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,sterron@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 \( A = \{a_1, a_2, \ldots, a_m\} \) of \( m \) agents. Time is discretized into time steps. The arrangement of agents at time-step \( t \) is denoted as \( \alpha_t \). 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 \( \Sigma \) is a sequence of arrangements \( S(\Sigma) = [\alpha_0, \alpha_1, \ldots, \alpha_\mu] \) such that \( \alpha_\mu = \alpha_+ \) where \( \alpha_{t+1} \) results from valid movements from \( \alpha_t \) for \( t = 1, 2, \ldots, \mu - 1 \). 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:

\(^1\)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.

(1) sum-of-costs (denoted \( \xi \)) 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 \( \mu \)) 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 \( S(\Sigma) \) 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); Khoshid, 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-costs variant.
(2) Search-based solvers. By contrast, many recent opti-
In this paper we develop the first SAT-based solvers for the sum-of-costs variant below. A TEG is a directed acyclic graph (DAG). First, the set of vertices of the underlying graph $G$ are duplicated for all time-steps from 0 up to the given bound $\mu$. 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) horizontal 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 $\mu$ is a digraph $(V, E)$ where $V = \{u^*_j|t = 0, 1, \ldots, \mu \land u_j \in V\}$ and $E \subseteq \{(u^*_j, u^*_{k+1})|t = 0, 1, \ldots, \mu - 1 \land \{(u_j, u_k) \in E \land j = k\}$.

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 $\mu$, the formula represents the question of whether there is a solution in the TEG of $\mu$ time steps. The search for optimal makespan is done by iteratively incrementing $\mu = 0, 1, 2, \ldots$ 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 $\mu$). 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 $\xi$. 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 described next: (1) Cardinality constraint for bounding $\xi$ and (2) Bounding the Makespan.

#### 3.1 Cardinality Constraint for Bounding $\xi$

The SAT literature offers a technique for encoding a cardinality constraint (Baileux 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 $X = \{x_1, x_2, \ldots, x_k\}$ the cardinality constraint $\leq \lambda \{x_1, x_2, \ldots, x_k\}$ is satisfied if 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 ($\mu$) are needed to guarantee that if a solution with sum-of-costs $\xi$ exists then...
it will be found. In other words, in our encoding, the values we give to $\xi$ and $\mu$ must fulfill the following requirement:

R1: all possible solutions with sum-of-costs $\xi$ must be possible for a makespan of at most $\mu$.

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

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

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

Using Proposition 1, we can safely encode the decision problem of whether there is a solution with sum-of-costs $\xi$ by using $\mu = \mu_0 + \Delta$ time expansions, knowing that if a solution of cost $\xi$ 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, $\mu$ 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 $\xi$ and makespan $\mu$ is queried (Line 8). The first iteration starts with $\Delta = 0$. If such a solution exists, it is returned. Otherwise $\xi$ is incremented by one, $\Delta$ and consequently $\mu$ 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 $\xi$ (and $\Delta$) 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).

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 $\Delta$) 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 $a_0^0(a_i)$ to $a_i^\mu(a_i)$ 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 $\Delta$.

We want to ask whether a sum-of-costs solution of $\xi$ 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. $X_i^t(a_i)$ for every $t \in \{0, 1, ..., \mu\}$ and $v_j \in V_i$ – Boolean variable of whether agent $a_i$ is in vertex $v_j$ at time step $t$. 

---

**Algorithm 1: SAT consult**

```plaintext
1 MAPF-SAT(MAPF $\Sigma = (G = (V, E), A, \alpha_0, \alpha_+))$
2 $\mu_0 = \max_{a_i \in A} \xi_0(a_i); \Delta \leftarrow 0$
3 while Solution not found do
4     $\mu \leftarrow \mu_0 + \Delta$
5     for each agent $a_i$ do
6         build $TEG_i(\mu)$
7     end
8     Solution = Consult-SAT-SOLVER($\Sigma, \mu, \Delta$)
9     if Solution not found then
10        $\Delta \leftarrow \Delta + 1$
11     end
12 end
13 return (Solution);
```

---
2) \( \mathcal{E}^t_{j,k}(a_i) \) for every \( t \in \{0, 1, \ldots, \mu - 1\} \) and \( (u^t_j, u^{t+1}_k) \in (E_i \cup F_i) \) — 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 \( t \in \{0, 1, \ldots, \mu - 1\} \) 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 variables may use a slightly different encoding but the principle is the same. Let \( T^\mu = \{0, 1, \ldots, \mu - 1\} \). 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 \( u^t_j \in V_i \):

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

\[
\bigwedge_{a_i \in A \, a_i \neq a_j, \, u^t_j \in V_i} \neg \mathcal{E}^t_{j,k}(a_i) \tag{2}
\]

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 \( (u^t_j, u^{t+1}_k) \in E_i \cup F_i \):

\[
\mathcal{E}^t_{j,k}(a_i) \Rightarrow \mathcal{X}^t_j(a_i) \land \mathcal{X}^{t+1}_{j+1}(a_i) \tag{3}
\]

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 \( j \neq k \):

\[
\mathcal{E}^t_{j,k}(a_i) \Rightarrow \bigwedge_{a_i \in A \, a_i \neq a_j, \, u^t_j \in V_i} \neg \mathcal{X}^t_j(a_i) \tag{4}
\]

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 agents \( a_i, a_j \in A \) such that \( i \neq j \):

\[
\bigwedge_{u^t_j \in V_i \cap V_j} \neg \mathcal{X}^t_j(a_i) \lor \neg \mathcal{X}^t_j(a_j) \tag{5}
\]

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 \( \xi \) corresponds to traversing exactly \( \Delta \) extra edges from \( F_i \). The following cardinality constraints ensures this:

\[
\leq \Delta \left\{ \mathcal{C}^t(a_i) | i = 1, 2, \ldots, n \land t = 0, 1, \ldots, \mu - 1 \land \{(u^t_j, u^{t+1}_k) \in F_i \} \neq \emptyset \right\} \tag{7}
\]

**Final formula.** The resulting Boolean formula that is a conjunction of \( C1 \) through \( C6 \) 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.** \( \text{MAPF} \Sigma = (G = (V,E), A, \alpha_0, \alpha_+ \) has a sum-of-costs solution of \( \xi \) if and only if \( \mathcal{F}_{BASIC}(\Sigma, \mu, \Delta) \) is satisfiable. Moreover, a solution of \( \text{MAPF} \Sigma \) with the sum-of-costs of \( \xi \) can be extracted from the satisfying valuation of \( \mathcal{F}_{BASIC}(\Sigma, \mu, \Delta) \) by reading its \( \mathcal{X}^t(a_i) \) variables.

**Proof:** The direct consequence of the above definitions is that a valid solution of a given MAPF \( \Sigma \) 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 \( \Delta \) 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 \( m \geq 2 \) 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 \( O(\mu \cdot |E|) \) clauses. Equation 2 introduces at most \( O(m \cdot |E| \cdot D) \) clauses. Equation 3 introduces at most \( O(m \cdot |E|) \) clauses. Equation 4 introduces at most \( O(m^2 \cdot |E|) \) clauses. Equation 5 introduces at most \( O(m^2 \cdot |V|) \) clauses. Equation 6 introduces at most \( O(m \cdot |E|) \) clauses. Equation 7 introduces at most \( O(m \cdot |E| \cdot (\xi - \xi_0)) \) 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 \( TEG_m^i \) denote \( TEG_i \) for \( \mu \) time expansions. We set \( \mu = \mu_0 + \Delta \) in our solution. The data structure we use for reducing \( TEG_m^i \) 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, \( MDD_m^i \) is a digraph that represents all possible valid paths from \( \alpha_0(a_i) \) to \( \alpha_+(a_i) \) of cost \( \mu \) for agent \( a_i \). \( MDD_m^i \) has a single source node at level 0 and a single sink node at level \( \mu \). Every node at depth \( t \) of \( MDD_m^i \) corresponds to a possible location of \( a_i \) at time \( t \), that is on a path of cost \( \mu \) from \( \alpha_0(a_i) \) to \( \alpha_+(a_i) \). It is easy to see that \( MDD_m^i \) is subgraph of \( TEG_i \). While \( TEG_m^i \) includes all vertices of \( G \) at each time step, \( MDD_m^i \) includes only those vertices and edges that represent possible valid paths, and thus vertices not in \( MDD_m^i \) can be ignored.
Moreover, the maximum cost that can be consumed by single agent \( a_i \) under given sum-of-costs bound \( \xi \) 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 \( TEG_i^a \) 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 \( F_{MDD}(\Sigma, \mu, \Delta) \). \( F_{MDD}(\Sigma, \mu, \Delta) \) 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 \( \times \) 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.

<table>
<thead>
<tr>
<th>Grid 8x8</th>
<th>BASIC-SAT</th>
<th>MDD-SAT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Variables</td>
<td>Clauses</td>
</tr>
<tr>
<td>1</td>
<td>1552.8</td>
<td>11617.6</td>
</tr>
<tr>
<td>4</td>
<td>14712.0</td>
<td>127732.2</td>
</tr>
<tr>
<td>8</td>
<td>226391.2</td>
<td>2099127.6</td>
</tr>
<tr>
<td>16</td>
<td>4075187.2</td>
<td>32108347.2</td>
</tr>
</tbody>
</table>

Table 1: The number of variables and clauses

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 Jäarvisalo 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 \( \times \) 8, 16 \( \times \) 16, and 32 \( \times \) 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 (=percent 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 \( \times \) 16 and 32 \( \times \) 32 grids but
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 $8 \times 8$ 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 get 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: $\{8, 16, 24, \ldots, 200\}$. 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 formula 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 $c_0$ 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.

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

<table>
<thead>
<tr>
<th>MDD-SAT</th>
<th>Variables</th>
<th>Distance</th>
<th>Clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>101.4</td>
<td>N/A</td>
<td>8</td>
<td>758.0</td>
</tr>
<tr>
<td>12.8</td>
<td>9.7</td>
<td>64</td>
<td>34 648.7</td>
</tr>
<tr>
<td>13.2</td>
<td>4.4</td>
<td>128</td>
<td>932 440.9</td>
</tr>
<tr>
<td>3.8</td>
<td>0.6</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>13.5</td>
<td>9.6</td>
<td>3.2</td>
<td>1.2</td>
</tr>
<tr>
<td>22.7</td>
<td>10.7</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>36.9</td>
<td>49.6</td>
<td>2.5</td>
<td>8</td>
</tr>
<tr>
<td>12.0</td>
<td>2.6</td>
<td>1.4</td>
<td>64</td>
</tr>
<tr>
<td>163</td>
<td>157.0</td>
<td>49</td>
<td>201 960.0</td>
</tr>
</tbody>
</table>

References


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