Journal of Artificial Intelligence Research 73 (2022) 553-618 Submitted 09/2021; published 02/2022

Migrating Techniques from Search-Based Multi-Agent Path Finding Solvers to SAT-Based Approach

Pavel Surynek PAVEL.SURYNEK@FIT.CVUT.CZ
Faculty of Information Technology, Czech Technical University in Prague
Thákurova 9, 160 00 Praha 6, Czechia

Roni Stern STERNRON@BGU.AC.IL
Ben Gurion University, Beer-Sheva 84105, Israel
Palo Alto Research Center, CA, USA

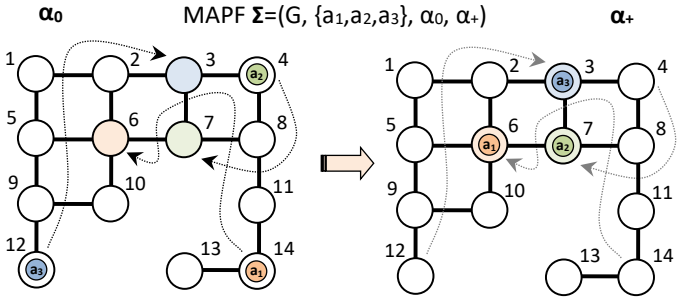
Eli Boyarski BOYARSKE@POST.BGU.AC.IL
Ariel Felner FELNER@BGU.AC.IL
Ben Gurion University, Beer-Sheva 84105, Israel

Abstract

In the *multi-agent path finding* problem (MAPF) we are given a set of agents each with respective start and goal positions. The task is to find paths for all agents while avoiding collisions, aiming to minimize a given objective function. Many MAPF solvers were introduced in the past decade for optimizing two specific objective functions: *sum-of-costs* and *makespan*. Two prominent categories of solvers can be distinguished: *search-based* solvers and *compilation-based* solvers. Search-based solvers were developed and tested for the sum-of-costs objective, while the most prominent compilation-based solvers that are built around Boolean satisfiability (SAT) were designed for the makespan objective. Very little is known on the performance and relevance of solvers from the compilation-based approach on the sum-of-costs objective. In this paper, we start to close the gap between these cost functions in the compilation-based approach. Our main contribution is a new SAT-based MAPF solver called MDD-SAT, that is directly aimed to optimally solve the MAPF problem under the sum-of-costs objective function. Using both a lower bound on the sum-of-costs and an upper bound on the makespan, MDD-SAT is able to generate a reasonable number of Boolean variables in our SAT encoding. We then further improve the encoding by borrowing ideas from ICTS, a search-based solver. In addition, we show that concepts applicable in search-based solvers like ICTS and ICBS are applicable in the SAT-based approach as well. Specifically, we integrate *independence detection*, a generic technique for decomposing an MAPF instance into independent subproblems, into our SAT-based approach, and we design a relaxation of our optimal SAT-based solver that results in a bounded suboptimal SAT-based solver. Experimental evaluation on several domains shows that there are many scenarios where our SAT-based methods outperform state-of-the-art sum-of-costs search-based solvers, such as variants of the ICTS and ICBS algorithms.

1. Introduction and Background

The *multi-agent path finding* (MAPF) problem consists of an undirected graph, $G = (V, E)$ with $V = \{u_1, u_2, ..., u_n\}$ and a set $A = \{a_1, a_2, \ldots a_k\}$ of k agents. Time is discretized into time steps. The configurations of agents at time-step t are denoted as $\alpha_t : A \rightarrow V$. 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*



S(Σ)	α0	α1	α2	α3	α4	α5= α+
a1	14	11	8	7	6	6
a2	4	4	4	4	8	7
a3	12	9	5	1	2	3

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

to an adjacent empty position in the previous step [1] or *wait* in its current position. 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 position 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 configurations $\pi = [\alpha_0, \alpha_1, ..., \alpha_\mu]$ such that $\alpha_\mu = \alpha_+$ where α_{t+1} results from valid movements from α_t for $t = 1 = 0, 1, ..., \mu - 1$. An example of MAPF and its solution are shown in Figure 1.

MAPF has practical applications in video games, traffic control, robotics etc., please see a survey for more applications (Sharon, Stern, Felner, & Sturtevant, 2015). 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) sum-of-costs (denoted ξ) is the summation, over all agents, of the number of time steps required to reach the goal position (Dresner & Stone, 2008; Standley, 2010b; Sharon, Stern, Goldenberg, & Felner, 2013; Sharon et al., 2015). Formally, $\xi = \sum_{i=1}^{k} \xi(a_i)$, 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 $\mu \leq \xi \leq k \cdot \mu$. Similarly it holds that the optimal *makespan* is smaller or equal to the optimal *sum-of-costs* since otherwise the inequalities would be violated.

Intuitively, sum-of-costs can be regarded as the total energy consumption of all agents such that at each time step spent before reaching the goal the agent consumes one unit of energy. In this respect, it is not surprising that optimization of one of these two objectives goes against the other - total time can be saved at the cost of increased energy consumption and vice versa. An example of an MAPF instance where any makespan-optimal solution has sum-of-costs that is greater than the optimum and any sum-of-costs optimal solution has makespan that is greater than the optimal makespan is shown in Figure 2.

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

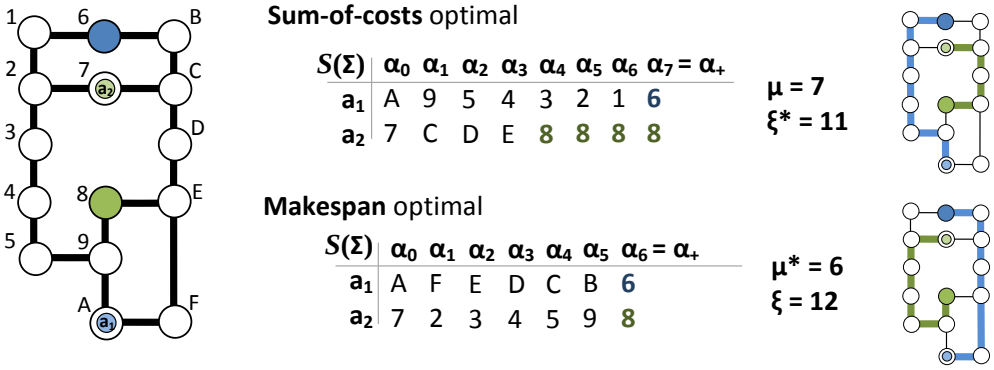


Figure 2: An instance of MAPF where *makespan* and *sum-of-costs* optimal solutions differ - that is, any makespan-optimal solution is strictly sum-of-costs suboptimal and any sum-of-costs optimal solution is strictly makespan suboptimal.

Finding optimal solutions for both variants with any standard style of agents' movement is NP-hard even on planar graphs (Ratner & Warmuth, 1986, 1990; Surynek, 2010, 2015; Yu & LaValle, 2013b; Yu, 2016). Therefore, many suboptimal solvers were developed, and are usually used, when the number of agents or the graph is very large (Cohen, Uras, & Koenig, 2015; Khorshid, Holte, & Sturtevant, 2011; Röger & Helmert, 2012; Ryan, 2010; Silver, 2005; Wang & Botea, 2011). In contrast to the difficulty of finding optimal solutions, finding any feasible solution or detecting unsolvability of a given instance can be done in polynomial time (de Wilde, ter Mors, & Witteveen, 2014; Kornhauser, Miller, & Spirakis, 1984; Luna & Bekris, 2011; Surynek, 2009, 2014c).

1.1 Optimal MAPF Solvers

Many optimal solvers were introduced in the past decade, most of which focus on one of these cost functions:

- **(1) Sum-of-costs.** Most optimal MAPF solvers that optimize this cost function are based on search. Some of these search-based solvers are variants of the A* algorithm on a global *search space* in which all different ways to place k agents into V vertices, one agent per vertex, are considered (Standley, 2010b; Wagner & Choset, 2015). Others employ novel search trees (Boyarski, Felner, Stern, Sharon, Tolpin, Betzalel, & Shimony, 2015; Sharon et al., 2015, 2013). Search-based solvers feature various search-space pruning techniques like *independence detection* (ID) (Standley, 2010b) or *multi-value decision diagrams* (MDDs) (Sharon et al., 2013).

- **(2) Makespan.** Many optimal solvers that optimize this cost function are *compilation-based solvers*, which means they solve MAPF by reducing it to known problems such as Constraint Satisfaction (CSP) (Ryan, 2010), Boolean Satisfiability (SAT) (Surynek, 2012c), Integer Linear Programming (ILP) (Yu & LaValle, 2013), or Answer Set Programming (ASP) (Erdem, Kisa, Oztok, & Schueller, 2013). These works mostly prove a polynomial-time reduction

from MAPF to these problems. Notably, these reductions are usually not directly applicable for the sum-of-costs variant.

1.2 Current Shortcomings and Contribution

A major weakness across all these works is that each of these algorithms was introduced and applied for one of these objective functions only. Furthermore, the connection/comparison between different algorithms was usually done only within a given class of algorithms but not across these two classes. Finally, experiments were always performed on one objective-function and very little is known on the performance and relevance of any given algorithm (developed for one cost function) on the other objective function.

This paper aims to start to close the gap. First, we discuss how to migrate algorithms across the different objective functions. Most of the search-based algorithms developed for the sum-of-cost objective function can be modified to the makespan variant with some technical adaptations such as modifying the cost function and the way the state-space is represented. Some initial directions are given by Sharon et al. (2015) and we give a complete picture here. By contrast, the compilation-based algorithms that were developed for the makespan objective function are not trivially modified to the sum-of-costs variant and sometimes a completely new encoding is needed.

A major algorithmic contribution of this paper is that we develop the first compilation-based solver for the sum-of-costs variant to SAT. Our SAT-based solver is based on establishing relations between the maximum makespan under the given sum-of-costs which enables to build SAT encodings that represent all feasible solutions for the given sum-of-costs. Bounds on the sum-of-costs in the SAT encoding are established by *cardinality constraints* (Bailleux & Boufkhad, 2003; Silva & Lynce, 2007). We show how to use known lower bounds on the sum-of-costs to reduce the number of Boolean variables that encode these cardinality constraints so as to be practical for current SAT solvers.

We then present how to migrate various techniques used in search-based approaches to our new SAT-based solver. First, we adapt ideas from the ICTS algorithm (Sharon et al., 2013) that uses *multi-value decision diagrams* (MDDs) (Srinivasan, Ham, Malik, & Brayton, 1990) to further reduce the size of SAT encodings. Next, we show how to integrate a modification of the *independence detection* technique (Standley, 2010b) into the SAT-based solver. Finally, we demonstrate the flexibility of our SAT-based solver by modifying it into a bounded-suboptimal sum-of-costs solver - a modification applicable in search-based approaches to trade-off the quality of solutions for shorter runtime (Barer, Sharon, Stern, & Felner, 2014).

Successful migration of techniques demonstrates the potential of combining ideas from both classes of approaches - search-based and compilation-based. Experimental results show that our SAT solver with various enhancements outperforms the best existing search-based solvers for the sum-of-costs variant on many scenarios. Hence, as a result of our unification provided in the beginning of this paper, we have a variety of algorithms which can be applied for both objective functions. We conclude this paper by providing experimental results comparing the hardness of solving MAPF with SAT-based and search-based solvers under the makespan and the sum-of-costs objectives in a number of domains.

Results presented in this work partly appeared in several conference papers (Surynek, Felner, Stern, & Boyarski, 2016a, 2016b, 2017a; Surynek, Svancara, Felner, & Boyarski, 2017c, 2017b). Here we provide a significantly more comprehensive summarization and extension of these results.

All SAT-based algorithms and encodings are described here in greater details, including detailed pseudo-codes. Detailed proofs of important properties such as completeness and time/space complexity are given. A broader context of how this work fits with existing works in classical planning, multi-agent planning, and planning using Markov Decision Processes is given. The experimental evaluation has been extended significantly, including experiments not reported in the conference papers that explore:

(i) The impact of using a newer SAT solver on the ability of our algorithms to solve more MAPF problems;

(ii) The relation between the computed makespan and the number of time expansions needed to prove its optimality;

(iii) The distribution of runtime spent in each time expansion.

1.2.1 ORGANIZATION

This paper is organized as follows. We first summarize related work on MAPF and introduce the context in which new methods are developed (section 2). Then a SAT-based solver for MAPF is introduced in several steps - starting with a basic solver (sections 3 and 4), which is further improved to obtain the final solver called MDD-SAT (section 5). A thorough experimental evaluation of MDD-SAT with respect to comparable search-based solvers is presented next (section 6). To further demonstrate the versatility of SAT-based approach, we show how to integrate independence detection into MDD-SAT (section 7) and how to build a bounded suboptimal solver using MDD-SAT (section 8) both with relevant experimental evaluation.

2. Related Work

We briefly summarize existing algorithmic approaches to MAPF in this section. We categorize algorithms into two streams according to the objective function they use. For optimization of *sum-of-costs* a great variety of algorithms has been proposed. On the other hand, previous *makespan*-optimal algorithms are limited to the compilation-based approach where the target formalism is represented by Boolean satisfiability. Many sum-of-costs optimal algorithms can be directly modified for the makespan variant. The opposite migration from the makespan-optimal case to the sum-of-costs optimality in compilation-based algorithms is however not straightforward.

2.1 Previous Sum-of-Costs Optimal Algorithms and Techniques

A*-based Algorithms. A* is a general-purpose algorithm that is well suited to solve MAPF. A common straight-forward state-space where the states are the different ways to place k agents into $n = |V|$ vertices, one agent per vertex is used. In the *start* and *goal* states agent a_i is located at vertices $\alpha_0(a_i)$ and $\alpha_+(a_i)$, respectively. Operators between states are all non-conflicting combinations of actions (including wait) that can be taken by the agents.

The branching factor in A*-based algorithms is an important measure. Let $b(a_i)$ be the branching factor of single agent a_i. Then the *effective branching factor* for k agents, denoted by b, is $b = \prod_{i=1}^{k} b(a_i)$. For example, in a 4-connected grid $b(a_i) = 5$ for most of agents; an agent can either move in four cardinal directions or wait at its current position. Then b is roughly 5^k; though

usually a bit smaller because many possible combinations of moves result in immediate conflicts, especially when the environment is dense with agents.

A simple admissible heuristic that is used within A* for MAPF is to sum the individual heuristics of single agents such as the Manhattan distance for 4-connected grids or the Euclidean distance for Euclidean graphs (Ryan, 2008). A more-informed heuristic is called the *sum of individual costs* heuristic . For each agent a_i we calculate its optimal path cost from its current state to $\alpha_+(a_i)$ assuming that other agents do not exist. Then, we sum these costs over all agents. More-informed heuristics use forms of pattern-databases (Goldenberg, Felner, Sturtevant, Holte, & Schaeffer, 2013; Goldenberg, Felner, Stern, Sharon, Sturtevant, Holte, & Schaeffer, 2014).

The most important drawback of A*-based algorithms is they need to tackle that the branching factor b of a given state may be exponential in k. We briefly summarize attempts to overcome the high branching factor.

Operator Decomposition (OD). Instead of moving all the agents to their next positions at once, agents advance to the next position one by one in a fixed order within the OD concept. The original operator for obtaining the next state is thus decomposed into a sequence of operators for individual agents each of branching factor $b(a_i)$. OD together with a *reservation table* enabled computations of next states where agents do not collide with each other in CA*, HCA*, and WHCA* (Silver, 2005).

Pruning of states by OD with respect to a given admissible heuristic was suggested by Standley (2010b). Two conceptually different states are distinguished - *standard* and *intermediate*. The intermediate states correspond to the situation when not all the agents performed their move while the standard states correspond to states in the original representation with no OD. The major strength of OD lies in the fact that the top-level A* algorithm does not need to distinguish between standard and intermediate states. The next node for expansion is selected among both standard and intermediate states while the cost function applies to both types of states. It may thus happen that a certain intermediate state is not expanded towards a standard state because other states turned out to be better according to the cost function. Such search space pruning cannot be done without operator decomposition as there would be standard states only.

Independence Detection (ID). Closely related to OD is the concept of *independence detection* (Standley, 2010b), which can be regarded as another branching factor reduction technique. The main idea behind this technique is that the difficulty of solving MAPF optimally grows exponentially with the number of agents. It would be ideal, if we could divide the problem into a series of smaller sub problems, solve them independently, and then combine them. The simple approach, called simple independence detection (SID), works as follows. First, it finds the shortest path for each agent to its goal, ignoring all other agents. Every pair of these single-agent paths is checked for conflicts (that is, when a collision between the agents will occur if they follow these paths concurrently). If a conflict is found, these agents are grouped together and a new optimal solution is found for the resulting group. This process continues, merging conflicting agents or groups of agents into larger groups until a set of solutions are found that do not conflict. This approach can be further improved by deliberately attempting to avoid the merging of groups.

More A*-based Algorithms. Enhanced Partial Expansion (EPEA*) (Goldenberg et al., 2014) avoids the generation of surplus nodes by using *a priori* domain knowledge. When expanding a node N EPEA* generates only the children N_c with $f(N_c) = f(N)$. The other children of N (with $f(N_c) \neq f(N)$) are discarded. This is done with the help of a domain-dependent *operator selection function* (OSF) (Goldenberg et al., 2014).

M* (Wagner & Choset, 2015) and its enhanced recursive variant (RM*) are important A*-based algorithms related to ID. M* dynamically changes the *dimensionality* and branching factor based on conflicts. The dimensionality is the number of agents that are not allowed to conflict. When a node is expanded, M* initially generates only one child in which each agent takes (one of) its individual optimal moves towards the goal (dimensionality 1). This continues until a conflict occurs between $q \geq 2$ agents at node N. At this point, the dimensionality of all the nodes on the branch leading from the root to N is increased to q and all these nodes are placed back in the OPEN list. When one of these nodes is re-expanded, it generates b^q children where the q conflicting agents make all possible moves and the $k - q$ non-conflicting agents make their individual optimal move.

An enhanced variant of M* called ODRM* (Ferner, Wagner, & Choset, 2013) builds RM* on top of Standley's OD rather than plain A*.

Increasing Cost Tree Search. The *increasing cost tree search* algorithm (ICTS) (Sharon et al., 2013) is a two-level MAPF solver that is conceptually different from A*. This algorithm is particularly important as its concepts will be be migrated into the SAT framework we are about to introduce. ICTS works as follows.

At its *high level*, ICTS searches the *increasing cost tree* (ICT). Every node in ICT consists of a k-ary vector $[C_1, \ldots C_k]$, which "represents *all* possible solutions in which the individual path cost of agent a_i is exactly C_i" (Sharon et al., 2013). The root of ICT is $[opt_1, ..., opt_k]$, where opt_i is the optimal individual path cost for agent a_i ignoring other agents, i.e., it is the length of the shortest path from $\alpha_0(a_i)$ to $\alpha_+(a_i)$ in G.

A child in ICT is generated by increasing the cost for one of the agents by 1. An ICT node $[C_1, .., C_k]$ is a *goal* if there is a complete non-conflicting solution such that the cost of the individual path for any agent a_i is exactly C_i. Figure 3 illustrates an ICT with 3 agents, all with optimal individual path costs of 10. Dashed lines mark duplicate children which can be pruned. The total cost of a node is $C_1 + \ldots + C_k$. For the root this is exactly $opt_1 + opt_2 + \ldots opt_k$. Since all nodes at the same height have the same total cost, a breadth-first search of the ICT will find the optimal solution.

The *low level* acts as a goal test for the *high level*. For each ICT node $[C_1, .., C_k]$ visited by the *high level*, the *low level* is invoked. Its task is to find a non-conflicting complete solution such that the cost of the individual path of agent a_i is exactly C_i. For each agent a_i, ICTS stores *all* single-agent paths of cost C_i in a special compact data-structure called a *multi-value decision diagram* (MDD) (Srinivasan et al., 1990).

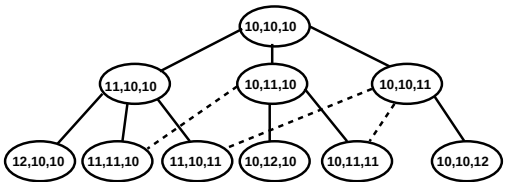


Figure 3: Increasing cost tree (ICT) for three agents.

The *low level* searches the cross product of the MDDs in order to find a set of k non-conflicting paths for the different agents. If such a non-conflicting set of paths exists, the *low level* returns *True* and the search halts. Otherwise, *False* is returned and the *high level* continues to the next high-level node (of a different cost combination).

ICTS also implements various pruning rules to enhance the search. These pruning rules and their connection to CSP was also studied (Sharon et al., 2013).

Conflict-based Search (CBS). Another optimal MAPF solver not based on A* is *conflict-based search* (CBS) (Sharon et al., 2015). In CBS, agents are associated with constraints. A *constraint* for agent a_i is a tuple $\langle a_i, v, t \rangle$ where agent a_i is prohibited from occupying vertex v at time step t. A *consistent path* for agent a_i is a path that satisfies *all* of a_i's constraints, and a *consistent solution* is a solution composed of only consistent paths.

Once a consistent path has been found for each agent, these paths are *validated* with respect to the other agents by simulating the movement of the agents along their planned paths.

If all agents reach their goal without any conflict the solution is returned. If, however, while performing the validation, a conflict is found for two (or more) agents, the validation halts and a conflict is resolved by adding constraints. If a *conflict*, $\langle a_i, a_j, v, t \rangle$ is encountered, that is agents a_i and a_j collide in vertex v at time t, we know that in any valid solution at most one of the conflicting agents, a_i or a_j, may occupy vertex v at time t. Therefore, at least one of the constraints, $\langle a_i, v, t \rangle$ or $\langle a_j, v, t \rangle$, must be satisfied. Consequently, CBS *splits* the search into two branches where one of these constraints is imposed in each branch.

2.2 Previous Makespan-Optimal Algorithms for MAPF

A major development in makespan-optimal MAPF solving has been done under the Boolean satisfiability (SAT) (Biere, Biere, Heule, van Maaren, & Walsh, 2009) compilation paradigm. Early works that compile MAPF to SAT focused on solution improvements in terms of shortening the makespan towards the optimum in an *anytime* manner (Surynek, 2012d). First, a makespan-suboptimal solution of the input MAPF is generated by a fast polynomial rule-based algorithm like BIBOX (Surynek, 2014c) or PUSH-AND-SWAP (Luna & Bekris, 2011; de Wilde, ter Mors, & Witteveen, 2013). Then continuous sub-sequences of time steps in the current solution are replaced by makespan-optimal ones. The length of replaced sub-sequences is increased in each iteration of the algorithm until it eventually covers the entire makespan. This ensures that given enough time the algorithm returns a makespan-optimal solution. A suboptimal solution is available at any stage of the algorithm.

INVERSE SAT encoding. Historically the first encoding of MAPF to SAT INVERSE relies on *log-space* encoded finite domain variables (Petke, 2015) that represent what agent is located at vertex v at each time step t - that is, the inverse $\alpha_t^{-1} : V \rightarrow A \cup \{\perp\}$ of $\alpha : A \rightarrow V$ (where assigning $\perp$ to a vertex means no agent is located at that vertex) is represented using log-space encoded bit-vectors $\mathcal{A}_t^v \in \{0, 1, 2, ..., k\}$. MAPF movement rules and state transitions are encoded by a number of constraints over $\mathcal{A}_t^v$ (Surynek, 2012d). Altogether Boolean formula $\mathcal{F}_\mu$ is constructed on top $\mathcal{A}_t^v$ Boolean variables such that it is satisfiable if and only if a solution to the input MAPF of makespan μ exists. The advantage of this encoding is that the *frame problem* (McCarthy & Hayes, 1969) of propagation of agents' positions to the next time step can be easily done by enforcing equalities between $\mathcal{A}_t^v$ and $\mathcal{A}_{t+1}^v$ (bit-wise equality for all bits of a pair of log-space encoded variables).

Further works in SAT-based approach to MAPF (Surynek, 2013a, 2013b) omitted the phase in which suboptimal solution was improved and a makespan-optimal solution was generated directly instead. The process of finding a makespan-optimal solution follows the scheme described in Algorithm 1. Assuming a solvable MAPF instance, a makespan-optimal solution is obtained by answering satisfiability of $\mathcal{F}_{\mu_0}, \mathcal{F}_{\mu_0+1}, ...$ until a satisfiable formula is found. The search starts with μ_0, the lower bound on makespan obtained as the length of longest path over all shortest paths connecting starting position $\alpha_0(a_i)$ and goal $\alpha_+(a_i)$ of each agent a_i. The first satisfiable $\mathcal{F}_\mu$ represents the optimal makespan and an optimal solution can be extracted from its satisfying assignment.

Algorithm 1: Framework of makespan-optimal SAT-based MAPF solving

```
1  Solve-MAPF-SAT_MAKESPAN(G = (V, E), A, α_0, α_+)
2      π ← {shortest path from α_0(a_i) to α_+(a_i) | i = 1, 2, ..., k}
3      μ ← max_{i=1}^{k} (length(π(a_i)))
4      while TRUE do
5          F(μ) ← encode(μ, G, A, α_0, α_+)
6          assignment ← consult-SAT-Solver(F(μ))
7          if assignment ≠ UNSAT then
8              π ← extract-Solution(assignment)
9              return π
10         μ ← μ + 1
```

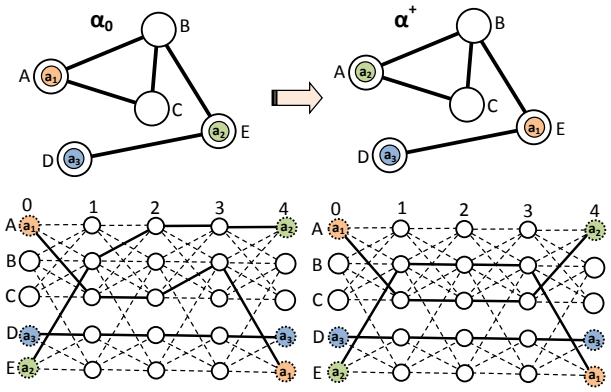


Figure 4: Searching of non-conflicting paths over anonymized agents - conflicts are reflected but an agent may end up in the wrong goal (lower right part).

ALL-DIFFERENT SAT encoding. The ALL-DIFFERENT encoding again employs the log-space representation of finite domain variables (Surynek, 2012b). The position of agent a_i at time step t, that is, α_t is represented instead of representing vertex occupancy - that is, finite domain variables $\mathcal{D}_t^{a_i} \in V$ are represented using log-space encoding. To ensure that conflicts among agents at vertices do not occur, the ALL-DIFFERENT constraint (Régin, 1994) is introduced for $\mathcal{D}_t^{a_i}$ variables over all agents for each time step t. The advantage of the ALL-DIFFERENT encoding is that various efficient encodings of the ALL-DIFFERENT (Biere & Brummayer, 2008; Surynek, 2012a) constraint over bit vectors can be integrated.

MATCHING SAT encoding. The next development has been done in SAT encoding called MATCHING that separates conflict rules in MAPF and agents transitions between time steps (Surynek, 2014a). Conflict rules are expressed over anonymized agents that are encoded by *direct* variables $\mathcal{M}_t^v \in \{TRUE, FALSE\}$ (Tamura, Taga, Kitagawa, & Banbara, 2009).

The presence of some agent in vertex v at time step t is indicated by a single propositional variable ($\mathcal{M}_t^v = TRUE$ if and only if $\exists a_i \in A$ such that $\alpha_t(a_i) = v$). Using anonymized agents is however not enough as agents may end up in other agent's goal - see Figure 4. For transitions where individual agents need to be distinguished, log-space encoded variables $\mathcal{A}_t^v \in \{0, 1, 2, ..., k\}$ represent what agent occupies a given vertex ($\mathcal{A}_t^v$ if and only if $\alpha_t(a_i) = v$) (Surynek, 2014a). The advantage of MATCHING over previous encodings INVERSE and ALL-DIFFERENT is that movement conflict rules can be expressed in a simpler way over direct variables $\mathcal{M}_t^v$ for anonymized agents. Compared to doing so over log-space encoded variables $\mathcal{A}_t^v$ or $\mathcal{D}_t^{a_i}$ that distinguish individual agents, smaller formula can be obtained with conflict reasoning over $\mathcal{M}_t^v$.

DIRECT SAT encoding. Lessons taken from the previous development was that introduction of directly encoded variables leads to significant performance improvements although encoding set of states by direct variable is not as space efficient as the in the log-space encoding case. The next encoding purely based on direct variables - called DIRECT MAPF encoding (Surynek, 2014b) - introduces a single Boolean variable for every triple of agent, vertex, and time step; formally there is a Boolean variable $\mathcal{X}(a_i)_t^v$ such that it is $TRUE$ if and only if agent a_i occupies v at time step t (some triples may be forbidden as unreachable). In this work we are partly inspired by the DIRECT encoding as for the idea of directly encoded variables.

ASP, CSP, and ILP approach. Although much work in makespan-optimal solving has been done for SAT, other compilation-based approaches to MAPF like ASP-based (Erdem et al., 2013) and CSP-based (Ryan, 2010) exist. Both ASP and CSP offer rich formalism to express various objective functions in MAPF. The ASP-based approach adopts a more specific definition of MAPF where bounds on lengths of paths for individual agents are specified as a part of the input. Except the bound on sum-of-costs the ASP formulation works with other constraints such as *no-cycle* (the agent shall not visit the same part of the environment multiple times), *no-intersection* (only one agent visits each part of the environment), or *no-waiting* (when minimization of idle time is desirable). The ASP program for a given variant of MAPF consisting of a combination of various constraints is solved by the CLASP SAT solver (Gebser, Kaufmann, Neumann, & Schaub, 2007).

Ryan (2010) proposed a CSP-based approach that focuses on the structure of the underlying graph G. The graph is partitioned into *halls* (singly-linked chain of vertices with any number of entrances and exits) and *cliques* (represents large open spaces with many entrances and exists) commonly referred to as sub-graphs. The plan is searched using CSP techniques over an abstract graph whose nodes are represented by the sub-graphs. Specific properties of different sub-graphs are reflected in the constraints - for example, agents in a clique sub-graph usually cannot exceed the capacity of the clique while in a hall sub-graph the agents must preserve their ordering. The resulting CSP is eventually solved using the GECODE solver (Tack, 2009).

The similarity of MAPF and multi-commodity flows has also been studied (Yu & LaValle, 2013b), where each agent is regarded as a different commodity of a multi-commodity flow. The individual depths of the multi-commodity flow are associated with individual time steps of MAPF solution. Finding optimal solutions of MAPF with respect to various objective functions can be then modeled as finding an optimal solution to a Integer Linear Programming (ILP) problem (Yu & LaValle, 2013a).

2.3 Multi-agent Planning and Multi-robot Planning

Multi-agent path finding can be understood as a specific problem in the wider paradigm of multi-agent planning (MAP) (de Weerdt & Clement, 2009; Torreño, Onaindia, Komenda, & Stolba, 2018). Agents in MAP operate in a shared environment and fulfil a given common goal. To achieve the common goal the agents need to be assigned their individual goals. The planning phase in MAP needs to coordinate actions of individual agents so that they benefit from positive interactions and avoid harmful interactions (Dimopoulos & Moraitis, 2006).

There exist many diverse techniques for MAP including heuristic search-based techniques with various improvements like divide-and-conquer (Ephrati & Rosenschein, 1994) as well as compilation-based techniques using SAT (Dimopoulos, Hashmi, & Moraitis, 2012) or classical planning (Dimopoulos & Moraitis, 2006) as a target formalism.

An important issue in MAP is also the negotiation phase in which agents decompose the common goal and assign individual tasks. Popular source methods for negotiation in MAP are game theoretical methods such as combinatorial auctions (Robu, Noot, Poutré, & van Schijndel, 2011), voting (Rosenschein, 1995), and argumentation (Sapena, Onaindia, & Torreño, 2010). Another important issue in MAP is that agents deliberately do not share their knowledge completely with other agents, referred to as privacy-preserving MAP (Tozicka, Jakubuv, Komenda, & Pechoucek, 2016).

MAPF as a special MAP task is greatly simplified as individual agents' goals are assigned as part of the input so the negotiation phase is missing in MAPF. On the other hand, there is a strong emphasis on avoiding the negative interactions during the planning phase in MAPF.

Natural motivation for multi-agent path finding on graphs rests in *multi-robot planning* and coordination (Chai & Su, 2013; Jones, Dias, & Stentz, 2011) (MRPC). The MRPC task consists in determining actions for a team of robots so that the team can fulfill a given task by executing the actions by individual robots. It is important that robots in the team collaborate with each other to fulfill the task, for example certain actions may require multiple robots to be executed.

One stream of research in MRPC focuses on heterogeneous teams of mobile robots (Parker, 1994), the team usually consists of a small number of robots. Typical applications are search and rescue operations (Beck, 2016) with multiple robots each with different abilities to perform relatively complex actions (Jorgensen, Chen, Milam, & Pavone, 2018). The other stream of works focuses on homogeneous teams of mobile robots (Oh, Park, & Ahn, 2015). The homogeneous teams typically consist of larger number of robots but individual robots can perform only simpler actions such as motion and actions like loading and unloading.

MAPF appears in MRPC as a sub-problem when robots need to plan their paths to reach destinations where actions are performed. Physical occupation of space by robots in MRPC can be modeled by MAPF rules so collision free planning can be carried out via MAPF. In the case of homogeneous teams, the MAPF problem can represent even the whole phases of the MRPC such as in the case of motion planning for swarms of UAVs (Kumar & Michael, 2012; Sørli, Graven, & Bjerknes, 2017).

3. SAT-Based Solvers for Makespan-Optimal MAPF

SAT-based MAPF solvers reduce the question of existence of a solution to given MAPF instance to the Boolean satisfiability problem (SAT) (Biere et al., 2009). The reduction encompasses Boolean variables to model the MAPF problem. As Boolean satisfiability answers binary questions the challenge is to apply SAT for MAPF where there is a cumulative cost function. A general scheme for using SAT solvers to solve an optimization problem is to reduce the optimization problem into a sequence of decision problems. The questions are *which decision problem to encode*, *how to encode it*, and *how to devise an appropriate sequence of these decision problems* that will guarantee a solution to the optimization problem at hand.

This challenge is stronger for the sum-of-costs variant where each agent has its own cost. We first recall concepts from SAT encodings for the makespan objective. Then, we present our SAT encoding for sum-of-costs.

A *time expansion graph* (denoted TEG) is a basic concept used by SAT-based MAPF solvers (Surynek, 2014a; Surynek et al., 2016a). 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 underlying graph G are duplicated for all time-steps from 0 up to the given makespan bound μ. Then, possible actions (move along edges

or wait) are represented as directed edges between successive time steps. Figure 5 shows graph $G = (V, E)$ and its TEG of depth 2 for time steps 0, 1 and 2 (vertical layouts) denoted $TEG(2)$. Copies of the underlying graph G within TEG will be called *layers*.

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 μ corresponding to undirected graph $G = (V, E)$ is a digraph $TEG(\mu) = (V', E')$ where $V' = \{u_j^t \mid t = 0, 1, ..., \mu \wedge u_j \in V\}$ and $E' = \{(u_j^t, u_l^{t+1}) \mid t = 0, 1, ..., \mu - 1 \wedge (\{u_j, u_l\} \in E \vee j = l)\}$.*

The encoding for MAPF introduces TEGs for individual agents. That is, we have $TEG_i(\mu) = (V_i, E_i)$ for each agent $a_i \in \{1, 2, ..., k\}$. Directed non-conflicting paths in TEGs correspond to valid non-conflicting movements of agents in the underlying graph G. The existence of non-conflicting paths in TEGs is encoded as satisfiability of a Boolean formula. Next, we describe in more details the encoding style used in the DIRECT encoding.

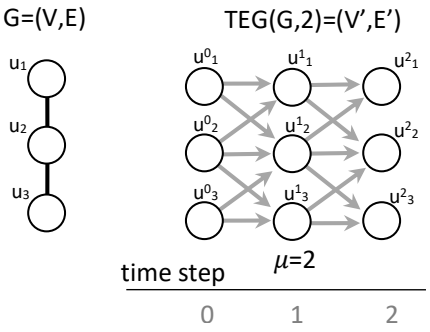


Figure 5: An example of time expansion graph: input graph (left) and its expansion for 3 steps (depth 2). For each vertex $u_j \in V$ its copy u_j^t indexed by time step t is introduced and corresponding directed edges are added between successive time steps to represent actions including wait actions.

Boolean variables and constraints (clauses) for a single time step $t \in \{0, 1, ..., \mu\}$ in $TEG_i(\mu)$ represent any possible position of agent a_i at time step t; that is, we have $\mathcal{X}(a_i)_v^t$ from the DIRECT encoding. Boolean variables for all TEGs together represent all possible configurations of agents from time step 0 up to time step μ. It is ensured by constraints that configurations of agents in consecutive time steps of TEGs correspond to valid actions: agent a_i can appear in vertex u_j^t if it can move there from the previous time step in TEG_i along a directed edge, that is, if a_i is in some u_l^{t-1} such that $(u_l^{t-1}, u_j^t) \in E_i$. We also have inter-TEG constrains ensuring that agents do not collide with each other (detailed list of constraints will be introduced lter for the sum-of-costs variant).

Given a desired makespan μ, formula $\mathcal{F}_\mu$ represents the question of whether there is a collection of non-conflicting directed paths in TEG_1,...,TEG_k of depth μ such that the first configuration equals to α_0 and the last one equals to α_+. The search for optimal makespan is done by iteratively incrementing $\mu = 0, 1, 2...$ until a satisfiable formula $\mathcal{F}_\mu$ is obtained as shown in Algorithm 1.

This process ensures finding a makespan-optimal solution in case of a solvable input MAPF instance, since the satisfiability of $\mathcal{F}_\mu$ is a non-decreasing function of μ. It is important to note that solvability of a given MAPF can be checked in advance by a fast polynomial algorithm like PUSH-AND-ROTATE (de Wilde et al., 2013). More information on SAT encoding for the makespan-optimal variant can be found in various studies (Surynek, 2014a, 2014, 2014b). The detailed transformation

of a question of whether there are non-conflicting paths in TEGs will be shown in the following sections.

4. SAT-Based Solvers for Sum-of-costs Optimal MAPF

The general scheme described above for finding the optimal makespan is to convert the optimization problem (finding the minimal makespan) to a sequence of decision problems. The decision problem was: is there a solution of makespan μ, and the sequence of decision problems was to increment μ until the minimal makespan is found. The *numeric objective function* to minimize, i.e., the makespan μ corresponds directly to the number of time expansions of the underlying graph G in TEG. Thus, the decision problem corresponded to the existence of directed paths in TEGs satisfying additional constraints encoding the MAPF rules.

We apply an analogous scheme for finding the optimal sum-of-costs, converting it to a sequence of decision problems – is there a solution of a given sum-of-costs ξ, and the sequence of decision problems is to increment ξ until finding the minimal sum-of-costs.

It is important to note that an incremental strategy to obtain the optimal value of the objective function is suitable only when the cost of a query is exponential in μ or ξ. In the case of uniform query costs different strategies, like binary search, would be more suitable. This roughly holds in MAPF, as increasing μ corresponds to adding a fresh time step in the TEGs, which is reflected in the encoded Boolean formula by adding a number of variables and constraints proportional to the size of G.

Since the runtime of a SAT solver is exponential in the size of the input formula in the worst case we have that the runtime for answering $\mathcal{F}_\mu$ is exponential in μ in the worst case. In such a setup with incremental strategy, the cost/runtime of the last query is roughly the same as the total cost/runtime of previous queries in the worst case. As we will see later, the same applies also for the sum-of-costs ξ.

However, encoding the decision problem for the sum-of-costs 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 ξ and **(2)** Bounding the makespan.

- **Cardinality constraints.** This is a technique from the SAT literature that enables counting and bounding a numeric cost in a Boolean formula (Bailleux & Boufkhad, 2003; Silva & Lynce, 2007; Sinz, 2005). Technically this is done by counting and bounding the number of Boolean variables from a given set that are assigned $TRUE$. This consequently enables encoding in a Boolean formulate a constraint that bounds the sum-of-costs (details in Section 4.3).

- **Upper bound on the required time expansions.** We show below how to compute for a given sum of cost value ξ a value μ such that all possible solutions with sum-of-costs ξ must be possible for a makespan of at most μ (details in Section 4.2). This enables encoding the decision problem of whether there is a solution of sum-of-costs ξ by using a SAT encoding similar to the makespan encoding with μ time expansions. In other words, it will be sufficient to use TEGs of depth μ in order to represent all solutions that fits under the given sum-of-costs ξ.

In the following sections, we explain each of this techniques in detail, along with additional implementation details.

4.1 Bounding ξ via Cardinality Constraint

The SAT literature offers a technique for encoding a *cardinality constraint* (Silva & Lynce, 2007; Sinz, 2005), which allows calculating and bounding a numeric cost within the Boolean formula. Formally, for a bound $\lambda \in \mathbb{N}$ and a set of Boolean variables $X = \{x_1, x_2, ..., x_m\}$ the *cardinality constraint* $\leq_\lambda \{x_1, x_2, ..., x_m\}$ is satisfied if and only if the number of variables from the set X that are set to $TRUE$ is $\leq \lambda$. There are various ways how to encode cardinality constraints in Boolean formulae. The standard approach is to simulate arithmetic circuits (Bailleux & Boufkhad, 2003) within the formula (either using binary or unary representation of numbers encoded by vectors of Boolean variables).

In our SAT encoding, we use such cardinality constraints to encode a constraint that upper bounds the sum-of-costs. Specifically, we map every agent's action to a Boolean variable, which is $TRUE$ if that action is performed in the plan. Then, we add cardinality constraint over these variables, thereby encoding a bound on the sum-of-cost of the plan. To find a plan with an optimal sum-of-cost, we use the general structure of the makespan SAT encoding, increasing this sum-of-cost bound by one in every iteration, until a solution is found.

A challenge with such a SAT encoding is that we must set both the number of time expansions as well as the sum of cost bound in every iteration. We address below the challenge of how to connect these two factors – the number of time expansions and the sum of cost bound. The explicit form of the cardinality constraint will be expressed later after we will define the Boolean variables that the cardinality constraint will bound.

4.2 Bounding the Makespan for the Sum of Costs

We compute how many time expansions μ are needed to guarantee that if a solution with sum-of-costs ξ exists then it will be found within at most μ time expansions. In other words, in our encoding, the values we give to ξ and μ must fulfill the following requirement:

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) for given ξ, we require the following definitions. Let $\xi_0(a_i)$ be the cost of the shortest individual path for agent a_i, and let $\xi_0 = \sum_{a_i \in A} \xi_0(a_i)$. ξ_0 is called the *sum of individual costs* (SIC) (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. That is, let $\Delta = \xi - \xi_0$.

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

Algorithm 2: Basic SAT-based sum-of-costs optimal MAPF solving.

```
1  Solve-MAPF-SAT_SOC(Σ = (G = (V, E), A, α_0, α_+))
2      if Σ is unsolvable then
3          return UNSOLVABLE
4      μ_0 ← max_{a_i ∈ A} ξ_0(a_i)
5      ξ_0 ← Σ_{a_i ∈ A} ξ_0(a_i)
6      Δ ← 0
7      while TRUE do
8          μ ← μ_0 + Δ
9          for each agent a_i ∈ A do
10             TEG_i(μ, ξ_0^i) ← build-TEG(a_i, μ, ξ_0(a_i), Σ)
11         F(μ_0, ξ_0, Δ) ← encode-MAPF(Σ, Δ, TEG_1(μ, ξ_0^1), ..., TEG_k(μ, ξ_0^k))
12         π ← consult-SAT-SOLVER(F(μ_0, ξ_0, Δ))
13         if π ≠ UNSAT then
14             return π
15         Δ ← Δ + 1
```

Proof: 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, knowing that if a solution of cost ξ exists then it will be found within $\mu = \mu_0 + \Delta$ time expansions. In other words, Proposition 1 shows the relation between the parameters μ and ξ, which will both be changed by changing Δ.

Algorithm 2 summarizes our optimal sum-of-costs algorithm. In every iteration, μ is set to $\mu_0 + \Delta$ (line 7) and the relevant TEGs of depth μ for various agents are built (these TEGs also take into account the minimum individual cost $\xi_0^i = \xi_0(a_i)$ of individual agents a_i, described below). Using TEGs of individual agents a formula $\mathcal{F}(\mu_0, \xi_0, \Delta)$ is constructed that encodes a decision problem whether there is a solution with sum-of-costs $\xi = \xi_0 + \Delta$ and makespan $\mu = \mu_0 + \Delta$. Afterwards the formula is queried to the SAT solver (line 12).

The first iteration starts with $\Delta = 0$. If such a solution exists, it is returned. Otherwise Δ is incremented by one, μ and consequently ξ are modified accordingly and the next iteration of SAT consulting is activated.

Proposition 2 *The SAT-based algorithm MAPF-SAT (Algorithm 2) is sound and complete.*

Proof: This algorithm clearly terminates; for unsolvable instances after the initial solvability test; for solvable MAPF instances as we start seeking a solution of $\xi = \xi_0$ ($\Delta = 0$) and increment Δ (which increments ξ and μ as well) to all possible values. Hence using assumption that solvability of MAPF with respect to the sum-of-costs bound is non-decreasing function we eventually encounter Δ (ξ and μ) for which Σ is solvable and valid solution is calculated and returned. □

The initial unsolvability check of an MAPF instance can be done by any polynomial-time complete suboptimal algorithm such as PUSH-AND-ROTATE (de Wilde et al., 2014).

Algorithm 3: Construction of the time expansion graph.

```
1  Construct-TEG(μ, ξ_0^i, G = (V, E))
2      V_i ← ∅
3      E_i ← ∅
4      F_i ← ∅
5      for u_j ∈ V do
6          E ← E ∪ {u_j, u_j} /* adding loops to ensure the frame axiom */
7      for t ∈ {0, 1, ..., μ} do
8          V_i ← V_i ∪ {u_j^t | u_j ∈ V}
9      for t ∈ {0, 1, ..., μ − 1} do
10         for each {u_j, u_l} ∈ E do
11             if t ≤ ξ_0(a_i) then
12                 E_i ← E_i ∪ {u_j^t, u_l^{t+1}}
13             else
14                 if {u_j, u_l} ≠ {t} then
15                     F_i ← F_i ∪ {u_j^t, u_l^{t+1}}
16                 else
17                     E_i ← E_i ∪ {u_j^t, u_l^{t+1}}
18     return (TEG_i(μ, ξ_0^i) = (V_i, E_i, F_i))
```

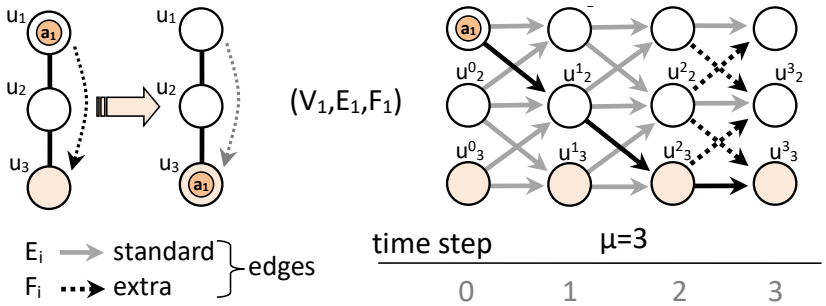


Figure 6: A $TEG_1(3, 2)$ for agent a_1 that needs to go from u_1 to u_3.

4.2.1 EFFICIENT USE OF THE CARDINALITY CONSTRAINT

The complexity of basic encoding of a cardinality constraint depends quadratically on the number of constrained variables (Silva & Lynce, 2007; Sinz, 2005); more precisely $\leq_\lambda \{x_1, x_2, ..., x_m\}$ requires $\mathcal{O}(m^2)$ Boolean variables and clauses. Hence bounding too many variables using the cardinality constraint may contribute significantly to the total size of the encoding.

Since each agent a_i must move at least $\xi_0^i = \xi_0(a_i)$ times, we can reduce the number of variables counted by the cardinality constraint by only counting the variables corresponding to extra movements over the first $\xi_0(a_i)$ movements a_i makes. We implement this by introducing a slightly modified TEG for a given agent a_i, labeled $TEG_i(\mu, \xi_0^i)$.

Algorithm 3 lists how to construct the TEG for a given makespan bound μ and individual cost bound ξ_0^i, agent a_i, and the underlying graph G. $TEG_i(\mu, \xi_0^i) = (V_i, E_i, F_i)$ 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 $\leq \xi_0^i = \xi_0(a_i)$ or represent a wait action in the goal vertex. These are called *standard edges*. F_i denoted as *extra edges* are directed edges whose destination is at time step $> \xi_0^i = \xi_0(a_i)$ and do not correspond to a wait action in the goal vertex.

Figure 6 shows an underlying graph for agent a_1 (left) and the corresponding $TEG_1(3, 2)$. Note that the optimal solution of cost 2 is denoted by the diagonal path in $TEG_1(3, 2)$. Edges that belong to F_i are those that their destination is time step 3 and are not corresponding to wait actions (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 the cardinality constraint more efficient (bounding the number of all edges by $\xi = \xi_0 + \Delta$ is equivalent to bounding the number of extra edges by Δ). The case when agent a_i leaves its goal vertex at time step $t > \xi_0^i$ will be treated separately. In such a case, we need to bound the number of wait actions in the goal vertex between time steps ξ_0^i and t, let us call these wait actions *non-terminal*.

4.3 Detailed Description of the SAT Encoding

Agent a_i must go along a directed path from its initial position to its goal within $TEG_i(\mu, \xi_0^i)$; that is from a vertex in the first layer corresponding to the start position $\alpha_0(a_i)$ to a vertex in the μ-th layer corresponding to the goal position $\alpha_+(a_i)$. The directed path in $TEG_i(\mu, \xi_0^i)$ simulates movement of agent in the underlying graph G (waiting and repeated visits to the same vertex can be modeled).

Hence we need to encode searching for a path from $\alpha_0^0(a_i)$ to $\alpha_+^\mu(a_i)$ in $TEG_i(\mu, \xi_0^i)$ within the Boolean formula. To solve the MAPF problem we need to search for directed paths for individual agents in parallel. Thus possible interactions between agents need to be taken into account to reflect the MAPF movement rules correctly. Additional constraints will be added to capture *collision avoidance* etc. Finally, we will encode the cardinality constraint saying that the number of extra edges and non-terminal wait actions across all TEGs is at most Δ.

We want to ask whether a sum-of-costs solution of ξ exist. For this we build $TEG_i(\mu_0 + \Delta, \xi_0^i) = (V_i, E_i, F_i)$ for each agent $a_i \in A$. Next we introduce the Boolean encoding (denoted `BASIC-SAT`), formula $\mathcal{F}_{BASIC}(\mu_0, \xi_0, \Delta)$, 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 u_j at time step t.

2:) $\mathcal{E}_{j,k}^t(a_i)$ for every $t \in \{0, 1, ..., \mu-1\}$ and $(u_j^t, u_l^{t+1}) \in (E_i \cup F_i)$ – Boolean variables that model transition of agent a_i from vertex u_j to vertex u_l 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 $u_j^t \in V_i$ and $u_l^{t+1} \in V_i$ with $(u_j^t, u_l^{t+1}) \in F_i$ — Boolean variables that model cost of movements along **extra edges** (from F_i) between time steps t and $t + 1$. We denote by $\mathcal{C}$ the set of all such variables.

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 a vertex in the next time step. This is encoded by the following two constraints, which are posted for every $t \in T_\mu$ and $u_j^t \in V_i$

$$\mathcal{X}_j^t(a_i) \Rightarrow \bigvee_{(u_j^t, u_l^{t+1}) \in E_i \cup F_i} \mathcal{E}_{j,l}^t(a_i), \tag{1}$$

$$\sum_{(u_j^t, u_l^{t+1}) \in E_i \cup F_i} \mathcal{E}_{j,l}^t(a_i) \leq 1 \tag{2}$$

The pseudo-Boolean constraint (2) (often called *at-most-one* constraint) can be translated to clauses in multiple different ways (Ansótegui, Bofill, Coll, Dang, Esteban, Miguel, Nightingale, Salamon, Suy, & Villaret, 2019). One simple and efficient translation at the same time is to forbid all possible pairs variables to be simultaneously $TRUE$ as follows. As the translation consists of binary clauses only it supports *unit propagation* (Dowling & Gallier, 1984) inside the SAT solver.

$$\bigwedge_{(u_j^t, u_l^{t+1}), (u_j^t, u_{l'}^{t+1}) \in E_i \cup F_i \wedge l < l'} \neg \mathcal{E}_{j,l}^t(a_i) \vee \neg \mathcal{E}_{j,l'}^t(a_i) \tag{3}$$

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 $(u_j^t, u_l^{t+1}) \in E_i \cup F_i$:

$$\mathcal{E}_{j,l}^t(a_i) \Rightarrow \mathcal{X}_j^t(a_i) \wedge \mathcal{X}_l^{t+1}(a_i) \tag{4}$$

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

$$\mathcal{E}_{j,l}^t(a_i) \Rightarrow \bigwedge_{a_{i'} \in A \wedge i' \neq i \wedge u_j^t \in V_{i'}} \neg \mathcal{X}_j^t(a_{i'}) \tag{5}$$

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 vertex $u_j^t \in V^t$:

$$\sum_{u_j^t \in V_i} \mathcal{X}_j^t(a_i) \leq 1 \tag{6}$$

Again this pseudo-Boolean at-most-one constraint can be translated to clauses by forbidding any pair of agents $a_i, a_{i'} \in A$ such that $i < i'$ to appear simultaneously in a vertex:

$$\bigwedge_{u_j^t \in V_i \cap V_{i'}} \neg \mathcal{X}_j^t(a_i) \vee \neg \mathcal{X}_j^t(a_{i'}) \tag{7}$$

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

$$\mathcal{E}_{j,l}^t(a_i) \Rightarrow \mathcal{C}^t(a_i) \tag{8}$$

C6: If agent a_i leaves the goal all wait actions before leaving the goal must be accumulated to the cost. These actions correspond to non-terminal wait actions. Hence the following constraint is added for each agent a_i and $t \in T_\mu$:

$$\mathcal{C}^t(a_i) \Rightarrow \bigwedge_{t' \in \{\xi_i^0, \xi_i^0+1, \ldots, t-1\}} \mathcal{C}^{t'}(a_i) \tag{9}$$

C7: 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 or non-terminal wait actions. This corresponds to the following simple cardinality constraint over the variables in $\mathcal{C}$:

$$\leq_\Delta \{C \in \mathcal{C}\} \tag{10}$$

Final formula. The resulting Boolean formula $\mathcal{F}_{BASIC}(\mu_0, \xi_0, \Delta)$ is a conjunction of $C1, C2$, $\ldots C7$ and is the one that is constructed and consulted by Algorithm 2 (lines 11 and 12).

The constraints also imply that agent cannot swap their positions, that is for example two agents moving in opposite directions in a corridor to cross each other and move on is forbidden. The following propositions summarize the correctness and the space complexity of our `BASIC-SAT` encoding.

Proposition 3 *MAPF $\Sigma = (G = (V, E), A, \alpha_0, \alpha_+)$ has a sum-of-costs solution of ξ if and only if $\mathcal{F}_{BASIC}(\mu, \Delta)$ is satisfiable. Moreover, a solution of MAPF Σ with the sum-of-costs of ξ can be extracted from the satisfying assignment of $\mathcal{F}_{BASIC}(\mu_0, \xi_0, \Delta)$ by interpreting its $\mathcal{X}_j^t(a_i)$ variables.*

Proof: The direct consequence of the above definitions is that a valid solution of a given MAPF Σ of sum-of-costs ξ corresponds to non-conflicting directed paths in the TEGs of the individual agents that in total use at most Δ extra edges and non-terminal wait actions. These non-conflicting paths further correspond to variable assignment satisfying $\mathcal{F}_{BASIC}(\mu_0, \xi_0, \Delta)$, i.e., variable assignments represent directed paths in TEGs plus they satisfy bounds imposed by the cardinality constraint. □

Proposition 4 *Let D be the maximal degree of any vertex in G and let k be the number of agents. If $k \cdot |E| \geq \Delta = \xi - \xi_0$ and $k \geq D$ then the number of clauses in $\mathcal{F}_{BASIC}(\mu_0, \xi_0, \Delta)$ is $\mathcal{O}(\mu \cdot k^2 \cdot |E|)$, and the number of variables is $\mathcal{O}(\mu \cdot k \cdot |E|)$*

Proof: The components of $\mathcal{F}_{BASIC}(\mu_0, \xi_0, \Delta)$ are described in equations (1) – (10). Equation (1) introduces at most $\mathcal{O}(k \cdot \mu \cdot |E|)$ clauses. Equation (3) introduces at most $\mathcal{O}(k \cdot \mu |E| \cdot D)$ clauses. Equation (4) introduces at most $\mathcal{O}(k \cdot \mu \cdot |E|)$ clauses. Equation (5) introduces at most $\mathcal{O}(k^2 \cdot$

$\mu \cdot |E|)$. Equation (7) introduces at most $\mathcal{O}(k^2 \cdot \mu \cdot |V|)$ clauses. Equation (8) introduces at most $\mathcal{O}(k \cdot \mu \cdot |E|)$ clauses. Equation (9) introduces at most $\mathcal{O}(k \cdot \mu \cdot (\xi - \xi_0))$ binary clauses since each implication from (9) develops into at most $\xi - \xi_0$ binary clauses. Equation (10) introduces at most $\mathcal{O}(k \cdot \mu \cdot (\xi - \xi_0))$ clauses, since the constraint checking that m variables has a cardinality constraint of λ requires $\mathcal{O}(m \cdot \lambda)$ clauses (Sinz, 2005). Summing all the above results in a total of $\mathcal{O}(\mu \cdot k \cdot (|E| \cdot (D + k) + (\xi - \xi_0)))$. If we assume that $k > D$ and that $k \cdot |E| > (\xi - \xi_0)$ then the number of clauses is $\mathcal{O}(\mu \cdot k^2 \cdot |E|)$. The number of variables is computed in a similar way. □

4.4 Improving Basic SAT by Adding MDDs

The 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 $|V|$ and $|E|$ factors in the encoding size). To obtain a significant speedup we reduce the size of TEG_i for agent a_i in terms of the number of vertices while the soundness of encoding is preserved. To do this we borrow the ideas of *Multi-Value Decision Diagram* (MDD) (Andersen, Hadzic, Hooker, & Tiedemann, 2007) from the search-based MAPF algorithm ICTS (Sharon et al., 2013). This shows the advantage of combining techniques from both classes of approaches (search-based and SAT).

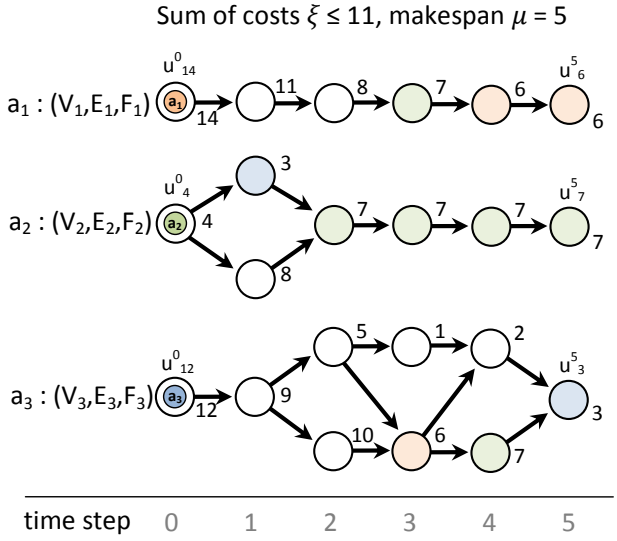


Figure 7: MDDs for agents a_1, a_2, and a_3 for the MAPF from Figure 1 for *sum of individual cost* $\xi \leq 11$. Specifically, $MDD_1(5, 4)$, $MDD_2(5, 2)$, and $MDD_3(5, 5)$ are shown (no extra edges are present).

Let $TEG_i(\mu, \xi_0^i)$ denote TEG_i for μ time expansions where $\mu = \mu_0 + \Delta$. $MDD_i(\mu, \xi_0^i)$ is a digraph that represents all possible valid paths from $\alpha_0(a_i)$ to $\alpha_+(a_i)$ of makespan/cost μ for agent a_i (in the case of single agent the makespan and the sum-of-costs are the same). $MDD_i(\mu, \xi_0^i)$ has

a single *source node* at level 0 and a single *sink node* at level μ. Every node at depth t of MDD_i^μ corresponds to a possible position of a_i at time t, that is a position on a path of cost μ from $\alpha_0(a_i)$ to $\alpha_+(a_i)$. In other words, vertex $u_j \in V$ that is too far from either $\alpha_0(a_i)$ or $\alpha_+(a_i)$ is not represented in $MDD_i(\mu, \xi_0^i)$. If there is no path of makespan/cost μ from $\alpha_0(a_i)$ to $\alpha_+(a_i)$ containing u_j, then u_j is not included in $MDD_i(\mu, \xi_0^i)$. Similarly as in $TEG_i(\mu, \xi_0^i)$, the parameter ξ_0^i is used to distinguish between the standard edges (those that terminate in time step $\leq \xi_0^i$ or represent the wait action in the goal vertex) in MDD and extra edges (those that terminate at time step $> \xi_0^i$ and do not correspond to the wait action in the goal vertex).

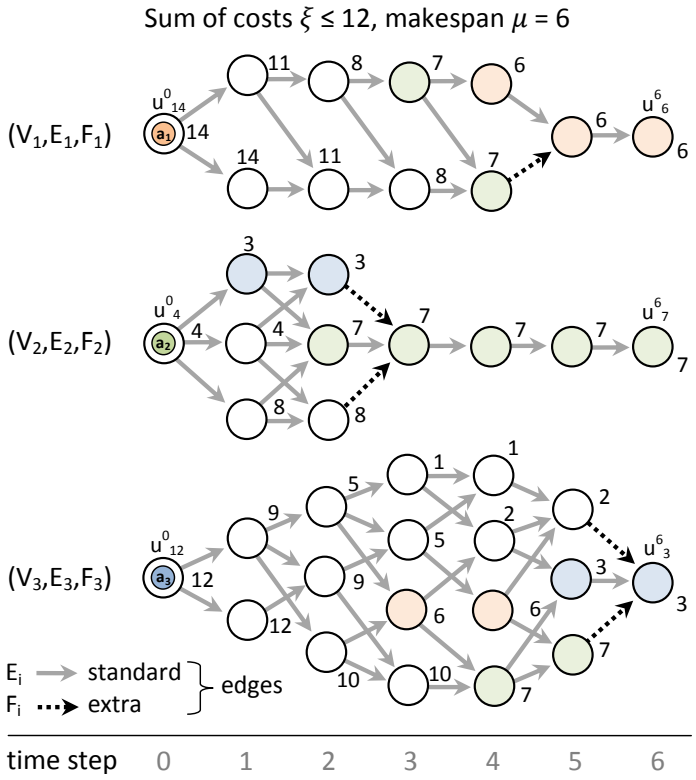


Figure 8: MDDs from Figure 7 for the incremented *individual cost* from 11 to 12 ($\xi \leq 12$).

It is easy to see that $MDD_i(\mu, \xi_0^i)$ is a subgraph of $TEG_i(\mu, \xi_0^i)$. While $TEG_i(\mu, \xi_0^i)$ includes all vertices of G at each time step, $MDD_i(\mu, \xi_0^i)$ includes only those vertices and edges that are reachable within makespan/cost μ, and thus vertices not in $MDD_i(\mu, \xi_0^i)$ can be ignored.

Moreover, the maximum cost that can be consumed by single agent a_i under given sum-of-costs bound $\xi = \xi_0 + \Delta$ is $\xi_0(a_i) + \Delta$ where, as defined above, $\xi_0(a_i)$ is the length of the shortest path connecting $\alpha_0(a_i)$ with $\alpha_+(a_i)$ in G (assuming no other agent exists). Thus, it is sufficient to replace $TEG_i(\mu_0 + \Delta, \xi_0^i)$ with $MDD_i(\mu_0 + \Delta, \xi_0^i)$ in the algorithm.

MDDs for the agents of Figure 1 are shown in Figures 7 and 8. Indeed, the size of the MDDs is much smaller than the corresponding TEGs, which include all states for all time steps. Though the increase in size caused by ability to reach more vertices under given the next sum-of-costs bounds is observable between Figures 7 and 8.

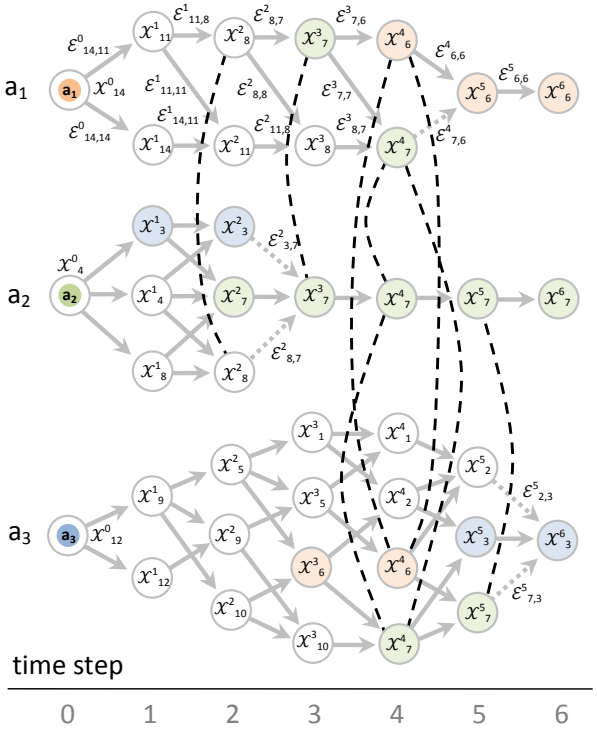


Figure 9: An illustration of MDD-SAT encoding using MDDs from Figure 8. Mutual exclusion constraints (C4) that prevent multiple agent occurrence in the same vertex are shown using dashed edges.

The encoding that uses MDD-based time expansion will be called MDD-SAT and the corresponding formulae will be denoted as $\mathcal{F}_{MDD}(\mu_0, \xi_0, \Delta)$. The formula is similar to the BASIC-SAT encoding. The only difference 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 Boolean 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 are shown. Up to two orders of magnitude reduction can be observed when MDDs are used.

An illustration of the $\mathcal{F}_{MDD}(\mu_0, \xi_0, \Delta)$ formula is shown in Figure 9. It is particularly observable that MDDs reduce the number of mutual exclusion (*mutex*) constraints (dashed edges) by omitting unreachable vertices (and all the constraints incident with them).

5. Experimental Results for MDD-SAT

We experimented on 4-connected grids with randomly placed obstacles (Silver, 2005) and on *Dragon Age* maps (Sharon et al., 2015; 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 configurations.

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 effect of using MDDs in the encoding in terms of the number of variables and clauses.

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 over 4-connected grids.

The SAT approaches were implemented in C++ [2]. The implementation consists of a top level algorithm for finding the optimal sum-of-costs ξ and *CNF* formula generator (Biere et al., 2009) that prepares input formula for a SAT solver into a file. The SAT solver is an external module our this architecture. We used `Glucose 3.0` (Audemard & Simon, 2009; Audemard, Lagniez, & Simon, 2013), which is a top-performing SAT solver in the recent editions of the *SAT Competition* (Järvisalo, Berre, Roussel, & Simon, 2012; Surynek, 2014a; Balyo, Heule, & Järvisalo, 2017). Since the SAT solver is called multiple times when solving a single MAPF problem, we call it directly through its API. This SAT solver also supports incremental SAT solving (Fazekas, Biere, & Scholl, 2019; Audemard et al., 2013), that is, it is possible to add variables and clauses incrementally between calls, and the solver is able utilize this to learn clauses and speedup its search.

The cardinality constraint was encoded using the standard circuit based encoding called *sequential counter* (Sinz, 2005). In our initial testing we considered various encodings of the cardinality constraint (Bailleux & Boufkhad, 2003; Silva & Lynce, 2007). Our finding is that changing the encoding has a minor effect [3]

ICTS and ICBS were implemented in C#, based on their original implementation (here we used a slight modification in which the target vertex of a move must be empty). All experiments were performed on a system with Xeon 2.8Ghz core with 32 Gb of memory.

5.1 Square Grid Experiments

We first experimented on 8×8, 16×16, and 32×32 grids with 10% obstacles while increasing the number of agents from 1 up to the last number where at least one solver was able to solve an instance with that number of agents (in case of the 8×8 grid this is 17 agents; and 32 and 60 in case of 16×16 and 32×32 grids respectively). For each number of agents 10 random instances were generated.

Figure 10 presents *success rate* results where each algorithm was given a time limit of 300 seconds (as was done by Sharon et al., 2013, Boyarski et al., 2015; Sharon, Stern, Felner, & Sturtevant,

2. The implementation is available as part of a MAPF experimental project on: https://github.com/surynek/reLOC.
3. Due to the knowledge of lower bounds on the sum-of-costs, the number of variables involved in the cardinality constraint is relatively small and hence the different encoding style has not enough room to show its benefit.

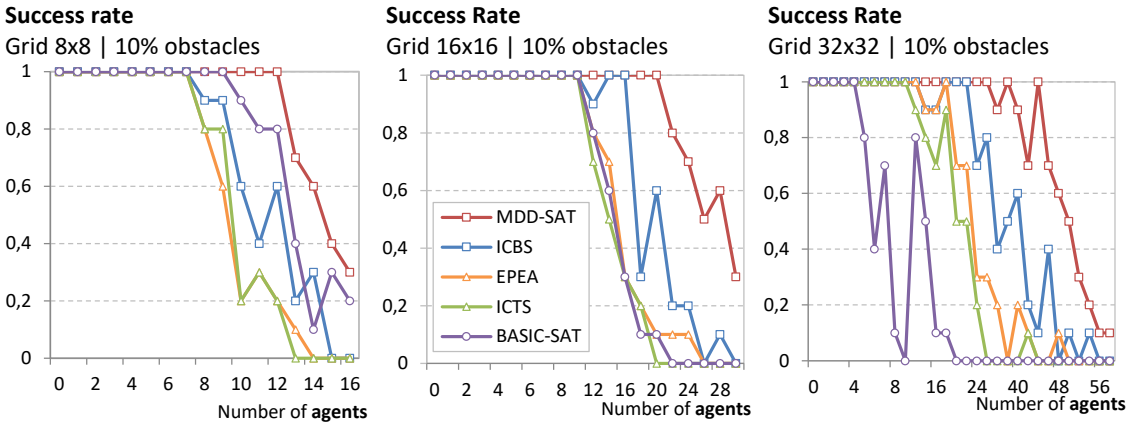


Figure 10: Success rate results for 8×8, 16×16, and 32×32 grids (10 instances per number of agents).

2015a). Success rate corresponds to percentage out of given 10 random instances solved within the time limit as a function of the number of agents (higher curves are better).

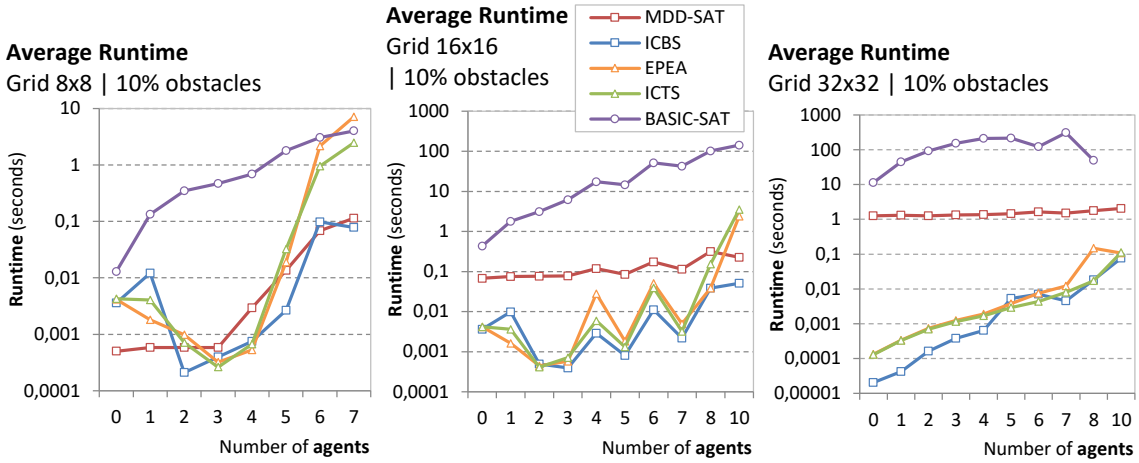


Figure 11: Average runtime for 8×8, 16×16, and 32×32 grids measured out of 10 instances per number of agents (vertical axis uses a logarithmic scale).

Figure 11 reports the average runtime for instances that were solved by all algorithms (lower curves are better). Here, we required 100% success rate for all the tested algorithms to be able to calculate the average runtime; this is also the reason why the number of agents is smaller.

Figure 12 visualizes the results on 8×8, 16×16, 32×32 grid in a different way. Here, we present the number of instances (out of all instances for all number of agents) that each method solved (y-

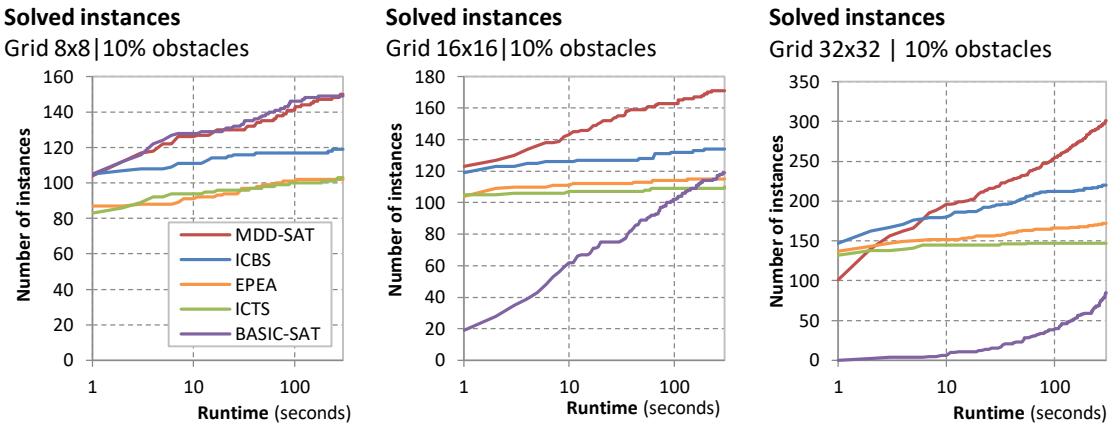


Figure 12: Number of solvable instances in 8×8, 16×16, and 32×32 grids as a function of time limit (the horizontal runtime axis uses logarithmic scale).

axis) as a function of the elapsed time (x-axis). Thus, for example says that MDD-SAT was able to solve 145 instances in time less than 10 seconds on the 16×16 (higher curves are better). A different view is provided when instances are sorted according to their runtimes for each individual tested algorithm (Figure 13). The lower curve now represents a better performing algorithm.

The first trend is that MDD-SAT significantly outperforms BASIC-SAT in all aspects. This shows the importance of developing enhanced SAT encodings for the MAPF problem. The performance of the BASIC-SAT encoding compared to the search-based algorithm degrades as the size of the grids grows larger: in the 8x8 grids it is second only to MDD-SAT, in the 16x16 grid it is comparable to most search-based algorithms, and in the 32x32 grid it is even substantially worse.

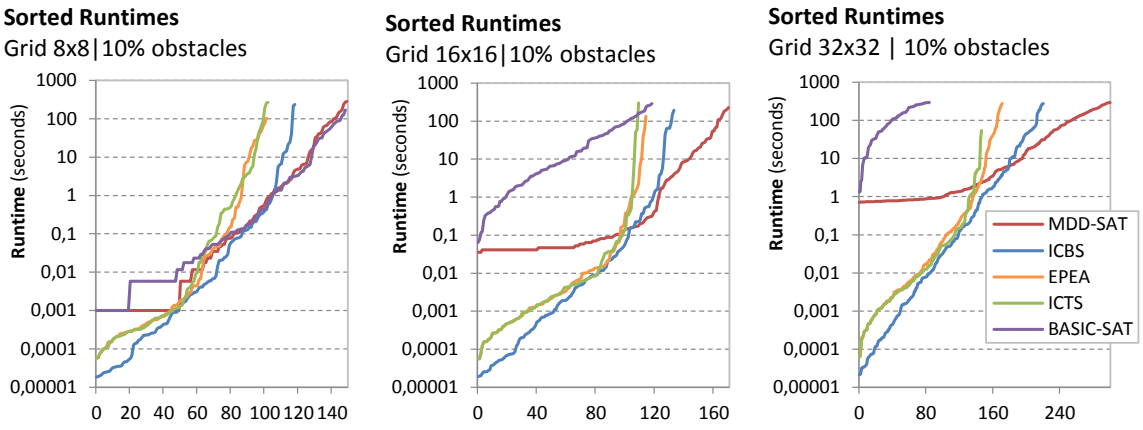


Figure 13: Sorted runtimes for 8×8, 16×16, and 32×32 grids (the horizontal axis corresponds to instances sorted differently for each tested algorithm).

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 particular, in highly constrained instances (containing many agents) the MDD-SAT solver is the best option.

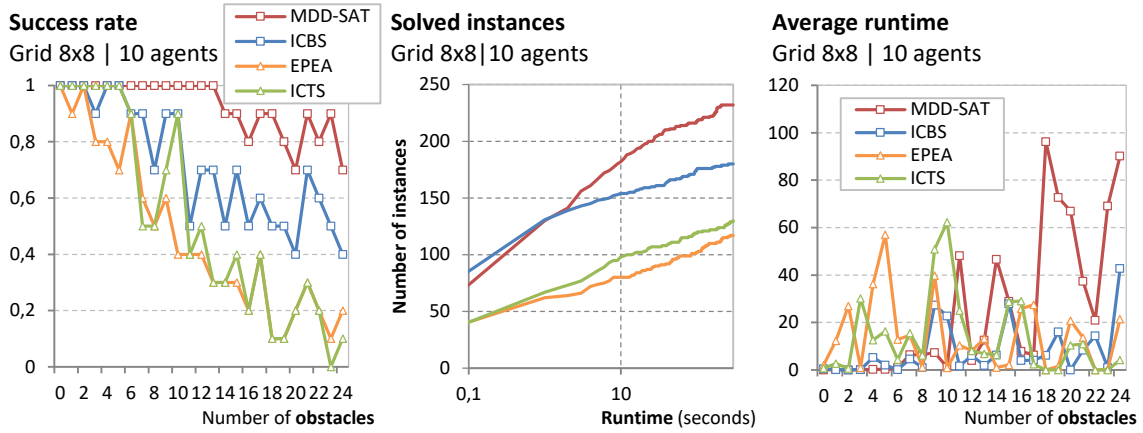


Figure 14: Success rate and runtime on the 8×8 grid with increasing number of obstacles (out of 64 cells).

However, on the 32×32 grid (rightmost figure) for easy instances when the available runtime was less than 10 seconds, MDD-SAT is weaker than the search-based algorithms. This is mostly due to the architecture of the MDD-SAT solver, which has an overhead of constructing the formula, running the external SAT solver, passing the input to it, and extracting the output from it (the truth-value assignment of variables needs to be interpreted back to agents' moves).

This effect is also seen in the 8x8 plot for the BASIC-SAT solver, which out of all solvers has the highest runtime for instances containing few agents. The effect of the overhead with calling the external SAT solver is multiplied by relatively inefficient formula based on TEGs in terms of its size.

Next, we varied the number of obstacles for the 8×8 grid with 10 agents to see the impact of shrinking free space and increasing the frequency of interactions among agents. Success rate, the number of solved instances, and the average runtime are shown in Figure 14. BASIC-SAT is omitted in this experiment due to its weaker performance from the previous test. MDD-SAT has the highest success rate except for some easy instances (that needed up to 1 second) where ICBS was slightly faster, which is again due to the overhead in the setup of SAT solving by an external solver.

We observe counter-intuitive behavior of MDD-SAT here. Increasing the number of obstacles reduces the number of open cells. This is an advantage for the SAT formula generator in MDD-SAT as the formula has fewer variables and constraints. By contrast, the combinatorial difficulty of instances increases with adding obstacles for all the solvers as it means that graphs gets denser with agents (i.e., the ratio of agents to possible agent positions gets higher) and harder to solve.

The rightmost part of Figure 14 shows average runtime for each number of agents out of 10 instances being solved under the time limit. Here we can see that MDD-SAT tends to use more time in more difficult instances while other search based solvers tend to finish quickly on easier instances while being not successful on harder ones.

5.2 MDD-SAT and the Makespan Objective

To provide more comprehensive picture we compare MDD-SAT with previous SAT-based solvers. However previous SAT-based solvers for MAPF using the MATCHING encoding (the solver called MATCHING-SAT) (Surynek, 2014a) and the DIRECT encoding (DIRECT-SAT) (Surynek, 2014b) are implemented for the makespan objective. Hence comparing MDD-SAT directly with these solvers is not relevant. Therefore we modified MDD-SAT for the makespan objective as first suggested by Surynek et al. (2016b). The modified solver will be denoted MDD-SAT(make).

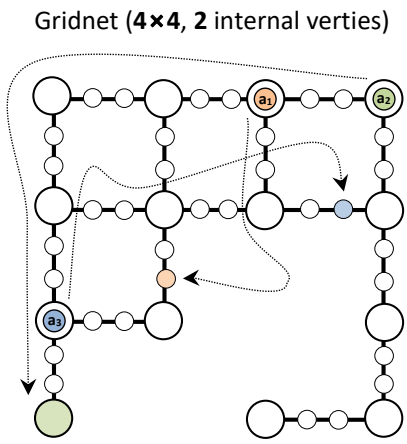


Figure 15: An example of MAPF with 3 agents on a $gridnet(4 \times 4, 2)$ graph originating from a 4×4 grid with obstacles.

The modification of MDD-SAT for the makespan objective consists in removing the cost bound ξ from the formula. That is, cardinality constraints concerning bounding ξ are omitted. The rest of iterative scheme that increases makespan μ is kept. In the course of execution of MDD-SAT(make), we again generate the sequence of MDDs. However these MDDs are different from those in MDD-SAT for the sum-of-costs where each MDDs is constructed for different cost $\xi_0(a_i) + \Delta$. We need to ensure that each agent can use all cost bounded only by common makespan μ applicable to all agents. Hence all MDDs in MDD-SAT(make) are constructed for cost $\mu = \mu_0 + \Delta$, where $\mu_0 = max_{i=1}^{k}\{\xi_0(a_i)\}$.

Agents for which $\xi_0(a_i) < \mu_0$ are given more freedom due to larger MDDs in MDD-SAT(make) compared to the original MDD-SAT. Due to this relaxation and absence of the overall cost bound represented by the cardinality constraint the resulting encodings in MDD-SAT(make) is simpler and less constrained. Therefore we hypothesize that MDD-SAT(make) will be faster than MDD-SAT.

We also modified search-based solvers ICTS, EPEA*, and ICBS for the makespan objective. Since these MAPF solving algorithms are defined for any cumulative cost, their adaptation for the makespan objective is done in straightforward way by replacing the calculation of sum-of-costs by the calculation of makespan. The modified algorithms are denoted ICTS(make), EPEA*(make), and ICBS(make).

Comparison of sum-of-costs and makespan variants of all above-mentioned algorithms on 4-connected grid 16×16 is shown in Figure 17 (upper part). The figure shows sorted runtimes for all instances solved under given timeout of 300 seconds (lower line is better). We put results for both objectives into the same plot to see both: comparison of different algorithms for single objective and comparison of single algorithm across different objectives.

Given the results, it cannot be universally said that the makespan objective is easier than sum-of-costs. EPEA* performs better than EPEA*(make) while in the case of ICTS and ICBS, the makespan objective is easier. Search-based solvers perform better than compilation-based solvers in easier instances but as the difficulty grows their runtime goes up faster than in compilation-based solvers.

The notable exception is MDD-SAT, which is competitive to search-based solvers even for easy instances.

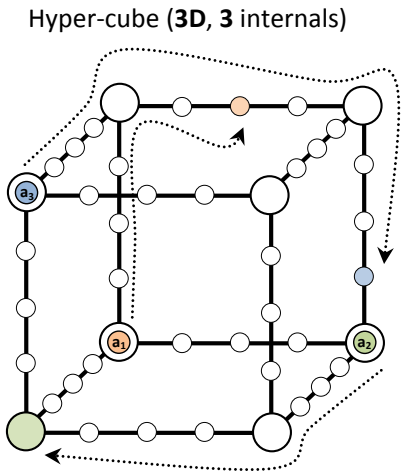


Figure 16: An example of MAPF with 3 agents on a $hypercube(3D, 3)$.

Comparison across SAT-based solvers shows that the previous MATCHING-SAT solver is far behind DIRECT-SAT and MDD-SAT(make) while the latter two solvers are very close to each other, still MDD-SAT(make) is faster. MATCHING-SAT uses the *log-space* encoding (Petke, 2015) for representation of the decision variables, which seemingly does not perform well due to poor *Boolean constraint propagation* (BCP) */unit propagation* (UP) (Dowling & Gallier, 1984) on top of this encoding. Both DIRECT-SAT and MDD-SAT(make) are based on the *direct encoding* (Tamura et al., 2009), which, on the other hand, supports BCP/UP well.

Several trends can be observed in the results. The hypothesis that MDD-SAT(make) performs better than MDD-SAT is confirmed by the results. However MDD-SAT(make) starts to prevail only in difficult problems where runtime of MDD-SAT grows much faster towards the time limit. The explanation of this behavior is that in larger and more difficult instances the cost bound represented by the cardinality constraint has more significant aggravating effect with respect to the performance.

In addition to square grids, we tested SAT-based solvers on other types of graphs representing different structure and connectivity to see whether it has any impact on the difficulty of finding optimal solution by a particular solver. We used two types of graphs:

- a **gridnet** graph, which can be constructed by starting with a 4-connected grid with obstacles (discussed previously) where edges are replaced by paths consisting of non-zero number internal vertices. Gridnet graphs originating from a 4-connected grid of size $m \times m$ where edges are replaced by paths with p internal vertices are denoted $gridnet(m \times m, p)$. Example of gridnet graph denoted $gridnet(4 \times 4, 2)$ is shown in Figure 15 - it is constructed from 4-connected grid 4×4 taken from Figure 1 where edges are replaced by paths containing 2 internal vertices.

- a **hyper-cube** graph, which is constructed by starting with a *hyper-cube* of certain dimension. Similarly to previous construction, we take the standard d-dimensional hyper-cube graph as a basis. Then each edge in the starting hype-cube is replaced by a path consisting of non-zero number of internal vertices. Analogously to the previous notation $hypercube(dD, p)$ denotes the hyper-cube graph originating from a d-dimmensional hyper-cube where edges are replaced by paths consisting of p internal vertices. An example of 3-dimensional hyper-cube (that is a cube) with 3 internal vertices per edge - denoted as $hypercube(3D, 3)$ - is shown in Figure 16.

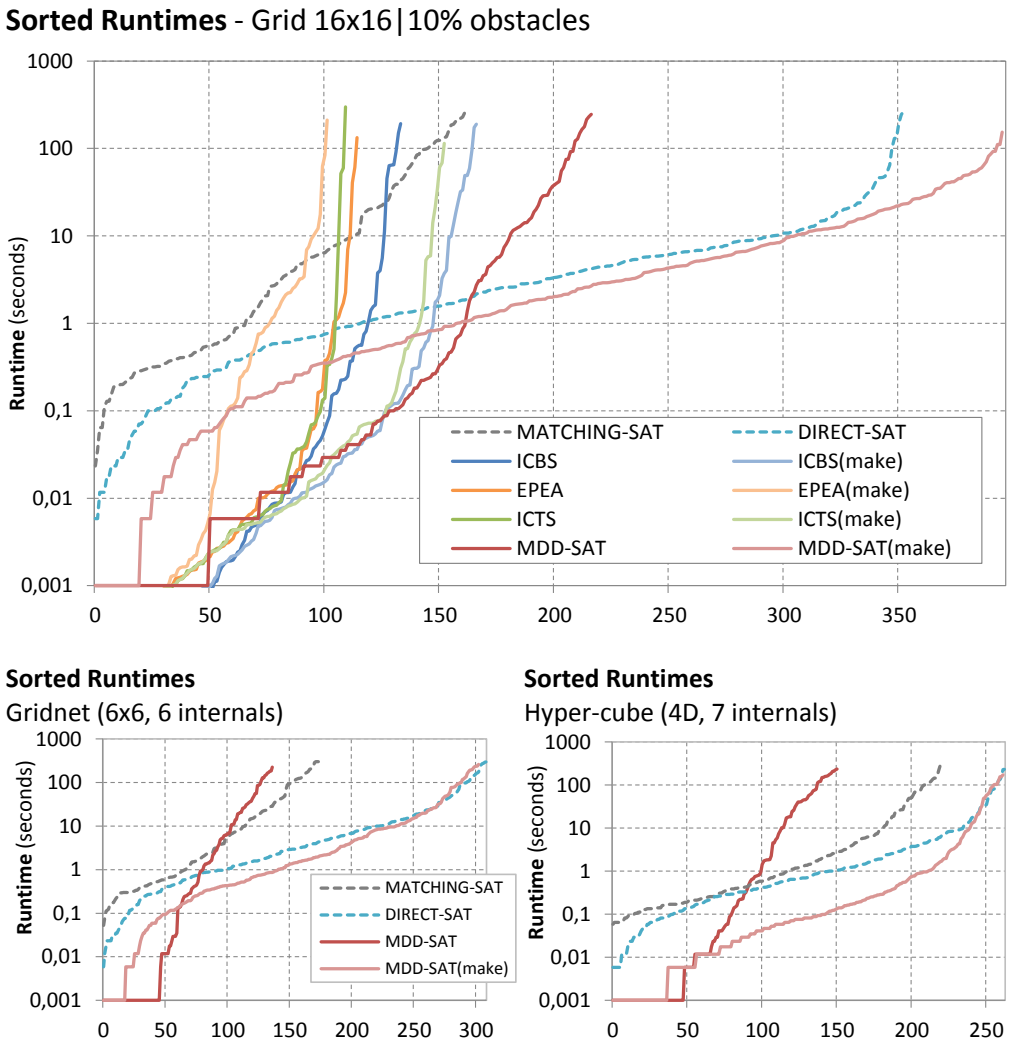


Figure 17: MDD-SAT with the makespan objective and the sum-of-costs objective compared to ICTS and ICBS and previous makespan-optimal SAT-based solvers MATCHING-SAT and DIRECT-SAT (lower line means better solver).

Paths replacing edges of the *high level* graph structure represent narrow corridors where it is expected that avoidance of agents will be more difficult than in the standard 4-connected grids. Moreover there is also practical motivation for such graphs. Floor-plans of many warehouses or shop-like spaces show similar structure where storage areas are surrounded by narrow aisles (Claes, Oliehoek, Baier, & Tuyls, 2017; Tsang, Ni, Wong, & Shi, 2018).

In the experiment with *gridnet* and *hyper-cube* graphs we need to omit comparison with ICTS and ICBS since their implementation does not support graphs other than grids. We also omit BASIC-SAT in this experiment. Results for $gridnet(6 \times 6, 6)$ and $hypercube(7D, 7)$ are shown in Figure 17 (lower part). The figure shows sorted runtimes of each tested solver across all instances solvable under the time limit (lower line corresponds to a better solver).

The relative ordering of individual solvers in terms of their performance for the makespan objective is almost the same as for the 16×16 grid. The notable difference is only diminishing difference between MDD-SAT(make) and DIRECT-SAT on hard instances. We attribute this to relatively long makespans in case of hard instances. Moreover in *gridnet* and *hyper-cube* tend to lead to long makespans because of the presence of narrow corridors in these graphs. When the makespan is long the effect of using MDDs compared to the TEG-based time expansion of the underlying graph is negligible.

The relative performance of sum-of-costs MDD-SAT compared to makespan-optimal solvers is far worse than in the 16×16 grid. While in the grid case MDD-SAT clearly dominates in easy instance no such clear dominance is observed in *hyper-cube* and only less significant dominance in easy cases is observed in the *gridnet* graph. Moreover the worst-performing makespan-optimal solver, MATCHING-SAT, eventually outperforms MDD-SAT for harder instances, which never happened in the grid case. We explain this trend by a great difference between the true optimal sum-of-costs and the lower bound estimation ξ_0 calculated as the sum of lengths of shortest paths: $\sum_{i=1}^{k} \xi_0(a_i)$. This difference is greater in case of *gridnet* and *hyper-cube* than in 4-connected grids due to narrow corridors where avoidance between agents is necessary. In such a case the lower bound estimation is inaccurate because it ignores this avoidance. This altogether leads to more iterations of the MDD-SAT algorithm. Moreover the sub-formula encoding the cardinality constraint representing the cost bound tends to be more complex in such a case.

5.3 Experiments with Large Maps

Experiments with MDD-SAT on large grid-based maps is presented in this section. We used benchmarks from Sturtevant's repository (Sturtevant, 2012) - three structurally different Dragon Age Origin (DAO) maps denoted as `brc202d`, `den520d`, and `ost003d`, which are a standard benchmark for MAPF (see Figure 18), are used. The size of the underlying grids for these maps is: 530×481 in `brc202d`, 256×257 in `den520d`, and 194×194 in `ost003d`. The relative size of the map belonging to the free space is depicted in Figure 18.

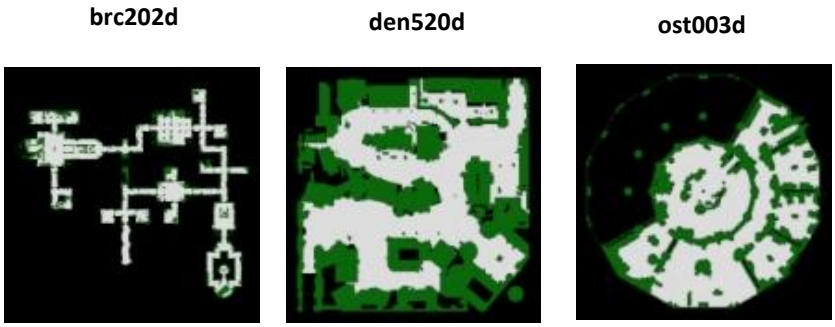


Figure 18: Dragon Age maps include: narrow corridors in `brc202d`, large open space in `den520d`, and open space with almost isolated rooms in `ost003d`.

The tests include optimal sum-of-costs algorithms MDD-SAT, ICTS and ICBS that were compared on instances containing 16, 24, and 32 agents in the selected DAO maps. To obtain instances

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)

of various difficulties we varied the average distance of agents from their goals. This particular measure is important for MDD-SAT as the distance of agent from its goal is the major parameter that determines the size of MDDs being generated during the solving process. The greater the distance is the larger MDD is produced and the instance is expected to be harder for MDD-SAT.

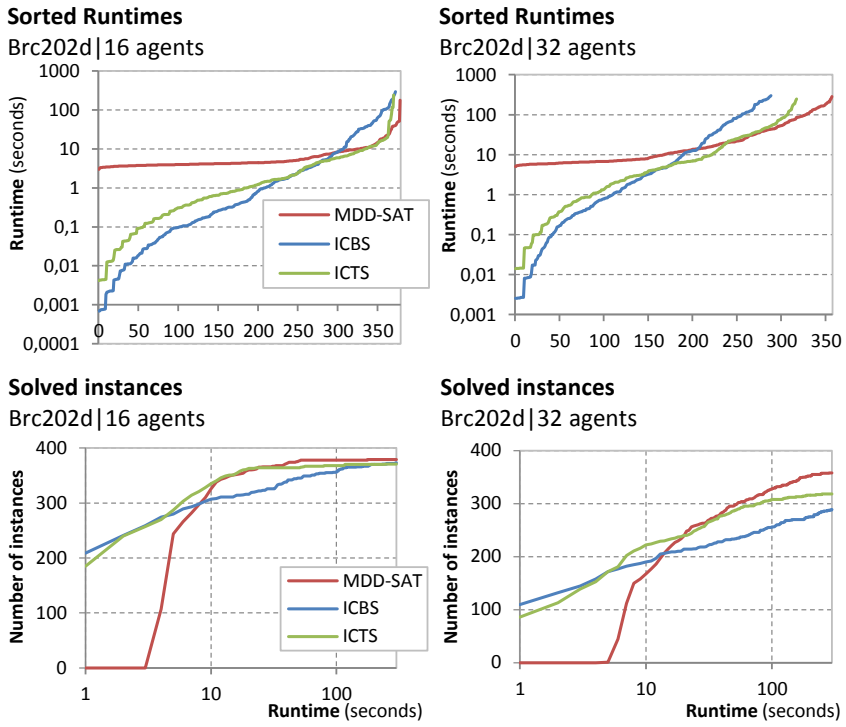


Figure 19: Runtime results for `brc202d` DAO map. Sorted runtimes (top) and number of solved instances as a function of time are shown (bottom).

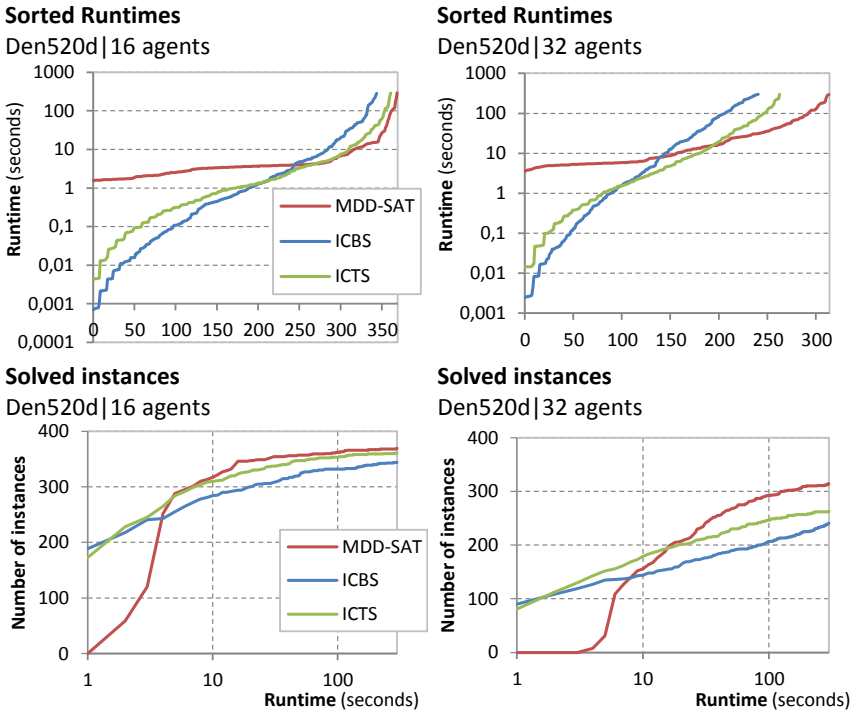


Figure 20: Runtime results for `den520d` DAO map. Sorted runtimes (top) and number of solved instances as a function of time are shown (bottom).

The initial positions of agents is set randomly. Given the average distance, the goals for agents were generated in two steps: **(i)** Potential goals for each agent were generated by long random walk in the graph in order to ensure solvability while for each potential goal its distance from the initial position was kept. **(ii)** Then random goal from potential goals for each agent was selected using the uniform distribution across distances from the initial position so that the given average distance is obtained. The mean distance from initial positions is varied from 8 up to 320 using step 8. Ten random instance were generated per distance. The maximum mean distance was set so that no of the tested algorithms was able to solve instances for the maximum average distance given the time limit of 300 seconds.

The illustration of the average size of formulae generated by MDD-SAT on `ost003d` is shown in Table 2 (right). For greater distance between the initial positions and the goals we can see rapid growth of the size of formulae. For the distance being increased twice from 64 to 128 the size of formulae in terms of the number of clauses increased 13-times for 32 agents, but 76 times for the 16 agents case. The size of formula is directly determined by the size of MDDs being generated whose size is proportional to the number of time expansions that is correlated with the agent's distance to the goal. The observed rapid growth of the size of formulae however needs additional explanation. We also need to take into account that using MDDs for great distances from the goal not only increases the makespan (the number of time expansions) but also can make more vertices accessible at individual layers of MDD.

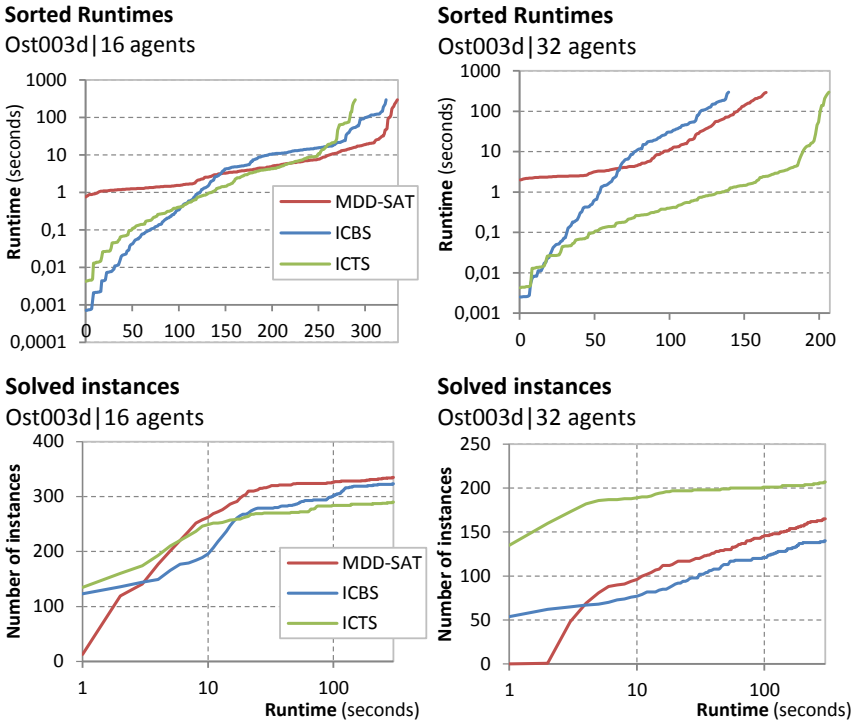


Figure 21: Runtime results for `ost003d` DAO map. Sorted runtimes (top) and number of solved instances as a function of time are shown (bottom).

Runtime results for MDD-SAT, ICTS and ICBS on large maps are presented in Figures 19, 20, and 21. The figures show the number of solved instances as a function of runtime (left - the higher line means better solver) and a different view showing sorted runtimes (right - higher line means better solver).

Two clear trends can be observed in the results. MDD-SAT is weak in easy instances where it is outperformed by both ICTS and ICBS. This observation is universal across all types of maps.

The explanation for poor performance of MDD-SAT on easy instances is that it consumes a lot of runtime for constructing MDDs with respect to large input map while at the same time the combinatorial difficulty of such instances is not significant hence the SAT solver inside MDD-SAT has an easy job. The benefit of using strong external solver is little in such case.

The second strong trend is that as the average distance of agents from their goals increases and instances become harder, the MDD-SAT solver starts to show its strength. MDD-SAT often outperforms other two search-based solvers though not always as in the case of `ost003d` with 32 agents where ICTS dominates. The explanation for having stronger MDD-SAT in harder instances is that in such cases the external SAT solver is employed in longer runs where its techniques like learning and constraint propagation have chance to prune the search space more significantly. In other words, not all runtime is consumed by the overhead of constructing MDDs and formulae from it and significant time is spend more efficiently during SAT solving.

Interestingly, comparison between ICTS and ICBS indicates a pattern in which ICBS is better in easy instances but is dominated by ICTS in harder ones. This suggests an explanation that represent-

ing the search space as MDDs, which is done in both MDD-SAT and ICTS and performing search on top of this data structure is beneficial.

More detailed results concerning runtimes on individual random instances for specific distance from the goal are shown in the left part of Table 2. Although both search-based solvers ICBS and ICTS achieve better runtimes when they manage to solve the instance, the success rate is eventually better for MDD-SAT. The explanation of this behaviour is that MDD-SAT is more likely to succeed in a hard instance requiring longer runtime.

Altogether we cannot say that there is universal winner across tested solvers in large maps as for different setups different solver turns out to be most promising.

5.4 In-Depth Evaluation of MDD-SAT

In this section, we provide a more detailed analysis of MDD-SAT's performance in order to understand it more deeply, including analyses of:

1. the solving runtime per time expansion,
2. the relation between the number of time expansions needed to prove optimality and the makespan of the optimal solution, and
3. the impact of using different SAT solvers on the performance of MDD-SAT.

5.4.1 EVALUATION OF RUNTIME PER TIME EXPANSION

To provide a better insight in how MDD-SAT works we evaluate its runtime per individual iteration of ξ to verify whether the adopted scheme of incrementing ξ by one is a good choice or whether there is a room for adopting a different scheme. MDD-SAT starts with ξ that equals to the lower bound ξ_0 ($\Delta = 0$) and usually continues by answering the satisfiability of $\mathcal{F}(\mu_0, \xi_0, \Delta)$ for few increments of ξ (Δ) by one until the first satisfiable $\mathcal{F}(\mu_0, \xi_0, \Delta)$ is found. The answers from the underlying SAT solver hence form a monotonic sequence of negative answers (unsatisfiable) followed by one positive answer (satisfiable).

Results from classical planning with SATPlan and SAT-based related planners (Kautz & Selman, 1992; Kautz, McAllester, & Selman, 1996; Kautz, 2006; Rintanen, 2012) indicate that difficulty of satisfiability testing of formulae modelling the existence of bounded-step plan exhibits a phase transition behavior where the phase changes on the boundary between the unsatisfiable and the satisfiable case. Namely the unsatisfiable case exhibits the exponential growth of difficulty (runtime) as a function of the bound while the satisfiable case is usually easier and more moderate growth of difficulty can be observed there.

Our hypothesis is that the difficulty of SAT solving within MDD-SAT behaves similarly. In order to verify this, we modified MDD-SAT to add few extra iterations of ξ (Δ) above the first satisfiable formula and measured runtimes per iteration in this satisfiable phase.

Results are for the 8×8, 16×16, and 32×32 grids with 10% random obstacles are shown in Figure 22 and results for DAO maps `brc202d`, `den520d`, and `ost003d` are shown in Figure 23. Three sizes of the set of agents for each map are selected for the presentation. We select the sizes of the set of agents for presentation according to the fact that in these cases MDD-SAT exhibits relatively many iterations in the unsatisfiable phase.

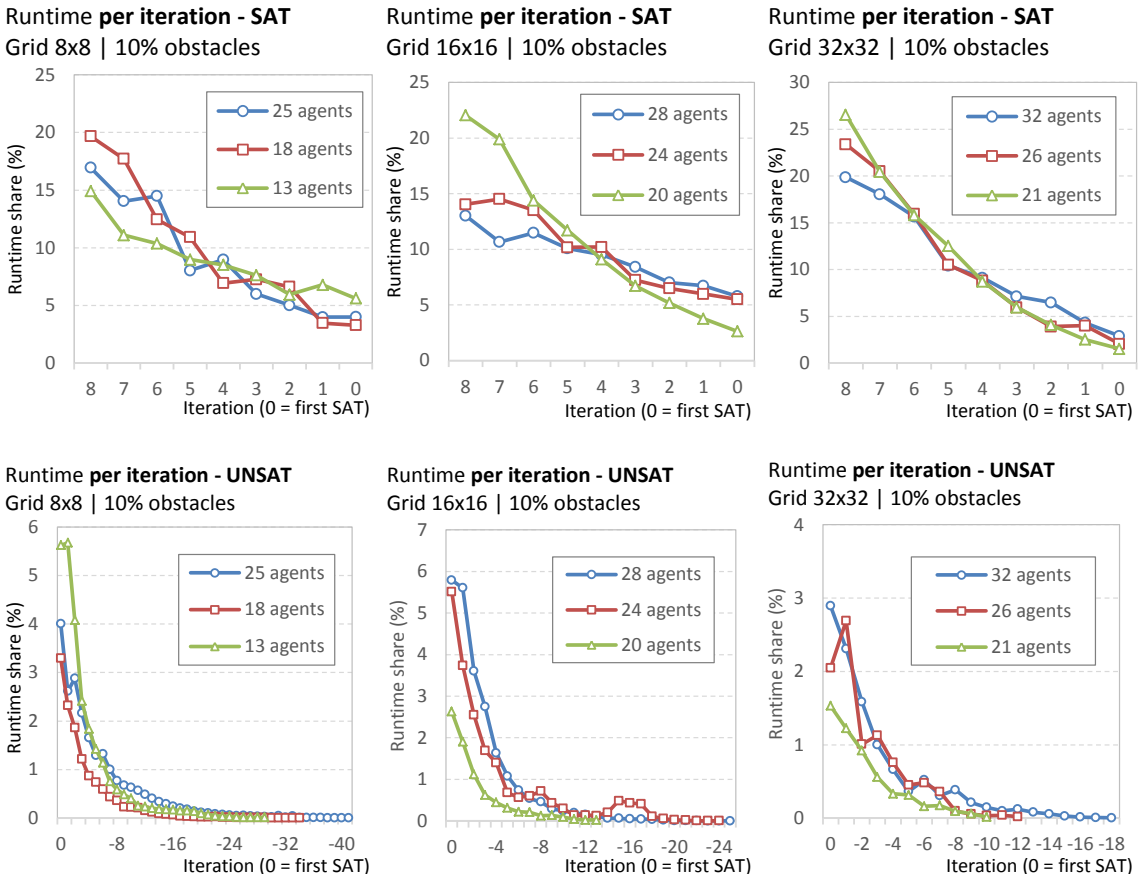


Figure 22: Runtime results for individual iterations in 8×8, 16×16, and 32×32 grids. The satisfiable (top) and the unsatisfiable (bottom) phases are shown. Eight extra iterations in the satisfiable phase are evaluated. The ratio of the total runtime consumed by each iteration is shown.

The horizontal axis shows iterations of ξ (Δ) indexed from the first satisfiable iteration that is assigned the index 0, iterations in the satisfiable phase are assigned positive indices while the unsatisfiable phase is assigned negative indices. The results for each grid/map are divided into separate plots for the satisfiable phase (upper plot) and for the unsatisfiable phase (lower plot). MDD-SAT runs for 8 extra iterations of ξ (Δ) above the first satisfiable one. The plots show the ratio of the total runtime consumed per iteration (the sum of runtimes in all iterations corresponds to 1.0). The results are averaged through 10 random instances per each number of agents.

The results show that in small grids the runtime in the unsatisfiable case exhibits an exponential growth as getting closer to the first satisfiable formula (lower three plots in Figure 22). After reaching the satisfiable phase the growth of rutimes is slower (upper three plots in Figure 22). Similar results can be observed for large maps in Figure 23 though there the hypothesized behaviour is less pronounced.

This result experimentally verifies that the adopted incremental scheme is suitable as answering by the SAT solver the formulae around the boundary between the unsatisfiable and the satisfiable

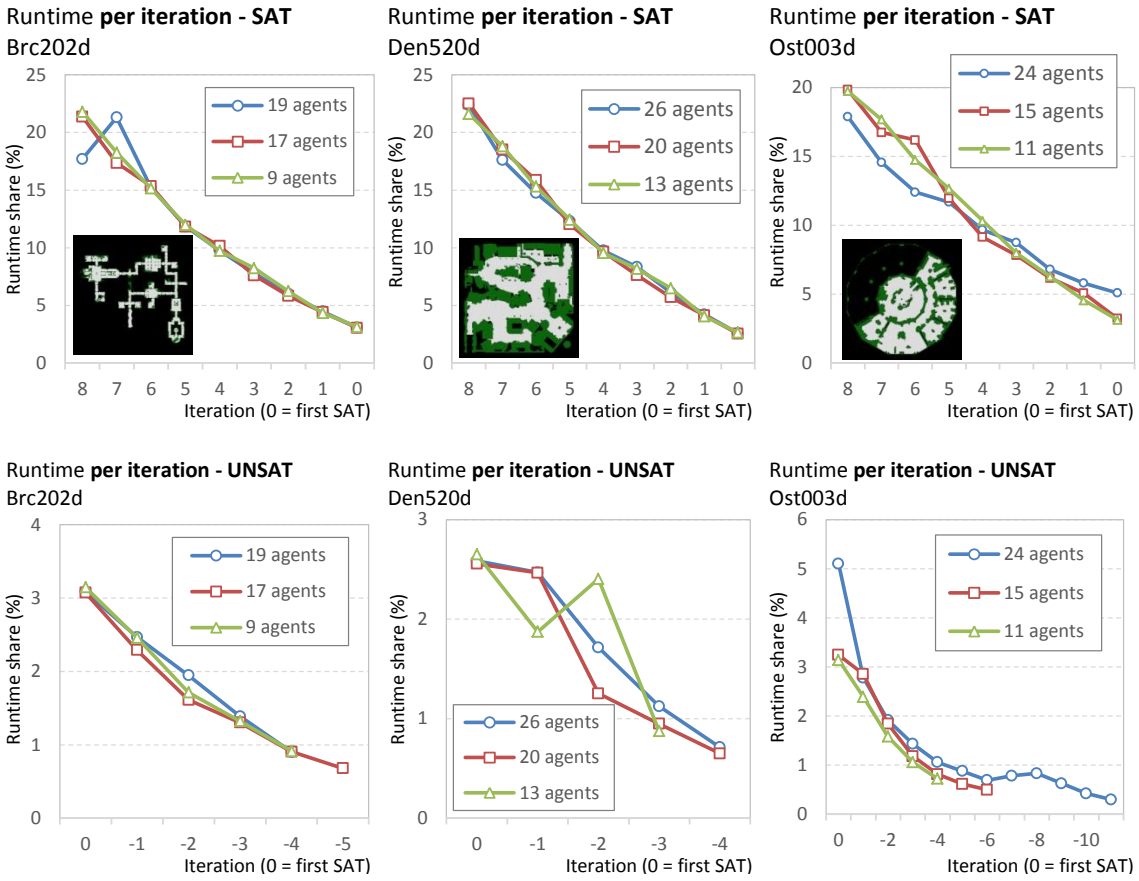


Figure 23: Runtime results for individual iterations in DAO maps `brc202d`, `den520d`, and `ost003d`. The satisfiable (top) and the unsatisfiable (bottom) phases are shown. Eight extra iterations in the satisfiable phase are evaluated. The ratio of the total runtime consumed by each iteration is shown.

case is as hard as to answer all formulae in the unsatisfiable phase. Different strategies that attempt to go further in the satisfiable phase could only add significant computational effort even by single consultation with the SAT solver.

5.4.2 EVALUATION OF TIME EXPANSIONS

The number of MDD time expansions (denoted μ in the pseudo-codes) needed to find an optimal solution may either be greater than or equal to the makespan of that optimal solution. Intuitively said the MDD-SAT algorithm needs to permit that all the extra cost when Δ is consumed by a single agent, resulting in the makespan of $\mu_0 + \Delta$. Since the runtime of our algorithm depends strongly on the number of time expansions, the gap between the number of time expansions and the makespan of the optimal solution suggests that suboptimal solutions (or even optimal solutions without proving they are optimal) may be found faster by having fewer time expansions.

Figures 24 and 25 plot the difference between the optimal makespan and number of time expansions done by our algorithm, for different MAPF configurations (number of agents, grid/map type). Results are aggregated for 10 random instances in a box-plot showing the median, the 1st and the 3rd quartil, maximum and minimum, and outliers.

As can be seen, the gap grows as we increase the number of agents. This trend is especially visible in grids. This suggests that a suboptimal variant of our algorithm in which we relax the cost bound constraint may be able to find valid solutions much faster for harder problems. We explore this option in Section 7.

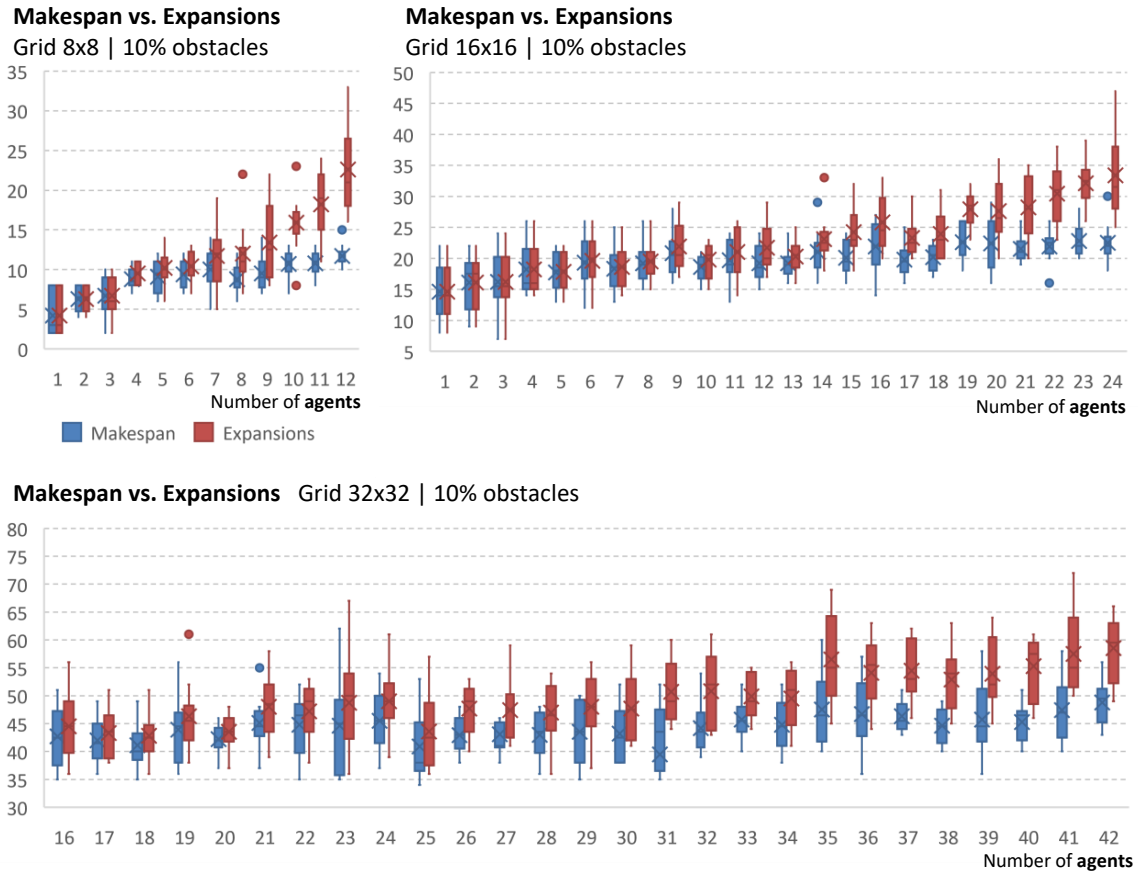


Figure 24: Comparison of computed makespan in sum-of-costs optimal solutions and the number of time expansions of MDDs in 8×8, 16×16, and 32×32 grids.

5.4.3 REFLECTING ADVANCES OF RECENT SAT SOLVERS

The major advantage of SAT-based solvers in general such as MDD-SAT is that they are modular and it is easy to exchange the SAT module for a newer one. The SAT-based solvers expected to perform better as the state of the art in SAT solving advances. In fact, in the course of writing

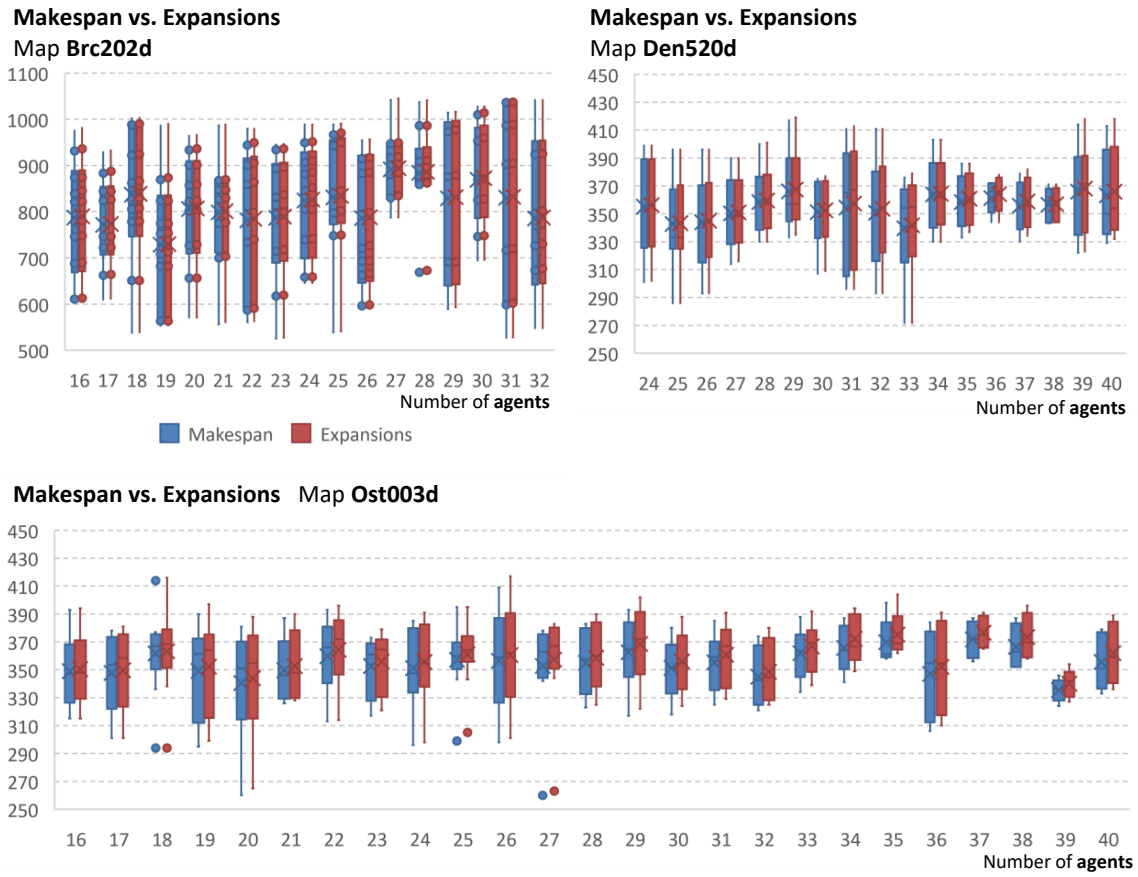


Figure 25: Comparison of computed makespan in sum-of-costs optimal solutions and the number of time expansions of MDDs in DAO maps `brc202d`, `den520d`, and `ost003d`.

the paper, a new SAT solver appeared, namely `MapleCOMSPS` (Liang, 2018; Liang, Oh, Mathew, Thomas, Li, & Ganesh, 2018), that according to the results in recent SAT competitions outperforms the `Glucose` SAT solver (Balyo et al., 2017), originally used in MDD-SAT. Indeed, the integration of this new solver resulted in an improvement of MDD-SAT in some cases.

To demonstrate this, we run MDD-SAT with the `MapleCOMSPS` SAT solver and compared it with MDD-SAT using the `Glucose` SAT solver (that is the SAT solver used in all the other experiments). We performed this comparison in a range of maps, including the 8×8, 16×16, and 32×32 grids with 10% obstacles and all the DAO maps used in our experiments above. There were no significant difference in performance for the grids. However, we have observed a significant advantage for MDD-SAT with the `MapleCOMSPS` SAT solver on the DAO maps. These results are given in Figure 26, which follows the same format as in Figure 21. The results suggests that the `MapleCOMSPS` SAT solver can tackle large formulae resulting from solving MAPF on large DAO maps more efficiently than the `Glucose` SAT solver.

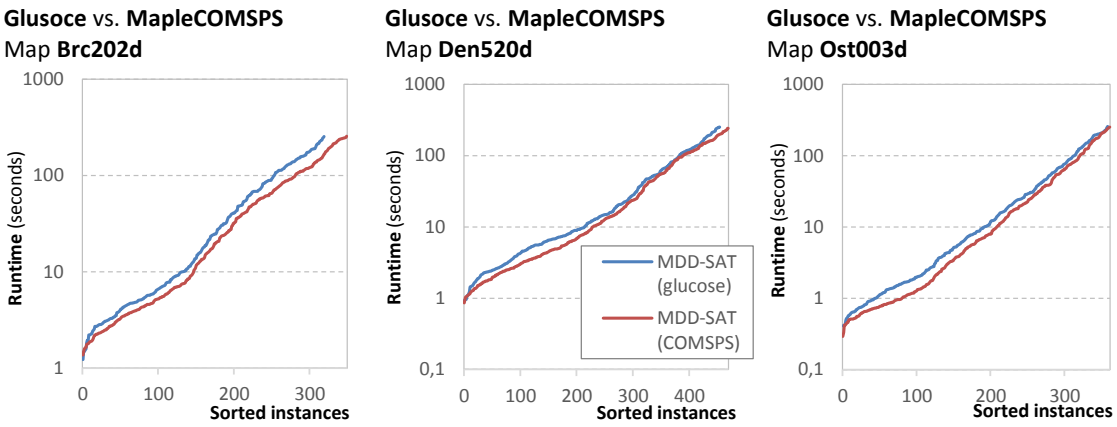


Figure 26: Sorted runtimes of MDD-SAT with the Glucose SAT solver and MDD-SAT with the MapleCOMSPS for DAO maps `brc202d`, `den520d`, and `ost003d`.

6. Independence Detection in SAT-Based Approach

A successful method for increasing performance of MAPF solving algorithms is a technique called *independence detection* (ID). In this section, we describe how to integrate independence detection for MAPF (Standley, 2010b; Standley & Korf, 2011) into MDD-SAT. We summarize here and extend the results originally published as conference papers (Surynek et al., 2017b, 2017c).

The main idea behind this technique is that the difficulty of optimal MAPF solving grows **exponentially** with the number of agents. It would be ideal, if we could divide the problem into a series of smaller independent sub-problems, solve them independently, and merge solutions to the sub-problems to form a solution of the original problem. Having exponential complexity for the original instance such decomposition could lead to significant speed up.

More precisely, assume that the time complexity of solving the given MAPF problem with k agents is $\mathcal{O}(2^k)$. Then the problem is decomposed into say two independent sub-problems with $k/2$ agents and these two sub-problems are solved separately taking time of $\mathcal{O}(2^{\frac{k}{2}}) + \mathcal{O}(2^{\frac{k}{2}}) = \mathcal{O}(2^{\frac{k}{2}})$. If the dividing phase and the final sub-solution merging phase have lower than exponential time complexity, for example polynomial in the number of agents, then the overall speed up could be $\mathcal{O}(2^{\frac{k}{2}})$-fold.

The important assumption that only some problems satisfy is the requirement of having independent sub-problems that can be solved separately. In MAPF fortunately, this is relatively common situation in sparse environments. Consider groups of agents in distant sub-graphs whose goals are located in the same sub-graph. Then it is unlikely that these groups will ever have chance to meet each other in an optimal solution. Hence optimal solution to the problem can be constructed as the union of optimal solutions for individual groups.

6.0.1 INDEPENDENCE DETECTION VARIANTS IN SEARCH-BASED SOLVERS

For completeness, we first give a detailed description of ID as it has been described by prior work in a search-based MAPF solver, namely A* (Standley, 2010a). The basic approach, called *simple*

independence detection (SID), divides agents into multiple groups. Initially, each agent is assigned into its own group (consisting of a single agent). The algorithm regards each of these groups as independent sub-problem and searches for optimal MAPF solutions of individual groups. Solutions obtained from this independent solving are then checked for conflicts. For every pair of solutions, it checks if a collision between agents from different solutions/groups occurs.

If a collision occurs then respective groups are merged together and new optimal solution is found for the merged group. The process is repeated until there are no conflicts between group solutions. If there are no conflicting solutions, the solutions for individual groups can be merged together to from a single solution of the original problem.

Merging groups together after detecting a conflict could quickly lead to large groups (an extreme case is having single group consisting of all agents). The SID approach can be further improved by avoiding group merging if possible.

Generally, there exists more than one optimal path of each agent or more than one optimal MAPF solutions for a group of agents. The SID technique, however, ignores these alternative paths and solutions. The improvement of SID known as *independence detection* (ID) is as follows.

Algorithm 4: MAPF solving algorithm based on independence detection technique. Planning for groups is always done to have the least number of conflicts with respect to conflict avoidance table.

```
1  Solve-MAPF-A*+ID(Σ = (G = (V, E), A, α0, α+))
2      for each ai ∈ A do
3          Gi ← {ai}
4          π(Gi) ←plan-MAPF-A*(G = (V, E), Gi, α0, α+)
5      while Gi and Gj exist such that π(Gi) and π(Gj) conflict do
6          if π(Gi) and π(Gj) not conflicted before then
7              π'(Gi) ←replan-MAPF-A*(G = (V, E), Gi, α0, α+, π, Gj)
8              if π'(Gi) =Fail then
9                  π(Gj) ←replan-MAPF-A*(G = (V, E), Gj, α0, α+, π, Gi)
10             else
11                 π(Gi) ← π'(Gi)
12         else
13             π(Gi) ← Fail
14             π(Gj) ←Fail
15         if π(Gi) = Fail or π(Gj) = Fail then
16             Gi ← Gi ∪ Gj
17             Gj ← ∅
18             π(Gi) ←plan-MAPF-A*(G = (V, E), Gi, α0, α+)
19     return π
```

Assume two conflicting groups of agents G_i and G_j. We try to avoid merging G_i with G_j via re-planning. First, we try to re-plan for G_i so that the new solution has the same cost but conflicts with G_j are forbidden. If no such solution exists, we try to replan for G_j. If re-planning for G_j fails as well, then we merge G_i and G_j into a new group. After successful re-planning, the new MAPF solution for the group needs to be evaluated with every other group again. This could lead to infinite cycle. Therefore, if two groups were in conflict before, we merge them without trying to replan.

The original ID technique was used in combination with the A* algorithm that was used for path finding (searching for solutions for individual groups). The independence detection can be further supported at the level of path finding. The A* algorithm can be fine tuned to prefer paths that create as few as possible conflicts with other groups that have their solutions finished.

The MAPF solving algorithm based on A* and ID is listed using pseudo-code as Algorithm 4. Let us note that initial paths for a group are found while ignoring all agents outside the group (lines 4 and 13) but when re-planning for a group we take into account existing paths for other groups including the group towards which we are trying to resolve conflicts (lines 7 and 9). When A* has a choice between several nodes with the same minimum $f()$ cost in the re-planning phase, the one with the least amount of conflicts with respect to other group's paths is expanded first. This technique, which is also known as using a Conflict Avoidance Table (CAT), yields an optimal solution that has the minimal number of conflicts with other groups. In other words, the re-planning phase strictly avoids paths of the other group in conflicting pair but also prefers avoiding paths of other groups though this is a preference only (not a strict constraint).

Both SID and ID do not solve MAPF on their own, they only divide the problem into smaller sub-problems that are solved by any possible MAPF algorithms. Thus, ID and SID are general frameworks which can be executed on top of any MAPF solver.

6.0.2 INTEGRATION OF INDEPENDENCE DETECTION INTO MDD-SAT

Next, we describe how ID can be applied in SAT based MAPF solvers. The common feature of SAT based MAPF solvers, MDD-SAT included, is that they consider the entire MAPF instance as a whole which can limit their scalability for large instances.

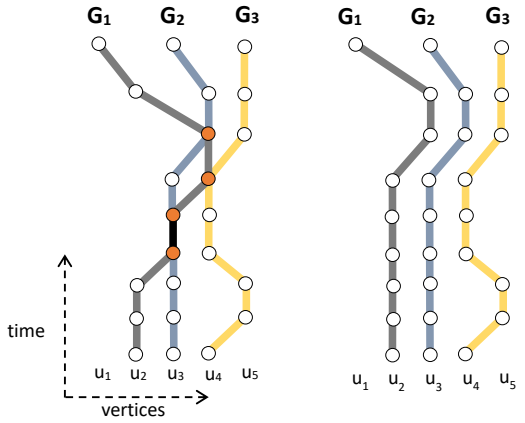


Figure 27: Groups G_1 conflicts with groups G_2 and G_3 (left). After replanning G_1 independent solutions for G_1, G_2, and G_3 can be merged together.

With large instances and many agents, MDD-SAT will eventually encounter a formula of prohibitive size even with the use of MDDs. In contrast to this, search-based solvers often use some variant of ID to further mitigate the size of the instance needed to be tackled at once.

The logical step is hence to integrate a variant of ID into MDD-SAT. The SAT-based approach however requires modification of the original ID since in the propositional formula it is not possible to express preference that individual paths of groups of agents should avoid the occupied positions by other groups. In the yes/no SAT environment we either manage to avoid occupied positions or not while in the negative case there is no tool how to control the number of conflicts. In other words, in our implementation of ID there we do not use a CAT to bias the paths returned for the agents.

The SAT-based version of ID works in a similar way to the original version but instead

of resolving conflicts between a pair of conflicting groups G_i and G_j it resolves conflict of group G_i with all other groups. If this attempt is successful, G_i is independent on others and the process can continue with resolving conflicts between remaining groups (see Figure 27 where G_1 has been made independent of G_2 and G_3). If the attempt to resolve conflicts between G_i and G_j by making G_i independent of all other groups fails, the same is tried for G_j. If the attempt for G_j fails too, then G_i and G_j are merged. The pseudo-code of the process is shown in Algorithm 5.

In contrast to the original ID we strictly require avoidance with respect to other groups. This is technically done by omitting the conflicting vertices in the MDD. More precisely, when planning for group G_i (line 7) we first build MDDs for agents $a \in G_i$. For each node in MDD we need to check whether it conflicts with paths for agents in the remaining groups.

Assume node u_j^t in MDD for agent a corresponding to $u_j \in V$ and time step t. Several cases need to be distinguished [4]:

- Case **(i)** occurs if vertex u_j is occupied by some agent $a' \in A \setminus G_i$ at time step t (that is, $\pi(a')[t] = u_j$; where $\pi(a')[t]$ denotes the t-th vertex of path for agent a'). This situation corresponds to violation of the constraint that at most one agent reside in each vertex. If this is detected then u_j^t is omitted from given MDD for agent a.
- Case **(ii)** occurs if vertex u_j was occupied by some agent $a' \in A \setminus G_i$ at time step $t-1$ which corresponds to violation of the constraint that agent can enter a vacant vertex only. Again in such case using u_j at time step t is forbidden for agent a and thus u_j^t is omitted from MDD.

The rest is automatically expressed by the semantics of the formula. Occupied vertices omitted from MDD are not represented in the formula. Hence the SAT solver cannot use these vertices for planning agents' paths. If the SAT solver succeeds with path finding then resulting paths automatically avoid missing vertices and resulting paths do not conflict with paths of other groups.

The resulting algorithm is called MDD-SAT+ID. The algorithm returns sum-of-costs optimal solutions for solvable MAPF instances. The optimality guarantees are a direct consequence of properties of ID. The ID technique can be also used with the makespan-optimal version of MDD-SAT.

6.1 Experimental Results for MDD-SAT with Independence Detection

We followed the same pattern in experiments for ID as in previous sections. Experiments were done on small 4-connected grids with obstacles and on large DAO maps. We implemented extensions of MDD-SAT using both simple independence detection (the solver denoted MDD-SAT+SID) and MDD-SAT with a independence detection (denoted MDD-SAT+ID). The implementation extends previous C++ implementation of MDD-SAT. The SID version only replans the conflicting group against the single group with which the conflict has been detected while other groups are ignored. We implement MDD-SAT+SID for having a reference base-line variant of independence detection. MDD-SAT+SID is expected to perform worse than MDD-SAT+ID where re-planning of conflicting group is done with respect to all remaining groups. MDD-SAT was also included in comparison too to see what is the benefit of ID in relation to its overhead. In addition to this, we included an implementation of Algorithm 4 referred to as A*+ID in the plots. A*+ID implements operator

4. Depending on the variant of MAPF the cases can be different. Here we follow the move-to-unoccupied variant where agents move to vacant vertices.

Algorithm 5: Independence detection in the sum-of-cost optimal SAT-based solver MDD-SAT. Conflict avoidance is strictly required.

```
1  Solve-MAPF-SAT_SOC+ID(Σ = (G = (V, E), A, α_0, α_+))
2      for each a_i ∈ A do
3          G_i ← {a_i}
4          π(G_i) ←Solve-MAPF-SAT_SOC(G = (V, E), G_i, α_0, α_+)
5      while G_i and G_j exist such that π(G_i) and π(G_j) conflict do
6          if π(G_i) and π(G_j) not conflicted before then
7              π'(G_i) ←reSolve-MAPF-SAT_SOC(G = (V, E), G_i, α_0, α_+, π)
8              if π(G_i) =Fail then
9                  π(G_j) ←reSolve-MAPF-SAT_SOC(G = (V, E), G_j, α_0, α_+, π)
10             else
11                 π(G_i) ← π'(G_i)
12         else
13             π(G_i) ← Fail
14             π(G_j) ←Fail
15         if π(G_i) = Fail or π(G_j) = Fail then
16             G_i ← G_i ∪ G_j
17             G_j ← ∅
18             π(G_i) ←Solve-MAPF-SAT_SOC(G = (V, E), G_i, α_0, α_+)
19     return π
```

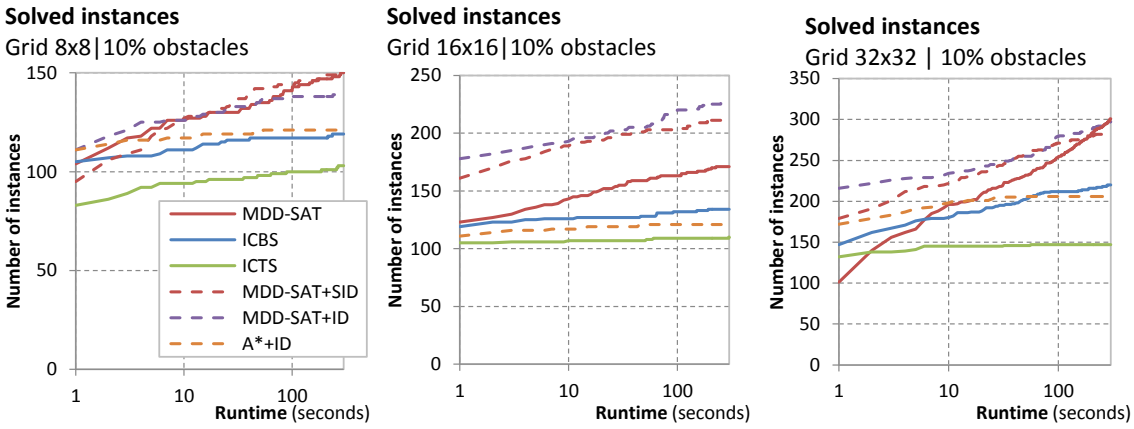


Figure 28: Evaluation of ID and SID integrated in MDD-SAT - the number of solvable instances in 8×8, 16×16, and 32×32 grids as a function of the time limit (the horizontal runtime axis uses logarithmic scale).

decomposition OD in the A*-based search for non-conflicting paths (procedures plan-MAPF-A* and replan-MAPF-A*), hence A*+ID corresponds to Standley's OD+ID (Standley, 2010a).

In all tests we used time limit of 300 seconds. Testing instances for both the small grids and large DAO maps are identical to those used in previous tests. The hypothesis is that independence

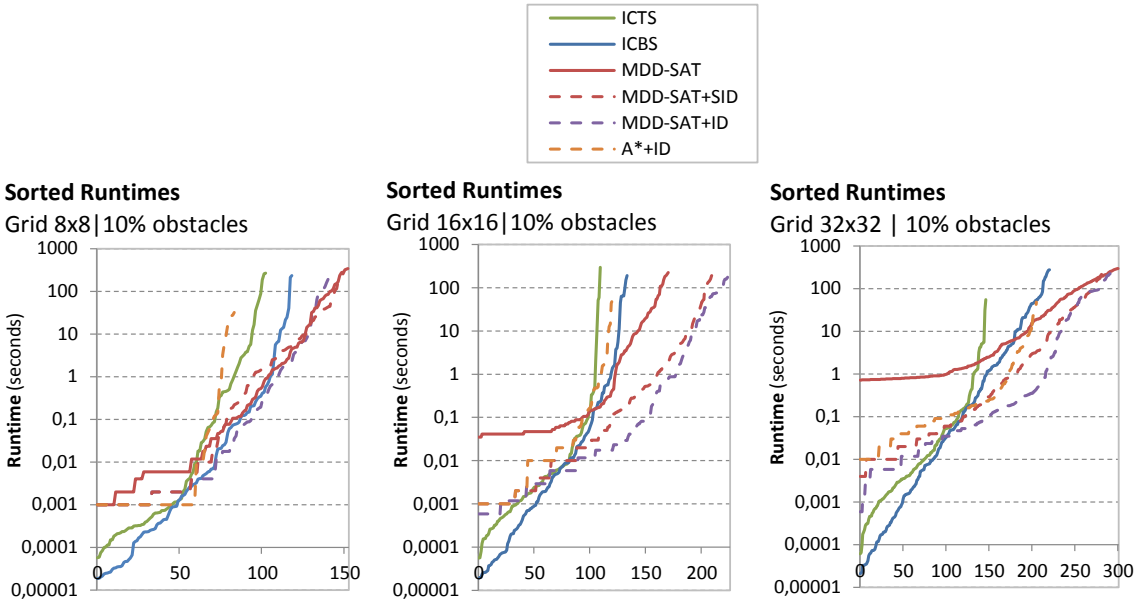


Figure 29: Evaluation of ID and SID integrated in MDD-SAT - sorted runtimes for 8×8, 16×16, and 32×32 grids (the horizontal axis corresponds to instances sorted differently for each tested algorithm).

detection could be helpful in instances with few agents and lot of free space where there is a little interaction among agents and/or the environment provides opportunity to perform a collision free avoidance if necessary. On the other hand, detecting independence could represent a significant overhead on small densely populated maps where eventually all agents merge into one large group.

6.1.1 RESULTS OF INDEPENDENCE DETECTION ON SMALL GRIDS

Results obtained with independence detection integrated in MDD-SAT on small grids of sizes 8×8, 16×16, and 32×32 are presented in Figures 28 and 29. We show two projections of the results: Figure 28 shows the number of successfully solved instances as a function of time. In Figure 29 we take all instances that the selected solver managed to finish under the given time limit and sort them according to the ascending runtime.

The trends we can observe in the result supports the hypothesis. On 8×8, MDD-SAT+SID represents almost no significant improvement compared to MDD-SAT. MDD-SAT+ID even shows worsening in harder instances (needing longer runtime) compared to MDD-SAT. In easy instances we can observe that ID contributes to some minor improvement but SID does the opposite. We can attribute this behavior to the situation where agents do interact so collision avoidance is needed but there is enough free space at the same time to plan collision avoidance successfully (the ID case) but there is no abundance of free space so SID often hits some other group after re-planning.

In larger girds, 16×16, and 32×32 results suggest that different effect takes places. We can observe two significant patterns: (i) there is a consistent significant benefit of using both SID and ID

across tested instances while the improvement decreases towards harder instances containing more agents, (ii) in instances of medium difficulty ID provides better results than SID.

The explanation for pattern (i) is that larger free space gives opportunity to independence detection to actually find independent groups of agents and finish searching with relatively small groups (small independent sub-problems). The region where pattern (ii) can be observed corresponds to the situation where we have enough free space for avoidance between groups but such free space is not abundant so all groups need to be carefully taken into account (like in the ID case) since otherwise some other group may be hit by the re-planned group (the SID case).

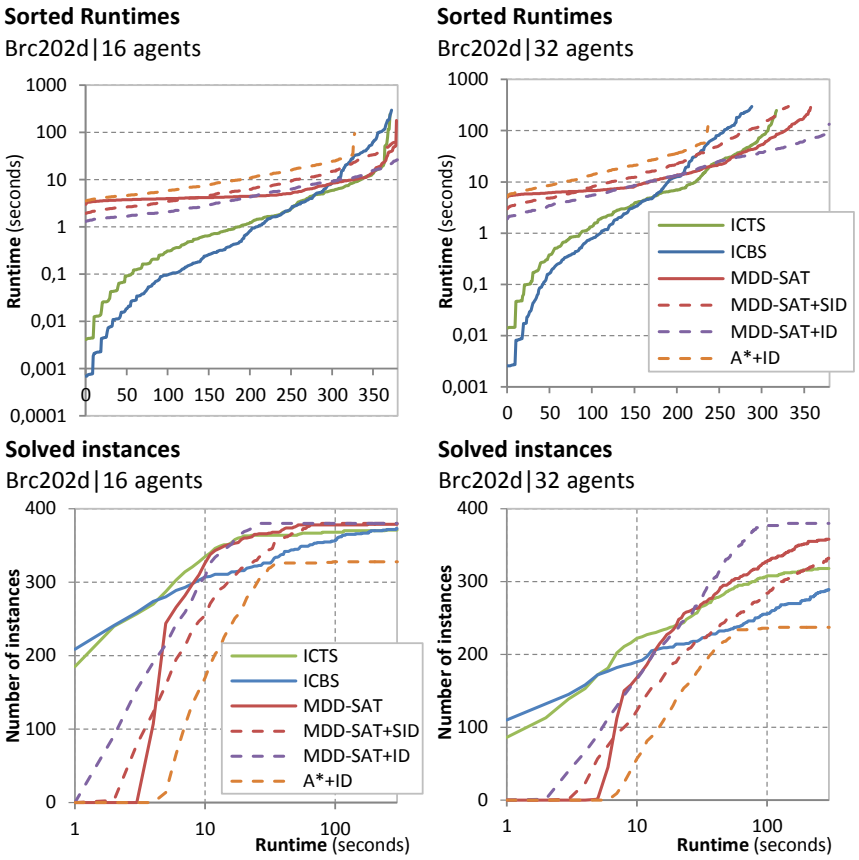


Figure 30: Runtime results with ID and SID for `brc202d` DAO map. Sorted runtimes (top) and the number of solved instances as a function of time are shown (bottom).

6.1.2 RESULTS OF INDEPENDENT DETECTION ON LARGE MAPS

Instances using DAO maps contained 16 and 32 agents. To obtain various difficulties we varied the distance of agents from their goals. This is again identical to the previous experimental setup with DAO maps. MDD-SAT with versions integrating SID and ID are compared. Results for DAO maps `brc202d`, `den520d`, and `ost003d` are shown in Figures 30, 31, and 32.

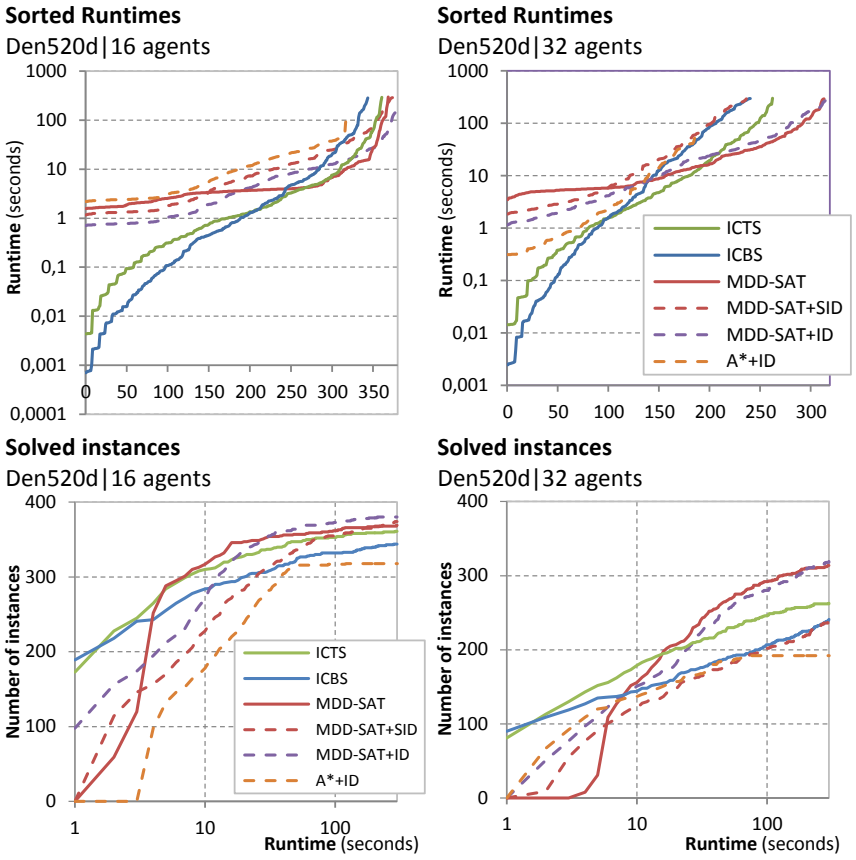


Figure 31: Runtime results with ID and SID for `den520` DAO map. Sorted runtimes (top) and the number of solved instances as a function of time are shown (bottom).

Results offers various patterns and it seems that no easy generalization valid across all types of maps is possible. Results suggest that structure of the map plays an important role. On the `brc202d` map we can observe that SID leads to a better performance only in minority of cases of easy instances where the distance from the goal small. In all other cases, SID represents an overhead and the plain MDD-SAT performs better. ID on the other hand has some observable benefit, we can see that for hard instances with 32 agents MDD-SAT+ID performs as the best.

The explanation of this behavior rests in the structure of the map which consists of long narrow corridors. After having the goal positions far enough from the initial positions, agents inevitably interact through encountering each other in the corridors and are merged into larger groups which pushes the performance of both ID and SID towards and behind the plain MDD-SAT. However the overhead with independence detection has slower growth than the complexity of solving one large group of agents which explains the convergence the performance of MDD-SAT and the versions with SID and ID in harder instances and better performance of MDD-SAT+ID in case of 32 agents.

The `den520d` map features a different structure. It can be regarded as one large open space surrounded by a shaped border. Such structure suggests a hypothesis that when the distance from the goal is small then lot of independence could be detected and both SID and ID should improve

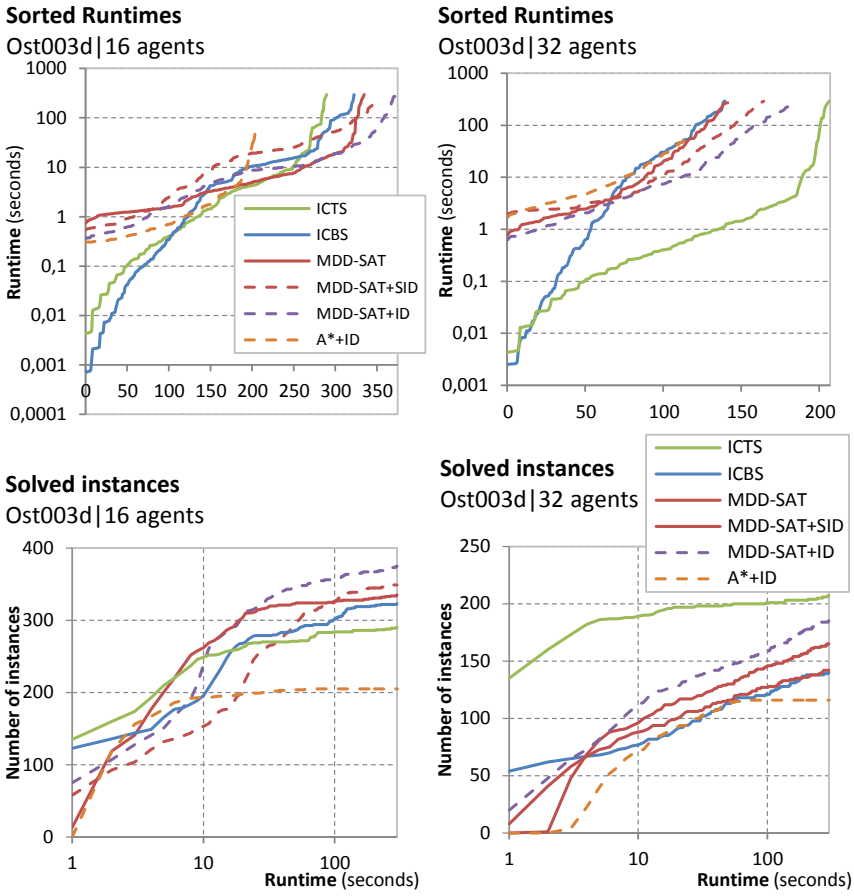


Figure 32: Runtime results with ID and SID for `ost003` DAO map. Sorted runtimes (top) and the number of solved instances as a function of time are shown (bottom).

the plain MDD-SAT. The verification against actual results supports the hypothesis. In addition to this, we can observe that longer distance from the goals causes worse performance of both SID and ID with respect to the plain MDD-SAT. Longer paths often go near the boundary of the open space where agents have an opportunity to meet each other and the group merging often arises from such situation.

The `ost003d` map can be regarded as relatively large open rooms separated by walls with narrows doors. This is again a different structure from both previous maps. Here we should expect that shortest paths need to touch the walls and corners which is again a situation that can give rise to the group merging. Comparing the shape of `den520d` and `ost003d` we can see that it is more likely to touch a wall or a corner in `ost003d` than touching the open space boundary in the `den520d` map. Results supports this hypothesis as we can see that significantly better performance is achieved with SID and ID only in a small portion of instances with the small distance from the goals. We can also observe that ID improves the plain MDD-SAT significantly in hard instances which suggests that we are able to detect non-trivial independences in such cases. Again the overhead caused

by using ID is less time consuming than the extra time needed to solve one large group instead of several smaller groups.

7. Bounded Suboptimal SAT-Based MAPF Solving

The versatility of SAT-based approach for solving MAPF can be further demonstrated by adapting it for suboptimal variants of the problem. Adaptation for suboptimal variants has been successfully done with search-based solvers including ICTS (Aljalaud & Sturtevant, 2013) and CBS (Barer et al., 2014). The motivation behind having a suboptimal solver is that we can trade-off the quality of solutions with respect to a given objective and the runtime. Usually lower quality solutions can be obtained faster [5]. In the bounded case, we have the control of how much do we tolerate the suboptimal solution to differ from the ideal optimal one.

In this section, we show how to convert MDD-SAT to a bounded suboptimal algorithm for the sum-of-costs objective. Converting MDD-SAT to a suboptimal any solution algorithm can be simply done by removing the cardinality constraints from the construction of $\mathcal{F}(\mu_0, \xi_0, \Delta)$. Let $\mathcal{F}'(\mu_0, \Delta)$ denote the resulting formula. Since $\mathcal{F}'(\mu_0, \Delta)$ has all constraints from original $\mathcal{F}(\mu_0, \xi_0, \Delta)$ except the cardinality constraints, then clearly a satisfying assignment to $\mathcal{F}'(\mu_0, \Delta)$ still represents a feasible solution (no collisions between agents etc.). Since $\mathcal{F}'(\mu_0, \Delta)$ is less constrained than $\mathcal{F}(\mu_0, \xi_0, \Delta)$, we expect it to be solved faster. Indeed, we observed this in our preliminary experiments. Using $\mathcal{F}'(\mu_0, \Delta)$ in Algorithm 2 instead of $\mathcal{F}(\mu_0, \xi_0, \Delta)$, however, looses the sum-of-cost optimality.

Hence, replacing $\mathcal{F}(\mu_0, \xi_0, \Delta)$ with $\mathcal{F}'(\mu_0, \Delta)$ in Algorithm 2 leads to a suboptimal version of the MDD-SAT solver that is faster than the optimal version. We refer to this unbounded version of MDD-SAT denoted as **uMDD-SAT**.

A key question is what is the suboptimality of the solutions returned by uMDD-SAT? Is it really unbounded? We show later that even without the cardinality constraints, the suboptimality of the solutions outputted is bounded, due to how $\mathcal{F}(\mu_0, \xi_0, \Delta)$ and $\mathcal{F}'(\mu_0, \Delta)$ are constructed.

Next, we show how to control the suboptimality of the returned solution by introducing a relaxed version of the optimal cardinality constraints, allowing the algorithm's user to balance the runtime and the suboptimality.

7.1 Bounded Suboptimal Version of MDD-SAT

The key to our bounded-suboptimal SAT-based solver is that it splits the Δ parameter used in construction of $\mathcal{F}(\mu_0, \xi_0, \Delta)$ into two parameters Δ_μ and Δ_ξ. The resulting formulae, denoted $\mathcal{F}(\mu_0, \xi_0, \Delta_\mu, \Delta_\xi)$, represents the decision problem "is there a valid solution to the given MAPF problem with a makespan smaller than or equal to $\mu_0 + \Delta_\mu$ and a sum-of-costs smaller than or equal to $\xi_0 + \Delta_\xi$".

One can view MDD-SAT as solving a sequence of such decision problem, setting $\Delta_\mu = \Delta_\xi = \Delta$ and incrementing both Δ_μ and Δ_ξ by one in every iteration. To return bounded-suboptimal solutions, we allow Δ_ξ to be less restrictive; that is, larger than Δ_μ by some integer value $\delta \geq 0$. This produces a formula $\mathcal{F}(\mu_0, \xi_0, \Delta, \Delta + \delta)$ which is approximately the same size as $\mathcal{F}(\mu_0, \xi_0, \Delta)$

5. Ignoring the quality completely could open opportunity to use polynomial time algorithms like PUSH-AND-ROTATE (de Wilde et al., 2014) or BIBOX (Surynek, 2009) where runtime is usually not a problematic factor.

but represents more solutions[6]. Since $\Delta + \delta > \Delta$, we expect a formula with the sum-of-costs bounded by $\Delta + \delta$ to be easier to solve than that with the original Δ.

The following proposition shows that for a solvable MAPF Σ the sum-of-costs of the solution obtained by the above process differs from the optimal one by at most δ.

Proposition 5 *Let δ be a non-negative integer and let $\mathcal{F}(\mu_0, \xi_0, \Delta, \Delta + \delta)$ be the first satisfiable formula found in the sequence of formulae $\mathcal{F}(\mu_0, \xi_0, 0, \delta)$, $\mathcal{F}(\mu_0, \xi_0, 1, 1 + \delta)$,...,$\mathcal{F}(\mu_0, \xi_0, \Delta - 1, \Delta + \delta - 1)$, $\mathcal{F}(\mu_0 + \Delta, \xi_0, \Delta, \Delta + \delta)$. Then the solution represented by $\mathcal{F}(\mu_0, \xi_0, \Delta, \Delta + \delta)$ has sum-of-costs $\xi \leq \xi_{opt} + \delta$ where ξ_{opt} is the optimal sum-of-costs for input MAPF Σ.*

Proof: Since $\mathcal{F}(\mu_0, \xi_0, \Delta, \Delta + \delta)$ is the first satisfiable formula in the sequence, we know that previous formula $\mathcal{F}(\mu_0, \xi_0, \Delta - 1, \Delta + \delta - 1)$ is not solvable. This means that no solution of makespan at most $\mu_0 + \Delta - 1$ and sum-of-costs at most $\xi_0 + \Delta + \delta - 1$ exists. In addition to this, we know that all solutions of sum-of-costs $\xi_0 + \Delta - 1$ fit under the makespan of at most $\mu_0 + \Delta - 1$.

Hence unsolvability of $\mathcal{F}(\mu_0, \xi_0, \Delta - 1, \Delta + \delta - 1)$ together with $\delta \geq 0$ implies that there is no solution of sum-of-costs smaller than or equal to $\xi_0 + \Delta - 1$ (without restricting the makespan) since otherwise $\mathcal{F}(\mu_0, \xi_0, \Delta - 1, \Delta + \delta - 1)$ would be solvable. Therefore, the optimal sum-of-costs is at least $\xi_0 + \Delta$. The solvability of $\mathcal{F}(\mu_0, \xi_0, \Delta, \Delta + \delta)$ tells that there is a solution of Σ of sum-of-costs $\xi_0 + \Delta + \delta$ which differs from the optimum by at most δ. ■

Observe that the only property of δ we used was that it is a non-negative integer but there is no requirement that it must be constant across individual iterations of the algorithm. Proposition 5 holds even if we use a non-negative δ as a function of Δ instead of a constant. This property can be used to modify the above SAT-based framework to a $(1 + \epsilon)$-bounded suboptimal algorithm. That is, the algorithm produces solutions that are no worse than $(1 + \epsilon)$ times than the optimum. How to calculate δ from given ϵ is summarized in the following corollary.

Corollary 1 *Given an error $\epsilon > 0$ the iterative SAT-based suboptimal framework can modified to an $(1 + \epsilon)$-bounded suboptimal algorithm by appropriate setting of δ.*

Proof: As δ is in fact a function of the number of iterations we will use notation $\delta(\Delta)$. Let $\delta(\Delta) = \epsilon \cdot (\xi_0 + \Delta)$. Then the sum-of-costs of the solution returned by the algorithm is at most $(1+\epsilon)\cdot(\xi_0+\Delta)$. Following the arguments from the proof of Proposition 5, the optimal sum-of-costs is at least $\xi_0 + \Delta$ hence the ratio between the sum-of-costs of returned solution and the optimum is at most $(1 + \epsilon)$. ■

The pseudo-code of the $(1 + \epsilon)$-bounded suboptimal version of the MDD-SAT algorithm is presented as Algorithm 6. We refer to this algorithm as **eMDD-SAT**.

Let us note that a further minor improvement of the pseudo-code could be done which uses a basic relation of time expansion used in the formula. Observe that in any solution to an MAPF problem it holds that $\mu \leq \xi \leq k \cdot \mu$ where k is the number of agents. Therefore, if $\xi_0 + \Delta + \delta(\Delta) \geq \mu \cdot k$, then there is no need to add any cardinality constraints to $\mathcal{F}(\mu, \Delta)$ to bound the sum-of-costs, since any solution of the makespan at most μ has automatically bounded its sum-of-costs by $\mu \cdot k$.

6. The MDDs used for construction of $\mathcal{F}(\mu_0, \xi_0, \Delta, \Delta + \delta)$ are identical as those for $\mathcal{F}(\mu_0, \xi_0, \Delta)$ but the former uses different cost bounds in the cardinality constraints whose encoding based on sequential counter has the size proportional to the value of the bound (Sinz, 2005).

Algorithm 6: eMDD-SAT, an $(1+\epsilon)$-bounded suboptimal version of the MDD-SAT MAPF solver

```
1  eSolve-MDD-SAT_SOC(MAPF Σ = (G = (V, E), A, α_0, α_+),ε)
2      if Σ is unsolvable then
3          return UNSOLVABLE
4      end
5      μ_0 = max_{a_i ∈ A} ξ_0(a_i)
6      ξ_0 = Σ_{a_i ∈ A} ξ_0(a_i)
7      Δ ← 0
8      while TRUE do
9          Δ_μ ← Δ
10         δ ← ε · (ξ_0 + Δ)
11         Δ_ξ ← Δ_μ + δ
12         F(μ_0, ξ_0, Δ_μ, Δ_ξ) ← encode-MAPF(Σ, μ_0, ξ_0, Δ_μ, Δ_ξ)
13         π ← consult-SAT-SOLVER(F(μ_0, ξ_0, Δ_μ, Δ_ξ))
14         if π ≠ UNSAT then
15             return π
16         end
17         Δ ← Δ + 1
18     end
19 end
```

This inequality represents a limit of the relaxation achievable by allowing more freedom over the cost bound imposed by the cardinality constraints. Hence the $(1+\epsilon)$-bounded suboptimal MDD-SAT algorithm tends to be near optimal anyway. The algorithm can be at most $\left(\frac{k\cdot(\mu_0+\Delta)}{\xi_0+\Delta}\right)$-bounded.

7.2 Experimental Results for uMDD-SAT and eMDD-SAT

We again evaluated suboptimal variants of MDD-SAT on small grids with obstacles and on large DAO maps. The set of instances for small grids was the same as in the previous experiments. Again we changed number of agents from 1 until we reach unsolvable instances under the given runtime of 300 seconds. For each number of agents 10 instances with random initial and goal configuration were generated.

In DAO maps we varied the number of agents instead of varying the distance from their goals to obtain instances of various difficulties. The number of agents is varied from 1 to 256 where the steps is initially 1 and then increased to 16. Ten instances with random initial and goal configuration per number of agents were used.

7.2.1 THE UNBOUNDED VARIANT: UMDD-SAT

Comparison of unbounded algorithms including SAT-based UNIAGENT (Surynek, 2015), ECBS (Barer et al., 2014), PUSH-AND-SWAP (Luna & Bekris, 2011), and uMDD-SAT are shown in Figures 33, 34, 35, and 36.

All these algorithms represent different approaches to finding unbounded suboptimal solutions. UNIAGENT is an unbounded suboptimal SAT-based method. Unlike MDD-SAT, the time expansion in the UNIAGENT algorithm adds a new layer, a copy of the underlaying graph G, only after agents cannot avoid each other within the existing layers. ECBS is a representative of suboptimal search-based algorithms, a relaxation of CBS. PUSH-AND-SWAP on the other hand views the

MAPF problem via transforming permutations of agents in the graph and is a major representative of rule-based algorithms.

Sorted runtimes of instances on 8×8, 16×16, and 32×32 grids solvable under the time limit are shown in Figure 33 (the lower line means faster algorithm). We can see clear dominance of the rule-based polynomial time PUSH-AND-SWAP algorithm whose runtime grows only moderately in more difficult instances. The opposite can be seen for search-based and compilation-based algorithms. Their runtime grows fast and in some cases UNIAGENT and uMDD-SAT fail to solve all instances. The UNIAGENT algorithm performs well only in harder instances on the 8×8 grid where it outperforms uMDD-SAT and is close to ECBS, but in other cases it is clearly loosing. Comparison of ECBS and uMDD-SAT does not show a clear winner. On easier instances ECBS tends to be faster white in harder cases uMDD-SAT tends to be faster.

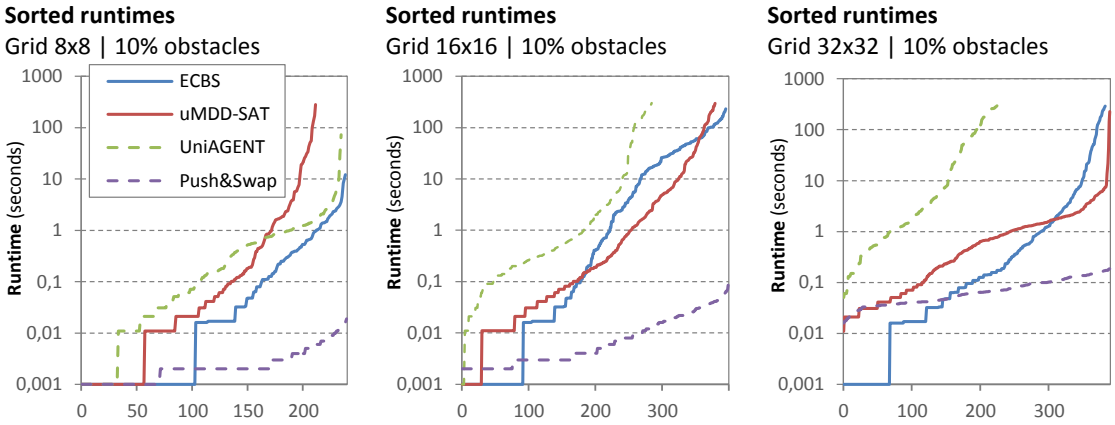


Figure 33: Comparison of unbounded suboptimal solvers - sorted runtimes for uMDD-SAT, PUSH-AND-SWAP, ECBS, and UNIAGENT are compared in 8×8, 16×16, and 32×32 grids.

To understand the performance of individual algorithms better we also present comparison of the sum-of-costs of solutions produced by tested algorithms in Figure 34 (lower line is closer to the optimum). We can see here that although PUSH-AND-SWAP is the fastest algorithm the quality of its solutions it the worst, that is the sum-of-costs is the highest among all tested algorithms. Sum-of-costs in UNIAGENT is also high. Comparison between uMDD-SAT and ECBS indicates that ECBS tends to generate lower sum-of-costs.

The generalization from the observed trends is that even if we completely relax from the quality of solutions in search-based and compilation-based algorithms still their performance cannot match the polynomial-time rule-based methods. Another generalization is that despite the relaxation of the cost bound search-based and compilation-based algorithms tend to produce solutions of higher quality than rule-based algorithm does.

The explanation of the observed trends is that search-based and compilation-based algorithms eventually traverse search space of exponential size. Relatively good performance of ECBS can attributed to its greedy nature after the cost bound is relaxed which more likely leads to guessing a solution. On the other hand uMDD-SAT is slowed down by proving non-existence of solutions for lower makespans.

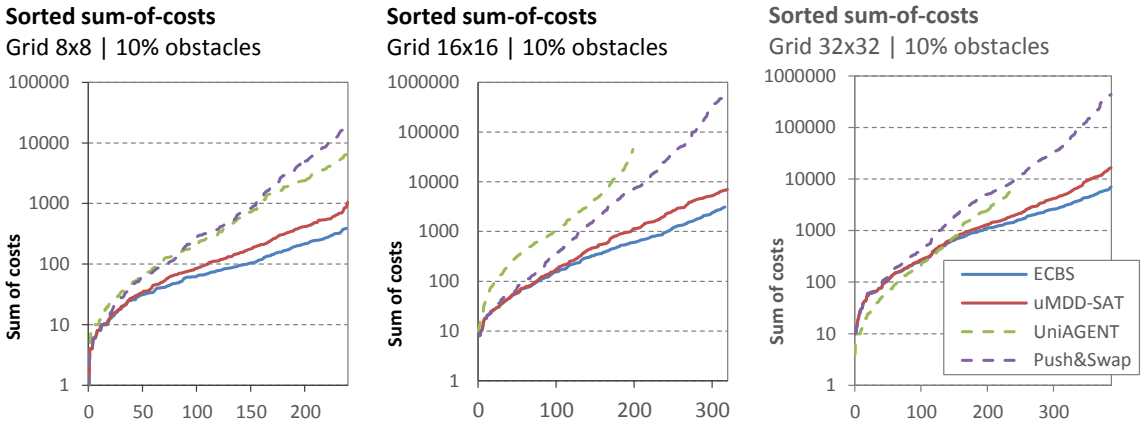


Figure 34: Comparison of solutions produced by unbounded suboptimal solvers - sorted sum-of-costs of solutions by uMDD-SAT, PUSH-AND-SWAP, ECBS, and UNIAGENT are compared in 8×8, 16×16, and 32×32 grids.

Results of the test of unbounded algorithms on large DAO maps is reported in Figures 35 and 36. The former shows sorted runtimes while the latter shows sorted sum-of-costs (lower plot means a better results in both figures). Due to its weak performance on large maps, the UNIAGENT algorithm was omitted in this test.

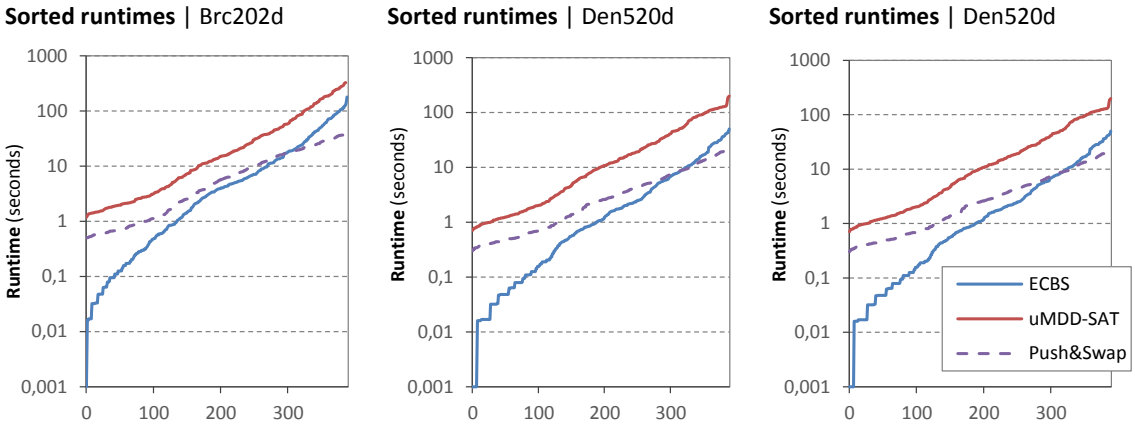


Figure 35: Comparison of unbounded suboptimal solvers on DAO maps - sorted runtimes for uMDD-SAT, ECBS, and PUSH-AND-SWAP.

Results indicate clear dominance of the ECBS algorithm which consistently outperforms the uMDD-SAT algorithm and even PUSH-AND-SWAP except hardest cases containing many agents. There is no significant difference in the sum-of-costs of solutions of uMDD-SAT and ECBS but PUSH-AND-SWAP produces solutions with significantly higher sum-of-costs. These trends can be explained two-fold: (i) uMDD-SAT has relatively big overhead of constructing large MDDs and it

must often prove the non-existence of a solution which is hard; (ii) ECBS has a great freedom to navigate itself greedily towards guessing a correct solution.

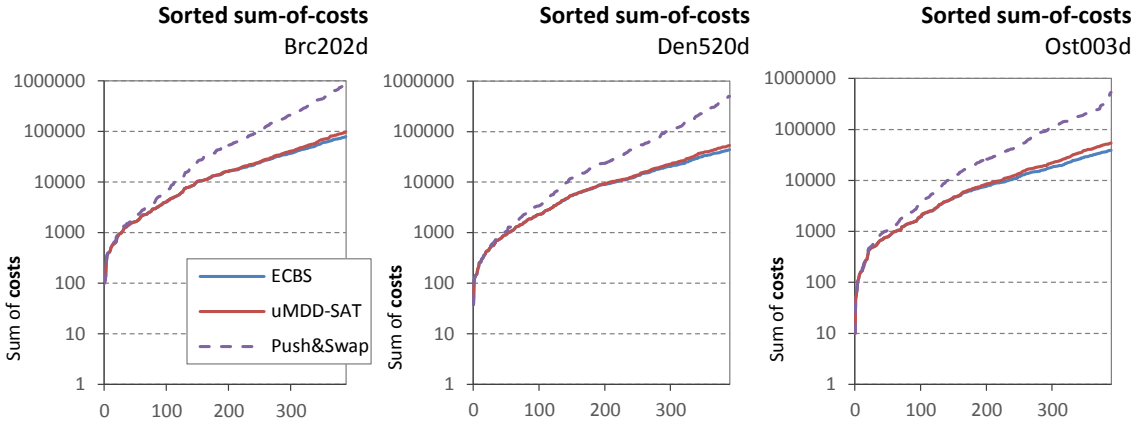


Figure 36: Comparison of unbounded suboptimal solvers on DAO maps - sorted sum-of-costs for uMDD-SAT, ECBS, and PUSH-AND-SWAP.

7.2.2 THE BOUNDED VARIANT: EMDD-SAT

In test of the bounded variants we used $\epsilon = 0.01$ and tested algorithms eMDD-SAT(1.01) and ECBS(1.01). Both algorithms were tested on small grids with obstacles and large DAO maps. Results are presented in Figures 37 and 38 showing sorted runtimes on small grids and large DAO maps respectively.

It can be observed that in small grids eMDD-SAT(1.01) provides better performance than ECBS(1.01). However as the environment gets larger the performance gap between the two algorithms narrows especially in easier instances. The continuation of this trend can be seen in results for DAO maps where ECBS(1.01) dominates except hard instances with many agents.

The explanation for these trends is that once the overhead of constructing MDDs is less significant, eMDD-SAT(1.01) has an advantage over ECBS(1.01). On the other hand if large MDDs are needed to be constructed then eMDD-SAT(1.01) maintains the advantage only for hard cases where ECBS(1.01) cannot find a solution greedily and is forced to perform intensive search while eMDD-SAT(1.01) can use smarter exploration of the search space through learning and Boolean constraint propagation.

Additional experiments confirmed that increasing ϵ generally leads to improving performance of ECBS$(1+\epsilon)$ while eMDD-SAT$(1+\epsilon)$ relatively looses. Ultimately increasing ϵ converges to the unbounded case.

7.2.3 EVALUATION OF THE NUMBER OF TIME EXPANSIONS

When using the unbounded MAPF solver uMDD-SAT, the actual number of time expansions of MDDs made by the algorithm and the resulting makespan are equal. But as shown in Section 5.4.2

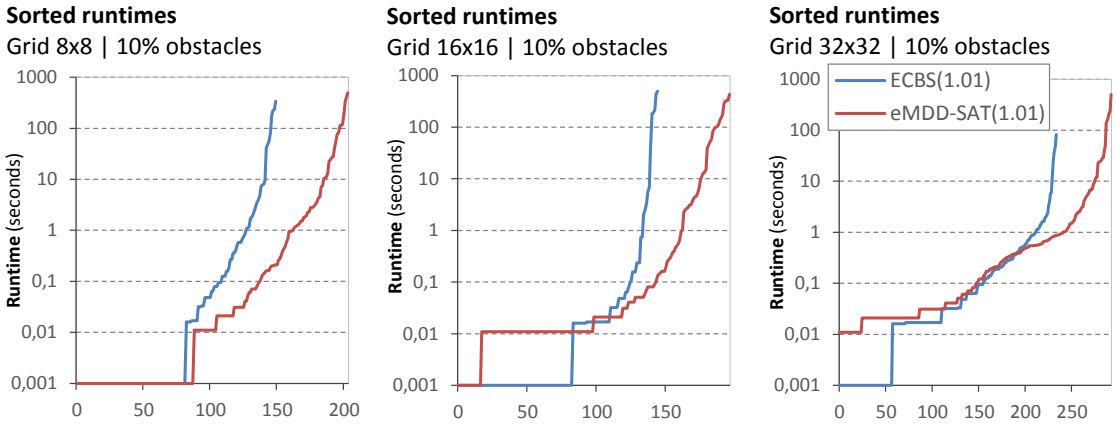


Figure 37: Comparison of bounded suboptimal solvers - sorted runtimes for eMDD-SAT(1.01) and ECBS(1.01) in 8×8, 16×16, and 32×32 grids.

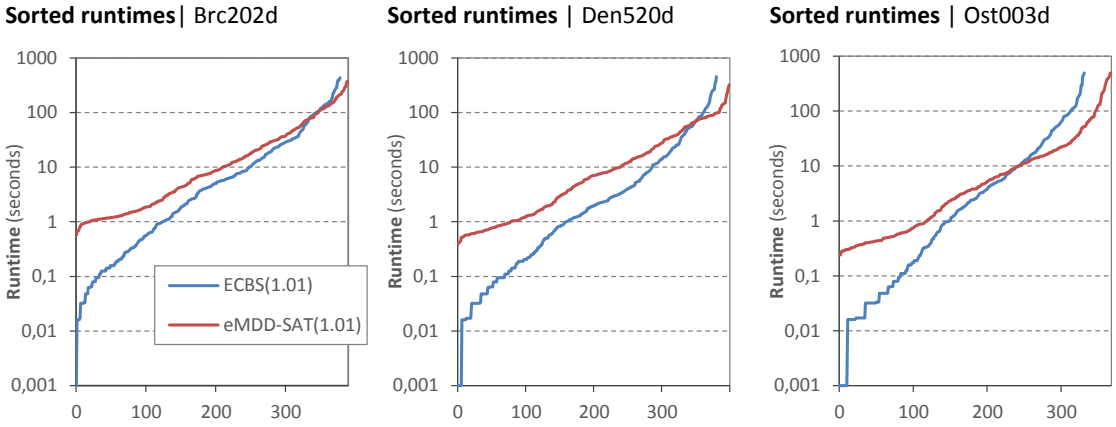


Figure 38: Comparison of bounded suboptimal solvers on DAO maps - sorted runtimes for eMDD-SAT(1.01) and ECBS(1.01) are reported.

there is a significant difference between these values when optimal MDD-SAT is used. Hence the natural question is: What is the trend of the difference between the number of MDD expansions and the makespan for bounded suboptimal variants of MDD-SAT depending on error ϵ?

Results analyzing the difference between the number of expansions and the makespan for grids are shown in Figures 39, 40, and 41. The plots use the same format as in Figures 24 and 25, that is a box plot based on 10 random instances per various number of agents is shown. Results for three values of the ϵ error are shown: $\epsilon = 0.0$ (corresponds to optimal MDD-SAT), $\epsilon = 0.01$, and $\epsilon = 0.05$.

Results for DAO maps using the same format as results for grids are shown in Figures 42, 43, and 44.

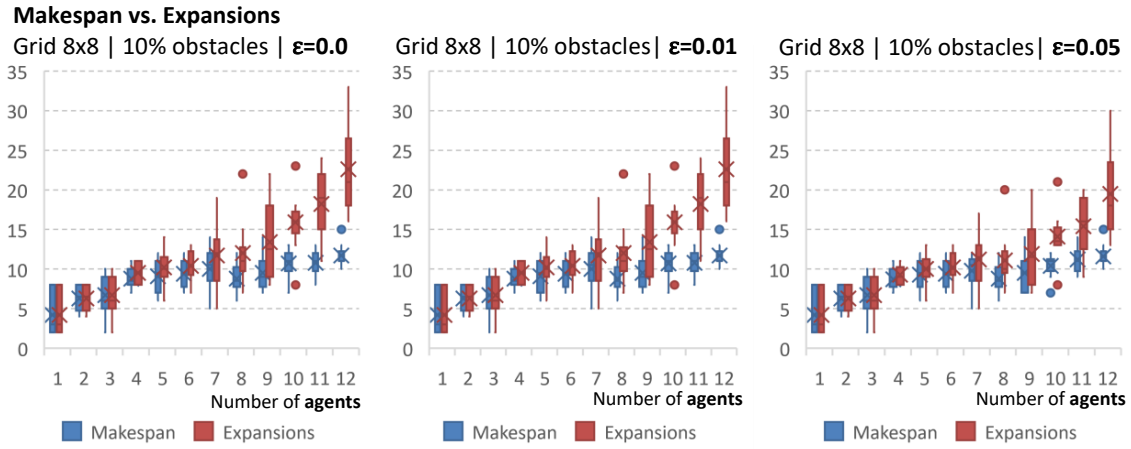


Figure 39: Comparison of computed makespan and the number of time expansions of MDDs in the 8×8 grid for MDD-SAT, eMDD-SAT(1.01), and eMDD-SAT(1.05).

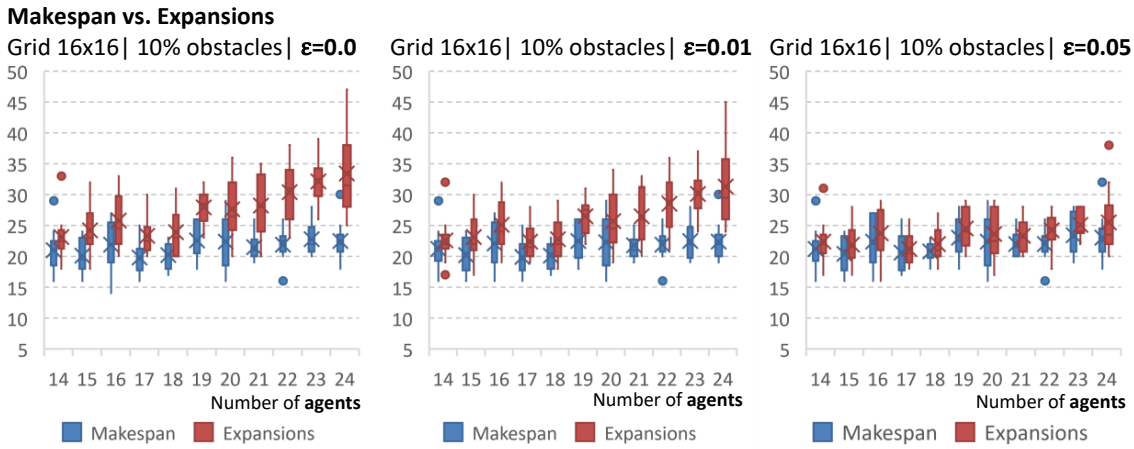


Figure 40: Comparison of computed makespan and the number of time expansions of MDDs in the 16×16 grid for MDD-SAT, eMDD-SAT(1.01), and eMDD-SAT(1.05).

A clearly observable trend in the grids is that the difference between the number of expansions and the resulting makespan quickly decreases even for very small increase of ϵ. Moreover, the larger is the grid the convergence of the difference towards zero is faster. The trend is even stronger in DAO maps where the non-zero difference is rare even for the optimal MDD-SAT and quickly diminishes with increased ϵ in bounded suboptimal eMDD-SAT.

The difference between the number of expansions of MDDs, that equals $\mu_0 + \Delta$, and the makespan of computed solution can be regarded through the distribution of extra cost Δ among agents. When the distribution is unequal, that is, most of the extra cost is consumed by few agents while other agents consume only little in addition to the cost of their shortest paths $\xi_0(a_i)$, then the difference between the number of expansions and the computed makespan is small. On the other

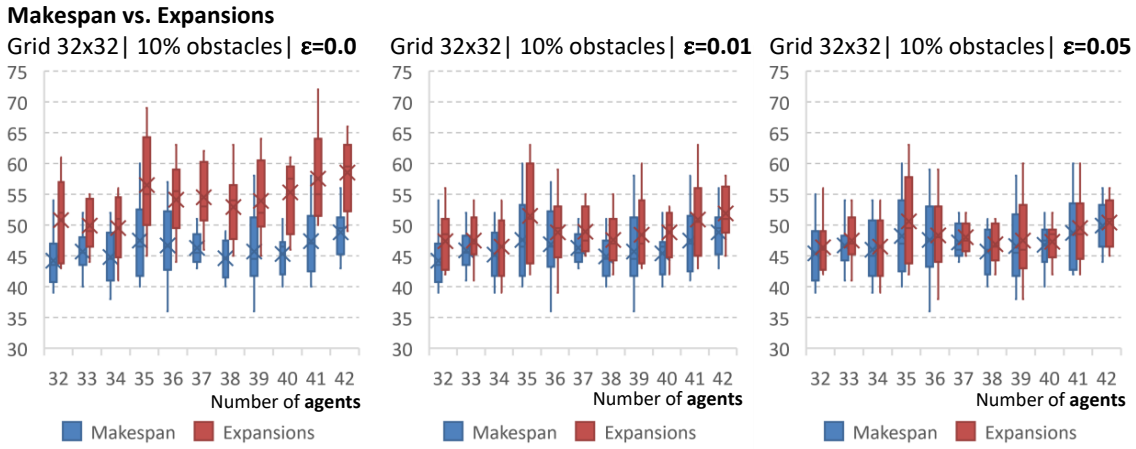


Figure 41: Comparison of computed makespan and the number of time expansions of MDDs in the 32×32 grid for MDD-SAT, eMDD-SAT(1.01), and eMDD-SAT(1.05).

hand, when the distribution of the extra cost is distributed equally among the agents, then the difference is large. This is implied by the design of MDD expansion scheme that counts with the worst case when all extra cost is consumed by a single agent.

The factors behind the large difference between the number of expansions and the makespan is first advancing to large extra cost Δ and second distributing large Δ equally between the agents so that the distribution is far from the worst case. This situation naturally appears on maps densely occupied by agents (small grids) where agents need to diverge far from their shortest individual paths due to intensive avoidance among each other while it is less common on large DAO maps.

Increasing ϵ enables the eMDD-SAT algorithm to find a solution with equally distributed extra cost Δ increased by the ϵ factor while only making $\mu_0 + \Delta$ expansions which reduces the difference.

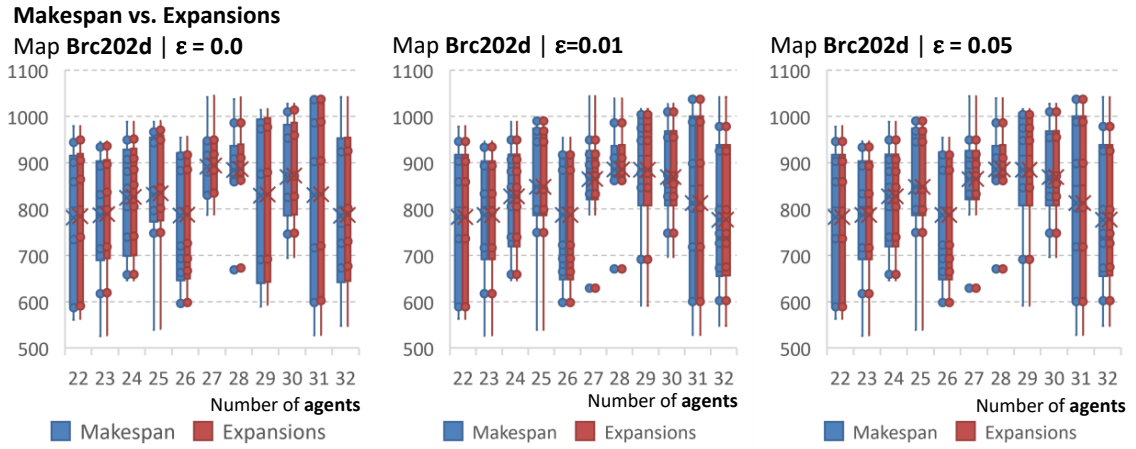


Figure 42: Comparison of computed makespan and the number of time expansions of MDDs in DAO map `brc202d` for MDD-SAT, eMDD-SAT(1.01), and eMDD-SAT(1.05).

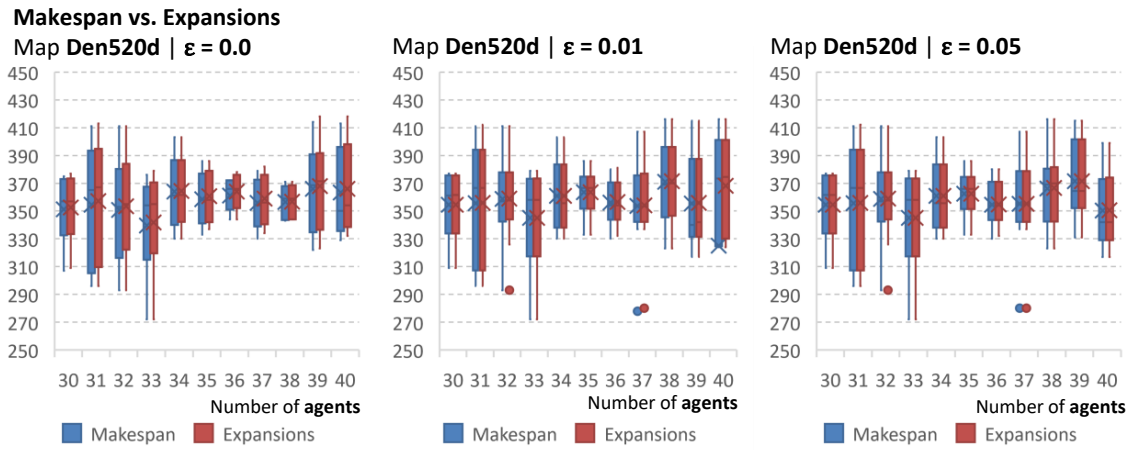


Figure 43: Comparison of computed makespan and the number of time expansions of MDDs in DAO map `den520d` for MDD-SAT, eMDD-SAT(1.01), and eMDD-SAT(1.05).

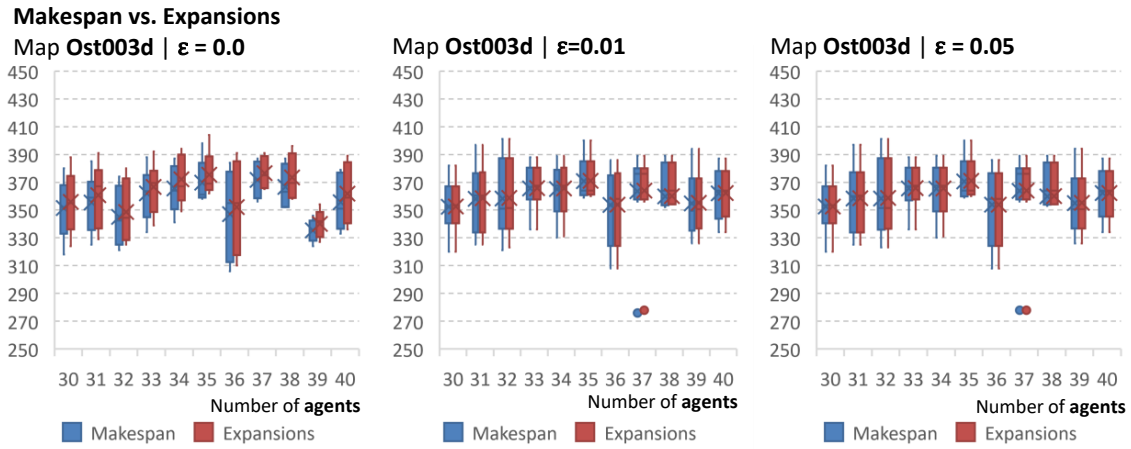


Figure 44: Comparison of computed makespan and the number of time expansions of MDDs in DAO map `ost003d` for MDD-SAT, eMDD-SAT(1.01), and eMDD-SAT(1.05).

8. Discussion

In this paper, we focused on how to migrate techniques from search-based to SAT-based MAPF solvers. This raises the higher-level question of whether the techniques from MAPF research in general can be migrated to and from other forms of multi-agent planning. For example, consider the relation between MAPF and Dec-POMDP (Bernstein, Givan, Immerman, & Zilberstein, 2002; Oliehoek, Witwicki, & Kaelbling, 2012; Witwicki, 2011; Witwicki & Durfee, 2010; Varakantham, Kwak, Taylor, Marecki, Scerri, & Tambe, 2009; Becker, Zilberstein, Lesser, & Goldman, 2004). Briefly, MAPF can be viewed as a special case of Dec-POMDP where the agents are fully observable, all action outcomes are deterministic, and the actions of each agent is just to move to

an adjacent cell. In MAPF, the agents' goal is to reach their corresponding goal positions, while in Dec-POMDP the goal is to maximize a joint reward function. In MAPF, the only interaction between the agents is negative, i.e., agents cannot help each other to achieve their goals, that is, adding agents can only make the problem solving harder. In contrast, in Dec-POMDP agents may assist each other, and having more agent may yield higher reward. In both cases, however, a major speedup can be obtained by decoupling the planning process as much as possible. The ID framework is one such example in MAFP, while all the mentioned references are examples of doing this for Dec-POMDP. Technically, in MAPF transitions are not independent, since agents may collide with each other. The reward function for the sum of costs is independent, as it adds one unit cost per movement of each agent.

In particular, the technique of independence detection (ID) is not limited to MAPF setting. ID in the MAPF setting considers local interactions between agent groups through collisions while collisions can be understood as a form of local interactions that altogether compose a network of interactions.

More general approaches for dealing with networks of interactions have been developed in multi-agent planning (MAP), especially in MAP under uncertainty addressed via partially observable Markov decision processes (POMDPs) (Nair, Varakantham, Tambe, & Yokoo, 2005). Finding optimal policies in POMDPs is a computationally hard problem in general, hence various techniques that simplify the problem via exploiting the structure of the interaction network have been developed. Specifically local independence such as *observation independence* and *transition independence* can be used for more efficient policy generation for POMDPs. Important concept for analyzing dependencies among agents is represented by *coordination graphs* (Guestrin, Koller, & Parr, 2001; Guestrin, Venkataraman, & Koller, 2002) that can be used to decompose the problem into smaller independent parts for which the policy generation is easier.

The assumption in the MAPF setting is that suitable actions for agents are being determined so that collisions are avoided and agents maximize their objective, that is, they head towards their goal positions. In contrast to this, different MAP approaches do not explicitly assume what are the correct actions leading to achieving the objective but instead agents learn suitable actions via reinforcement learning during the planning process (Kok & Vlassis, 2006).

Another important question is how to extend this work in the context of recent research of the interaction of SAT-based and search-based MAPF solvers. Contemporary studies indicate that SAT-based and search-based solvers are complementary in terms of suitability for certain types of MAPF (Kaduri, Boyarski, & Stern, 2020, 2021). That is, there exist types of MAPF instances in which the SAT-based approach dominates the search-based one and vice versa. These studies suggest that a more sophisticated portfolio-based solvers for MAPF could be a promising research direction.

9. Conclusion

The paper summarizes MDD-SAT, the first state-of-the-art SAT-based solver for the sum-of-costs variant of MAPF. We explain in details ideas necessary to reach the resulting enhanced propositional encoding that migrates the ideas from the search-based methods for the use by SAT solvers. MDD-SAT was experimentally compared to state-of-the-art search-based solvers over a variety of domains such as 4-connected grids with random obstacles and large maps from computer games. Our results show that MDD-SAT is often the better option in hard scenarios, while the search-based solvers may perform better in easier cases.

Nevertheless, there is no universal winner and each of the approaches has pros and cons and thus might work best in different circumstances. This calls for a deeper study of various classes of MAPF instances and their characteristics, and how the different algorithms behave across them. Although early studies were made, not too much is known at present to the MAPF community about these aspects.

There are several factors behind the performance of the SAT-based approach: clause learning, constraint propagation, and an efficient implementation of the SAT solver. On the other hand, the SAT solver does not 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.

Versatility of SAT-based approach is further demonstrated through integrating the independence detection technique into the MDD-SAT solver and by introducing the unbounded suboptimal and the bounded suboptimal variants of the solver. In both modifications of the basic solver, we obtained analogous results as in the case when these modifications were carried out for search-based solvers. Independence detection in MDD-SAT helps significantly when there is little interaction between agents. Suboptimal variants of MDD-SAT enables trading the solution quality for the runtime.

For our future work we plan to use the SAT solver more actively during the construction of propositional encoding. Mechanisms that build the encoding and simultaneously check its satisfiability seem to be of great potential. Another direction for future work is to develop efficient encodings for MAPF problems beyond classical MAPF, e.g., where edges have non-uniform costs.

Acknowledgments

This research was supported by the Czech Science Foundation (GAČR), the grant registration number 19-17966S.

References

Aljalaud, F., & Sturtevant, N. R. (2013). Finding bounded suboptimal multi-agent path planning solutions using increasing cost tree search (extended abstract). In Helmert, M., & Röger, G. (Eds.), *Proceedings of the Sixth Annual Symposium on Combinatorial Search, (SoCS 2013)*. AAAI Press.

Andersen, H. R., Hadzic, T., Hooker, J. N., & Tiedemann, P. (2007). A constraint store based on multivalued decision diagrams. In *Principles and Practice of Constraint Programming - CP 2007, Proceedings*, Vol. 4741 of *Lecture Notes in Computer Science*, pp. 118–132. Springer.

Ansótegui, C., Bofill, M., Coll, J., Dang, N., Esteban, J. L., Miguel, I., Nightingale, P., Salamon, A. Z., Suy, J., & Villaret, M. (2019). Automatic detection of at-most-one and exactly-one relations for improved SAT encodings of pseudo-boolean constraints. In Schiex, T., & de Givry, S. (Eds.), *Principles and Practice of Constraint Programming - 25th International Conference, CP 2019*, Vol. 11802 of *Lecture Notes in Computer Science*, pp. 20–36. Springer.

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

Audemard, G., & Simon, L. (2009). Predicting learnt clauses quality in modern SAT solvers. In Boutilier, C. (Ed.), *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pp. 399–404.

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

Balyo, T., Heule, M. J. H., & Järvisalo, M. (2017). SAT competition 2016: Recent developments. In Singh, S. P., & Markovitch, S. (Eds.), *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pp. 5061–5063. AAAI Press.

Barer, M., Sharon, G., Stern, R., & Felner, A. (2014). Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, pp. 961–962.

Beck, Z. (2016). *Collaborative search and rescue by autonomous robots*. Ph.D. thesis, University of Southampton.

Becker, R., Zilberstein, S., Lesser, V. R., & Goldman, C. V. (2004). Solving transition independent decentralized markov decision processes. *J. Artif. Intell. Res.*, *22*, 423–455.

Bernstein, D. S., Givan, R., Immerman, N., & Zilberstein, S. (2002). The complexity of decentralized control of markov decision processes. *Math. Oper. Res.*, *27*(4), 819–840.

Biere, A., Biere, A., Heule, M., van Maaren, H., & Walsh, T. (2009). *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press.

Biere, A., & Brummayer, R. (2008). Consistency checking of all different constraints over bit-vectors within a SAT solver. In *Formal Methods in Computer-Aided Design, FMCAD 2008, Portland, Oregon, USA, 17-20 November 2008*, pp. 1–4.

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

Chai, R., & Su, J. (2013). Motion planning for multi-robot coordination. *IFAC Proceedings Volumes*, *46*(13), 129–134. 13th IFAC Symposium on Large Scale Complex Systems: Theory and Applications.

Claes, D., Oliehoek, F. A., Baier, H., & Tuyls, K. (2017). Decentralised online planning for multi-robot warehouse commissioning. In Larson, K., Winikoff, M., Das, S., & Durfee, E. H. (Eds.), *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017*, pp. 492–500. ACM.

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

de Weerdt, M., & Clement, B. (2009). Introduction to planning in multiagent systems. *Multiagent Grid Syst.*, *5*(4), 345–355.

de Wilde, B., ter Mors, A., & Witteveen, C. (2014). Push and rotate: a complete multi-agent pathfinding algorithm. *J. Artif. Intell. Res. (JAIR)*, *51*, 443–492.

de Wilde, B., ter Mors, A. W., & Witteveen, C. (2013). Push and rotate: cooperative multi-agent path planning. In *AAMAS*, pp. 87–94.

Dimopoulos, Y., Hashmi, M. A., & Moraitis, P. (2012). μ-satplan: Multi-agent planning as satisfiability. *Knowl. Based Syst.*, *29*, 54–62.

Dimopoulos, Y., & Moraitis, P. (2006). Multi-agent coordination and cooperation through classical planning. In *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, Hong Kong, China, 18-22 December 2006*, pp. 398–402. IEEE Computer Society.

Dowling, W. F., & Gallier, J. H. (1984). Linear-time algorithms for testing the satisfiability of propositional horn formulae. *J. Log. Program.*, *1*(3), 267–284.

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

Ephrati, E., & Rosenschein, J. S. (1994). Divide and conquer in multi-agent planning. In *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1*, pp. 375–380. AAAI Press / The MIT Press.

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

Fazekas, K., Biere, A., & Scholl, C. (2019). Incremental inprocessing in sat solving. In *International Conference on Theory and Applications of Satisfiability Testing*, pp. 136–154.

Ferner, C., Wagner, G., & Choset, H. (2013). ODrM* optimal multirobot path planning in low dimensional search spaces. In *ICRA*, pp. 3854–3859.

Gebser, M., Kaufmann, B., Neumann, A., & Schaub, T. (2007). clasp : A conflict-driven answer set solver. In *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15-17, 2007, Proceedings*, pp. 260–265.

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

Goldenberg, M., Felner, A., Sturtevant, N. R., Holte, R. C., & Schaeffer, J. (2013). Optimal-generation variants of EPEA. In *SoCS*.

Guestrin, C., Koller, D., & Parr, R. (2001). Multiagent planning with factored mdps. In Dietterich, T. G., Becker, S., & Ghahramani, Z. (Eds.), *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pp. 1523–1530. MIT Press.

Guestrin, C., Venkataraman, S., & Koller, D. (2002). Context-specific multiagent coordination and planning with factored mdps. In Dechter, R., Kearns, M. J., & Sutton, R. S. (Eds.), *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada*, pp. 253–259. AAAI Press / The MIT Press.

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

Jones, E. G., Dias, M. B., & Stentz, A. (2011). Time-extended multi-robot coordination for domains with intra-path constraints. *Auton. Robots*, *30*(1), 41–56.

Jorgensen, S., Chen, R. H., Milam, M. B., & Pavone, M. (2018). The team surviving orienteers problem: routing teams of robots in uncertain environments with survival constraints. *Auton. Robots*, *42*(4), 927–952.

Kaduri, O., Boyarski, E., & Stern, R. (2020). Algorithm selection for optimal multi-agent pathfinding. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 161–165.

Kaduri, O., Boyarski, E., & Stern, R. (2021). Experimental evaluation of classical multi agent path finding algorithms. In *International Symposium on Combinatorial Search (SoCS)*, Vol. 12, pp. 126–130.

Kautz, H. A. (2006). Deconstructing planning as satisfiability. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference (AAAI 2006), 2006*, pp. 1524–1526. AAAI Press.

Kautz, H. A., McAllester, D. A., & Selman, B. (1996). Encoding plans in propositional logic. In Aiello, L. C., Doyle, J., & Shapiro, S. C. (Eds.), *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96)*, pp. 374–384. Morgan Kaufmann.

Kautz, H. A., & Selman, B. (1992). Planning as satisfiability. In Neumann, B. (Ed.), *10th European Conference on Artificial Intelligence, ECAI 1992. Proceedings*, pp. 359–363. John Wiley and Sons.

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

Kok, J. R., & Vlassis, N. A. (2006). Collaborative multiagent reinforcement learning by payoff propagation. *J. Mach. Learn. Res.*, *7*, 1789–1828.

Kornhauser, D., Miller, G. L., & Spirakis, P. G. (1984). Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *25th Annual Symposium on Foundations of Computer Science, West Palm Beach, Florida, USA, 24-26 October 1984*, pp. 241–250.

Kumar, V., & Michael, N. (2012). Opportunities and challenges with autonomous micro aerial vehicles. *Int. J. Robotics Res.*, *31*(11), 1279–1291.

Liang, J. H. (2018). *Machine Learning for SAT Solvers*. Ph.D. thesis, University of Waterloo.

Liang, J. H., Oh, C., Mathew, M., Thomas, C., Li, C., & Ganesh, V. (2018). Machine learning-based restart policy for CDCL SAT solvers. In Beyersdorff, O., & Wintersteiger, C. M. (Eds.), *Proceedings of the Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018*, Vol. 10929 of *Lecture Notes in Computer Science*, pp. 94–110. Springer.

Luna, R., & Bekris, K. E. (2011). An efficient and complete approach for cooperative path-finding. In *AAAI*.

McCarthy, J., & Hayes, P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., & Michie, D. (Eds.), *Machine Intelligence 4*, pp. 463–502. Edinburgh University Press. reprinted in McC90.

Nair, R., Varakantham, P., Tambe, M., & Yokoo, M. (2005). Networked distributed pomdps: A synthesis of distributed constraint optimization and pomdps. In Veloso, M. M., & Kambhampati,

S. (Eds.), *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pp. 133–139. AAAI Press / The MIT Press.

Oh, K., Park, M., & Ahn, H. (2015). A survey of multi-agent formation control. *Autom.*, *53*, 424–440.

Oliehoek, F. A., Witwicki, S. J., & Kaelbling, L. P. (2012). Influence-based abstraction for multi-agent systems. In Hoffmann, J., & Selman, B. (Eds.), *AAAI Conference on Artificial Intelligence*.

Parker, L. E. (1994). *Heterogeneous Multi-Robot Cooperation*. Ph.D. thesis, Massachusetts Institute of Technology.

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

Ratner, D., & Warmuth, M. K. (1986). Finding a shortest solution for the nxn extension of the 15-puzzle is intractable. In *National Conference on Artificial Intelligence*, pp. 168–172.

Ratner, D., & Warmuth, M. K. (1990). Nxn puzzle and related relocation problem. *J. Symb. Comput.*, *10*(2), 111–138.

Régin, J. (1994). A filtering algorithm for constraints of difference in csps. In *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1.*, pp. 362–367.

Rintanen, J. (2012). Engineering efficient planners with SAT. In Raedt, L. D., Bessiere, C., Dubois, D., Doherty, P., Frasconi, P., Heintz, F., & Lucas, P. J. F. (Eds.), *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track*, Vol. 242 of *Frontiers in Artificial Intelligence and Applications*, pp. 684–689. IOS Press.

Robu, V., Noot, H., Poutré, H. L., & van Schijndel, W. (2011). A multi-agent platform for auction-based allocation of loads in transportation logistics. *Expert Syst. Appl.*, *38*(4), 3483–3491.

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

Rosenschein, J. S. (1995). Multiagent planning as a social process: Voting, privacy, and manipulation. In *Proceedings of the First International Conference on Multiagent Systems, June 12-14, 1995, San Francisco, California, USA*, pp. 431–431. The MIT Press.

Ryan, M. (2008). Exploiting subgraph structure in multi-robot path planning. *JAIR*, *31*, 497–542.

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

Sapena, O., Onaindia, E., & Torreño, A. (2010). On the use of argumentation in multi-agent planning. In *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, Vol. 215 of *Frontiers in Artificial Intelligence and Applications*, pp. 1001–1002. IOS Press.

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

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

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

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

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

Sørli, J., Graven, O. H., & Bjerknes, J. D. (2017). Multi-uav cooperative path planning for sensor placement using cooperative coevolving genetic strategy. In *Advances in Swarm Intelligence - 8th International Conference, ICSI 2017, Fukuoka, Japan, July 27 - August 1, 2017, Proceedings, Part II*, Vol. 10386 of *Lecture Notes in Computer Science*, pp. 433–444. Springer.

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

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

Standley, T. S. (2010b). Finding optimal solutions to cooperative pathfinding problems. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010*. AAAI Press.

Standley, T. S., & Korf, R. E. (2011). Complete algorithms for cooperative pathfinding problems. In Walsh, T. (Ed.), *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, 2011*, pp. 668–673. IJCAI/AAAI.

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. (2014). A simple approach to solving cooperative path-finding as propositional satisfiability works well. In *PRICAI*, pp. 827–833.

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

Surynek, P. (2009). A novel approach to path planning for multiple robots in bi-connected graphs. In *2009 IEEE International Conference on Robotics and Automation, ICRA 2009, Kobe, Japan, May 12-17, 2009*, pp. 3613–3619.

Surynek, P. (2012a). An alternative eager encoding of the all-different constraint over bit-vectors. In *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012*, pp. 927–928.

Surynek, P. (2012b). On propositional encodings of cooperative path-finding. In *IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI 2012, Athens, Greece, November 7-9, 2012*, pp. 524–531.

Surynek, P. (2012c). Towards optimal cooperative path planning in hard setups through satisfiability solving. In *PRICAI 2012: Trends in Artificial Intelligence - 12th Pacific Rim International*

Conference on Artificial Intelligence, 2012. Proceedings, Vol. 7458 of *Lecture Notes in Computer Science*, pp. 564–576. Springer.

Surynek, P. (2012d). Towards optimal cooperative path planning in hard setups through satisfiability solving. In *PRICAI 2012: Trends in Artificial Intelligence - 12th Pacific Rim International Conference on Artificial Intelligence*, pp. 564–576. Springer.

Surynek, P. (2013a). Mutex reasoning in cooperative path finding modeled as propositional satisfiability. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan, November 3-7, 2013*, pp. 4326–4331.

Surynek, P. (2013b). Optimal cooperative path-finding with generalized goals in difficult cases. In *Proceedings of the Tenth Symposium on Abstraction, Reformulation, and Approximation, SARA 2013, 11-12 July 2013, Leavenworth, Washington, USA*.

Surynek, P. (2014a). Compact representations of cooperative path-finding as SAT based on matchings in bipartite graphs. In *26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014, Limassol, Cyprus, November 10-12, 2014*, pp. 875–882.

Surynek, P. (2014b). Simple direct propositional encoding of cooperative path finding simplified yet more. In *Nature-Inspired Computation and Machine Learning - 13th Mexican International Conference on Artificial Intelligence, MICAI 2014, Tuxtla Gutiérrez, Mexico, November 16-22, 2014. Proceedings, Part II*, pp. 410–425.

Surynek, P. (2014c). Solving abstract cooperative path-finding in densely populated environments. *Computational Intelligence*, *30*(2), 402–450.

Surynek, P. (2015). On the complexity of optimal parallel cooperative path-finding. *Fundam. Inform.*, *137*(4), 517–548.

Surynek, P., Felner, A., Stern, R., & Boyarski, E. (2016a). Efficient SAT approach to multi-agent path finding under the sum of costs objective. In *European Conference on Artificial Intelligence (ECAI)*, Vol. 285, pp. 810–818.

Surynek, P., Felner, A., Stern, R., & Boyarski, E. (2016b). An empirical comparison of the hardness of multi-agent path finding under the makespan and the sum of costs objectives. In *Proceedings of the Ninth Annual Symposium on Combinatorial Search, SOCS 2016.*, pp. 145–147.

Surynek, P., Felner, A., Stern, R., & Boyarski, E. (2017a). Modifying optimal sat-based approach to multi-agent path-finding problem to suboptimal variants. In Fukunaga, A., & Kishimoto, A. (Eds.), *Symposium on Combinatorial Search (SOCS)*, pp. 169–170.

Surynek, P., Svancara, J., Felner, A., & Boyarski, E. (2017b). Integration of independence detection into sat-based optimal multi-agent path finding - A novel sat-based optimal MAPF solver. In van den Herik, H. J., Rocha, A. P., & Filipe, J. (Eds.), *Proceedings of the 9th International Conference on Agents and Artificial Intelligence, ICAART 2017, Volume 2*, pp. 85–95. SciTePress.

Surynek, P., Svancara, J., Felner, A., & Boyarski, E. (2017c). Variants of independence detection in sat-based optimal multi-agent path finding. In van den Herik, H. J., Rocha, A. P., & Filipe, J. (Eds.), *Agents and Artificial Intelligence - 9th International Conference, ICAART 2017, Revised Selected Papers*, Vol. 10839 of *Lecture Notes in Computer Science*, pp. 116–136. Springer.

Tack, G. (2009). *Constraint Propagation - Models, Techniques, Implementation*. phdthesis, Saarland University, Germany.

Tamura, N., Taga, A., Kitagawa, S., & Banbara, M. (2009). Compiling finite linear CSP into SAT. *Constraints An Int. J.*, *14*(2), 254–272.

Torreño, A., Onaindia, E., Komenda, A., & Stolba, M. (2018). Cooperative multi-agent planning: A survey. *ACM Comput. Surv.*, *50*(6), 84:1–84:32.

Tozicka, J., Jakubuv, J., Komenda, A., & Pechoucek, M. (2016). Privacy-concerned multiagent planning. *Knowl. Inf. Syst.*, *48*(3), 581–618.

Tsang, K. F. E., Ni, Y., Wong, C. F. R., & Shi, L. (2018). A novel warehouse multi-robot automation system with semi-complete and computationally efficient path planning and adaptive genetic task allocation algorithms. In *15th International Conference on Control, Automation, Robotics and Vision, ICARCV 2018*, pp. 1671–1676. IEEE.

Varakantham, P., Kwak, J., Taylor, M. E., Marecki, J., Scerri, P., & Tambe, M. (2009). Exploiting coordination locales in distributed pomdps via social model shaping. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*. AAAI.

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

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

Witwicki, S. (2011). *Abstracting Influences for Efficient Multiagent Coordination Under Uncertainty.* Ph.D. thesis, University of Michigan.

Witwicki, S. J., & Durfee, E. H. (2010). Influence-based policy abstraction for weakly-coupled dec-pomdps. In Brafman, R. I., Geffner, H., Hoffmann, J., & Kautz, H. A. (Eds.), *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, pp. 185–192. AAAI.

Yu, J., & LaValle, S. (2013). Planning optimal paths for multiple robots on graphs. In *ICRA*, pp. 3612–3617.

Yu, J. (2016). Intractability of optimal multirobot path planning on planar graphs. *IEEE Robotics and Automation Letters*, *1*(1), 33–40.

Yu, J., & LaValle, S. M. (2013a). Planning optimal paths for multiple robots on graphs. In *2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 6-10, 2013*, pp. 3612–3617.

Yu, J., & LaValle, S. M. (2013b). Structure and intractability of optimal multi-robot path planning on graphs. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*.