid with 10% obstacles. The average values out of 10 random instances per number of agents is shown. Up to two orders of magnitude reduction is shown.

5 Experimental Evaluation

We experimented on 4-connected grids with randomly placed obstacles Silver (2005); Standley (2010) and on *Dragon Age* maps Sturtevant (2012). Both settings are a standard MAPF benchmarks. The initial position of the agents was randomly selected. To ensure solvability the goal positions were selected by performing a long *random walk* from the initial arrangement.

We compared our SAT solvers to several state-of-the-art search-based algorithms: the *increasing cost tree search* - ICTS Sharon et al. (2013), *Enhanced Partial Expansion A** - EPEA* Goldenberg et al. (2014) and *improved conflict-based search* - ICBS Boyarski et al. (2015). For all the search algorithms we used the best known setup of their parameters and enhancements suitable for solving the given instances.

The SAT approaches were implemented in C++ using `Glucose 3.0` Audemard and Simon (2009); Audemard, Lagniez, and Simon (2013); a top performing SAT solver in the *SAT Competition* Järvisalo et al. (2012); Surynek (2014a). The cardinality constraint was encoded using a simple standard circuit based encoding called *sequential counter* Sinz (2005). ICTS and ICBS were implemented in C#, based on their original implementation. All experiments were performed on a Xeon 2Ghz, and on Phenom II 3.6Ghz, both with 12 Gb of memory.

5.1 Square Grid Experiments

We first experimented on 8×8, 16×16, and 32×32 grids with 10% obstacles while varying the number of agents from 1 to 20. Figure 5 presents results over 10 instances where each algorithm was given a time limit of 300 seconds. The leftmost plot shows the *success rate* (=precantage of instances solved within the time limit) as a function of the number of agents. The next plot reports the average runtime for instances that were solved by all algorithms. The right plots visualize the results on 16×16 and 32×32 grids but

Grid 8x8	BASIC-SAT		MDD-SAT	
m	*Variables*	*Clauses*	*Variables*	*Clauses*
1	1 552.8	11 617.6	20.6	27.9
4	14 712.0	127 732.2	276.5	554.0
8	226 391.2	2 099 127.6	18 355.6	68 826.0
16	4 075 187.2	32 108 347.2	2 253 508.2	13 128 646.9

Table 1: The number of variables and clauses

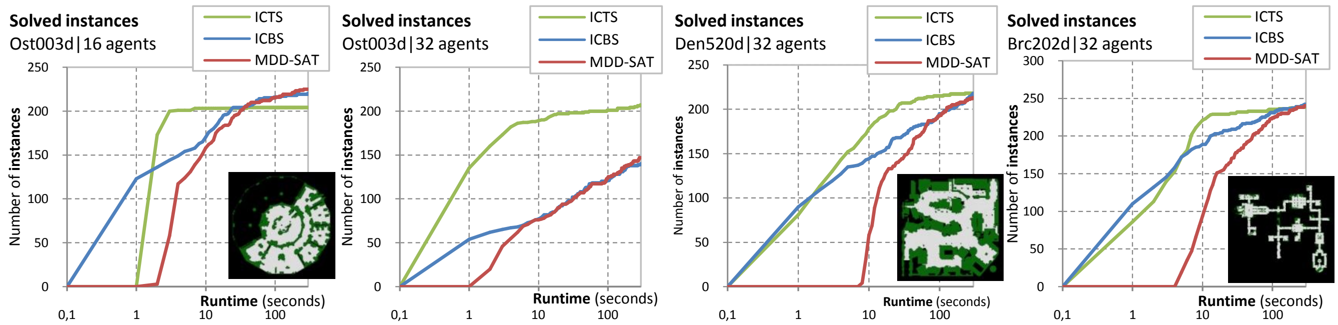


Figure 6: The number of solved instances in the given runtime on Dragon Age maps for 16 and 32 agents.

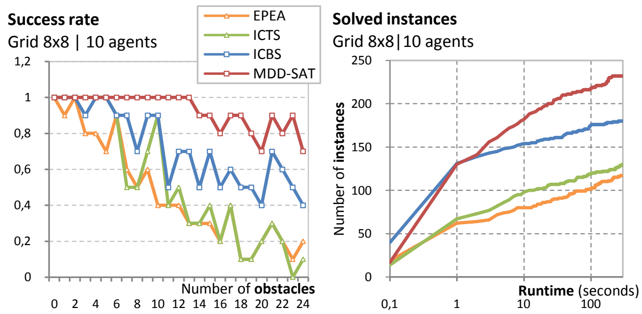


Figure 7: Success rate and runtime on the 8×8 grid with increasing number of obstacles (out of 64 cells).

in a different way. Here, we present the number of instances (out of all 200 instances for all number of agents) that each method solved (y-axis) as a function of the elapsed time (x-axis).

The first clear trend is that MDD-SAT significantly outperforms BASIC-SAT in all aspects. This shows the importance of developing efficient SAT encodings for this problem. In addition, a prominent trend observed in all the plots is that MDD-SAT has higher success rate and solves more instances than all other algorithms. In some cases, however, where the available runtime is very small, MDD-SAT is outperformed by the search-based algorithms.

For the rest of our experiments, we only evaluated the most efficient algorithms, namely, MDD-SAT, ICTS, and ICBS.

Next, we varied the number of obstacles for the 8×8 grid with 10 agents. Results are shown in Figure 7. Clearly MDD-SAT can solve more instances over all settings. MDD-SAT was always faster except for some easy instances where ICBS was slightly faster. Interestingly, increasing the number of obstacles reduces the number of open cells. This is an advantage for the SAT solver as the SAT formula has less variables. By contrast, for the search-based solvers, adding obstacles means that the graphs gets denser and harder to solve.

5.2 Results on the Dragon Age Maps

Next, we experimented on three Dragon-Age maps (*ost003d*, *den520d*, and *brc202d*) commonly used as testbeds. In these maps there is a large number of open cells but the graph is sparse with agents. This gives a clear advantage to the search-based solvers. To obtain instances of various difficulties we varied the distance between start and goal locations. 10 random instances were generated for each distance in the range: $\{8, 16, 24, \dots, 200\}$. The results are shown in Figure 6 (the number of instances solved as the function of time).

In the Dragon-Age setting there is no universal winner. Each algorithm was the best for some of the instances (especially in case of *ost003d*). When limited time is allowed ICTS or ICBS are better. However, given enough time MDD-SAT catches up and even outperforms the other algorithms. This was evident in all these experiments except for *ost003d* with 32 agents. Concrete runtimes for 10 instances of *ost003d* are given in Table 2. MDD-SAT solves the hardest instance (#1) while other solvers ran out of time. The right part of the table illustrates the cumulative size of the formulae generated during the solving process. Although the map is much larger than the square grids, the size of formulae is comparable to the densely occupied grid (see Figure 1). This is because ξ_0 is a good lower bound of the optimal cost in the sparse maps.

The entire set of experiments show a clear trend. When a small amount of time is given the search-based algorithm may be faster. But, given enough time MDD-SAT is the correct choice, even in the large maps where it has an initial disadvantage. One of the reasons for this is modern SAT solvers have the ability to learn and improve their speed during the process of answering a SAT question. But, this learning needs sufficient time and large search trees to be effective. By contrast, search algorithms do not have this advantage.

6 Summary and Conclusions

We introduced the first state-of-the-art SAT-based solver for the sum-of-costs variant of MAPF. The resulting encoding, called MDD-SAT, was shown to be competitive in comparison with the state-of-the-art search-based solvers over a variety of domains. Nevertheless, as previous authors mentioned Sharon et al. (2015); Boyarski et al. (2015) there is no universal winner and each of the approaches has pros and cons and worsk best in different circumstances. This calls for a deeper study of various classes of MAPF instances and their characteristics.

There are several factors behind the performance of the SAT-based approach: clause learning, constraint propagation, good implementation of the SAT solver. On the other hand, the SAT solver doesn't understand the structure of the encoded problem which may downgrade the performance. Hence, we consider that implementing techniques such as learning directly into the dedicated MAPF solver may be a future direction.

MAPF	**Ost003d** (seconds) 16 agents, distance=168		
	MDD-SAT	*ICBS*	*ICTS*
1	**101.4**	N/A	N/A
2	12.8	9.7	**2.4**
3	13.2	4.4	**2.4**
4	3.8	**0.6**	1.2
5	13.5	9.6	**3.2**
6	22.7	**10.7**	N/A
7	N/A	N/A	N/A
8	36.9	49.6	**2.5**
9	12.0	2.6	**1.4**
10	N/A	N/A	N/A

m	**MDD-SAT,** 16 agents	
Distance	*Variables*	*Clauses*
8	758.0	1 169.7
64	34 648.7	120 961.1
128	932 440.9	9 128 568.8

m	**MDD-SAT**, 32 agents	
Distance	*Variables*	*Clauses*
8	2 377.6	3 751.3
64	571 915.1	3 672 249.3
128	5 163 157.0	49 201 960.0

Table 2: Runtime for 10 instances (left) and the average size of the MDD-SAT formulae for `ost003d` (right)

References

Audemard, G., and Simon, L. 2009. Predicting learnt clauses quality in modern SAT solvers. In *IJCAI*, 399–404.

Audemard, G.; Lagniez, J.; and Simon, L. 2013. Improving glucose for incremental SAT solving with assumptions: Application to MUS extraction. In *Theory and Applications of Satisfiability Testing - SAT 2013*, 309–317.

Bailleux, O., and Boufkhad, Y. 2003. Efficient CNF encoding of boolean cardinality constraints. In *CP*, 108–122.

Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Tolpin, D.; Betzalel, O.; and Shimony, S. 2015. ICBS: improved conflict-based search algorithm for multi-agent pathfinding. In *IJCAI*, 740–746.

Cohen, L.; Uras, T.; and Koenig, S. 2015. Feasibility study: Using highways for bounded-suboptimal mapf. In *SOCS*, 2–8.

de Wilde, B.; ter Mors, A.; and Witteveen, C. 2014. Push and rotate: a complete multi-agent pathfinding algorithm. *JAIR* 51:443–492.

Dresner, K., and Stone, P. 2008. A multiagent approach to autonomous intersection management. *JAIR* 31:591–656.

Erdem, E.; Kisa, D. G.; Oztok, U.; and Schueller, P. 2013. A general formal framework for pathfinding problems with multiple agents. In *AAAI*.

Goldenberg, M.; Felner, A.; Stern, R.; Sharon, G.; Sturtevant, N.; Holte, R.; and Schaeffer, J. 2014. Enhanced partial expansion A*. *JAIR* 50:141–187.

Järvisalo, M.; Berre, D. L.; Roussel, O.; and Simon, L. 2012. The international SAT solver competitions. *AI Magazine* 33(1).

Khorshid, M. M.; Holte, R. C.; and Sturtevant, N. R. 2011. A polynomial-time algorithm for non-optimal multi-agent pathfinding. In *Symposium on Combinatorial Search (SOCS)*.

Petke, J. 2015. *Bridging Constraint Satisfaction and Boolean Satisfiability*. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer.

Röger, G., and Helmert, M. 2012. Non-optimal multi-agent pathfinding is solved (since 1984). In *SOCS)*.

Ryan, M. 2010. Constraint-based multi-robot path planning. In *ICRA*, 922–928.

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2013. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence* 195:470–495.

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

Silva, J., and Lynce, I. 2007. Towards robust CNF encodings of cardinality constraints. In *CP*, 483–497.

Silver, D. 2005. Cooperative pathfinding. In *AIIDE*, 117–122.

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

Srinivasan, A.; Ham, T.; Malik, S.; and Brayton, R. 1990. Algorithms for discrete function manipulation. In *(ICCAD*, 92–95.

Standley, T. 2010. Finding optimal solutions to cooperative pathfinding problems. In *AAAI*, 173–178.

Sturtevant, N. R. 2012. Benchmarks for grid-based pathfinding. *Computational Intelligence and AI in Games* 4(2):144–148.

Surynek, P. 2010. An optimization variant of multi-robot path planning is intractable. In *AAAI*.

Surynek, P. 2012. Towards optimal cooperative path planning in hard setups through satisfiability solving. In *PRICAI*. 564–576.

Surynek, P. 2014a. Compact representations of cooperative pathfinding as SAT based on matchings in bipartite graphs. In *ICTAI*, 875–882.

Surynek, P. 2014b. A simple approach to solving cooperative pathfinding as propositional satisfiability works well. In *PRICAI*, 827–833.

Surynek, P. 2014c. Simple direct propositional encoding of cooperative path finding simplified yet more. In *MICAI*, 410–425.

Surynek, P. 2015. Reduced time-expansion graphs and goal decomposition for solving cooperative path finding sub-optimally. In *IJCAI*, 1916–1922.

Wagner, G., and Choset, H. 2015. Subdimensional expansion for multirobot path planning. *Artif. Intell.* 219:1–24.

Wang, K., and Botea, A. 2011. MAPP: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. *JAIR)* 42:55–90.

Yu, J., and LaValle, S. 2013a. Planning optimal paths for multiple robots on graphs. In *ICRA*, 3612–3617.

Yu, J., and LaValle, S. M. 2013b. Structure and intractability of optimal multi-robot path planning on graphs. In *AAAI*.