Modifying Optimal SAT-based Approach to Multi-agent Path-finding Problem to Suboptimal Variants

Pavel Surynek
National Institute of Advanced Industrial Science and Technology (AIST)
Tokyo 135-0064, Japan
pavel.surynek@aist.go.jp

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

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

Abstract

In *multi-agent path finding* (MAPF) the task is to find non-conflicting paths for multiple agents. In this paper we focus on finding suboptimal solutions for MAPF for the *sum-of-costs* variant. Recently, a SAT-based approached was developed to solve this problem and proved beneficial in many cases when compared to other search-based solvers. In this paper, we present SAT-based unbounded- and bounded-suboptimal algorithms and compare them to relevant algorithms. Experimental results show that in many case the SAT-based solver significantly outperforms the search-based solvers.

1 Introduction

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

MAPF has practical applications in video games, traffic control, robotics etc. (see [Sharon *et al.*, 2015] for a survey). The scope of this paper is limited to the setting of *fully cooperative* agents that are centrally controlled. MAPF is usually solved aiming to minimize one of the two commonly-used global cumulative cost functions: **(1) Sum-of-costs** (denoted ξ) is the summation, over all agents, of the number of time steps required to reach the goal location [Dresner and Stone, 2008; Standley, 2010; Sharon *et al.*, 2013; 2015]. Formally, $\xi = \sum_{i=1}^{k} \xi(a_i)$, where $\xi(a_i)$ is an *individual path cost* of agent a_i. **(2) Makespan:** (denoted μ) is the time until the last agent reaches its destination (i.e., the maximum of the individual costs) [Surynek, 2010; 2014; 2015].

Optimal solvers for MAPF can be divided to two classes. **(1) Search-based solvers.** These algorithms consider MAPF as a graph search problem. Some of these algorithm are variants of the A* algorithm that search in a global *search space* – all different ways to place k agents into V vertices, one agent per vertex [Standley, 2010; Wagner and Choset, 2015]. Others algorithms such as ICTS [Sharon *et al.*, 2013] and CBS [Sharon *et al.*, 2015; Boyarski *et al.*, 2015] search different search spaces and employ novel (non-A*) search tree. All these search-based solvers were originally designed for the *sum-of-costs* MAPF variant. But, with simple modifications, they can be modified to work for the makespan variant. **(2) Reduction-based solvers**. By contrast, many recent optimal solvers reduce MAPF to known problems such as CSP [Ryan, 2010], SAT [Surynek, 2012], Inductive Logic Programming [Yu and LaValle, 2013a] and Answer Set Programming [Erdem *et al.*, 2013]. While most reduction-based solvers address the makespan variant, an optimal reduction-based solver for the sum-of-costs variant was recently introduced [Surynek *et al.*, 2016b]. In this paper we further widen this direction and introduce **SAT-based suboptimal solvers**.

Finding optimal solutions for both variants is NP-Hard [Yu and LaValle, 2013b; Surynek, 2010]; as the state-space grows exponentially with k (# of agents). Therefore, many suboptimal solvers were developed. Some suboptimal solvers aim to to quickly find paths for all agents while paying no attention to the quality of the solution, i.e., how far it is from the optimal solution. We refer to such algorithms as **any solution** MAPF solvers. Many any solution MAPF solvers were proposed [Ryan, 2010; Cohen *et al.*, 2015; Silver, 2005; Botea and Surynek, 2015; Sajid *et al.*, 2012], and there is even a polynomial time any solution MAPF solver. These algorithms are usually used when k is large and some of them are not complete.

In some cases, the user might ask for some guarantee on the quality of the solution returned. A common type of such a requirement is that the solution found is **bounded suboptimal**, that its cost is $\leq (1 + \epsilon) \times c_{opt}$ where c_{opt} is the cost of the optimal solution and ϵ is a parameter that sets the desired amount of suboptimality - sometimes called the *error*. A solver that returns bounded-suboptimal solutions is referred to as a **bounded-suboptimal** algorithm or more specifically $(1 + \epsilon)$-bounded suboptimal.

Despite the large number of papers devoted to optimal or to suboptimal solutions, we are only aware of two approaches that provided bounded suboptimal solutions ECBS [Barer *et*

al., 2014] and CBS with highways [Cohen *et al.*, 2015], both are modifications of the *conflict based search* (CBS) algorithm. In this paper we introduce two new SAT-based solvers: uMDD-SAT, an any solution MAPF solver, and eMDD-SAT, a bounded-suboptimal MAPF solver. We experimentally compare our new SAT solvers with relevant any solution or bounded-suboptimal algorithms and show that our SAT solvers is comparable and sometimes outperform other algorithms in many circumstances.

2 Background: Optimal SAT-based Solver

Our suboptimal algorithms presented in this paper are based on a SAT-based optimal MAPF (called MDD-SAT) algorithm for the sum-of-costs variant which was [Surynek *et al.*, 2016a]. The main idea in MDD-SAT is to convert the optimization problem (finding minimal sum-of-costs) to a sequence of decision problems – is there a solution of a given sum-of-costs ξ. A formula $\mathcal{F}_\xi$ has been introduced such that $\mathcal{F}_\xi$ is satisfiable if and only if there is a solution of sum-of-costs ξ. We now provide sufficient details about $\mathcal{F}_\xi$ that are needed for the rest of this paper. More information on this formula and its exact variables can be found in [Surynek *et al.*, 2016a].

Let $\xi_0(a_i)$ be the cost of the shortest individual path for agent a_i (ignoring collisions with the other agents), 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]. It is a known admissible heuristic for optimal sum-of-costs search algorithms, since it is a lower bound on the minimal sum-of-costs. ξ_0 is calculated by relaxing the problem by omitting the other agents, solving k single-agent shortest path problem. 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. $\mathcal{F}_\xi$ is built on top of the following understanding about the maximal makespan (μ) of solutions with sum-of-costs ξ. Let $\Delta = \xi - \xi_0$.

Proposition 1 *If a solution with sum-of-costs ξ exists then its makespan is at most $\mu \leq \mu_0 + \Delta$.*

Proof outline [Surynek *et al.*, 2016a] Clearly, if there is a solution of cost ξ_0 then its makespan will be no greater than μ_0. But, we want a solution of cost ξ, which is ξ_0 plus some Δ. In the worst-case all the Δ extra moves belong to the agent with the largest shortest-path. Thus, the resulting path of that agent would be $\mu_0 + \Delta$, as required. ■

Based on Proposition 1, $\mathcal{F}_\xi$ is constructed by generating a *time expansion graph* [Surynek *et al.*, 2016a] (denoted TEG) of $\mu_0 + \Delta$ layers. A TEG is a directed acyclic graph (DAG) in which the set of vertices of the underlaying graph G are duplicated for all time-steps from 0 up to the desired number of layers ($\mu = \mu_0 + \Delta$). Possible actions (move along edges or wait) are represented as directed edges between successive time steps. Formally a TEG with μ layers is defined as follows:

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

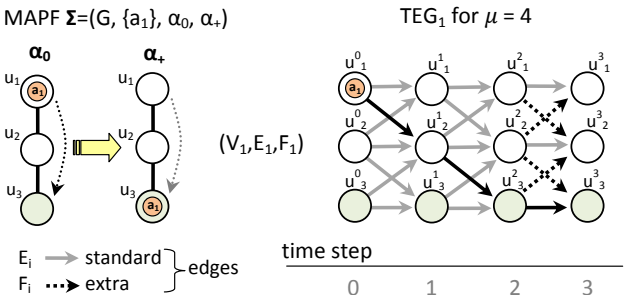

Figure 1: A TEG for an agent that needs to go from u_1 to u_3.

Algorithm 1: Sum-of-costs optimal SAT-based solving

```
1  MDD-SAT(MAPF Σ = (G = (V, E), A, α0, α+))
2      μ0 = max_{ai∈A} ξ0(ai)
3      Δ ← 0
4      while Solution not found do
5          μ = μ0 + Δ
6          ξ = ξ0 + Δ
7          Fξ ← Build-Formula(Σ, μ, Δ)
8          Solution ← Run-SAT-Solver(Fξ)
9          if Solution not found then Δ ← Δ + 1 ;
10     end
11     return Solution
12 end
```

Figure 1 illustrates a TEG with 4 layers. $\mathcal{F}_\xi$ has a a propositional variable for every pair of agent and edge in the TEG. Setting this variable to TRUE represents that the edge is traversed by that agent. Thus, an assignment to these variables represents a solution the the MAPF problem. Appropriate constraints are added to $\mathcal{F}_\xi$ to make sure the solution is valid.

To verify that a solution to $\mathcal{F}_\xi$ represents a solution with sum of costs lower than ξ, we add a *cardinality constraint* over these agent-edge variables. Cardinality constraint is a constraint that allows counting variables set to true in a formula and in general bound a numeric cost. The SAT literature offers several techniques for encoding a *cardinality constraint* [Bailleux and Boufkhad, 2003; Silva and Lynce, 2007]. Formally, for a bound $\lambda \in \mathbb{N}$ and a set of propositional variables $X = \{x_1, x_2, ..., x_k\}$ the *cardinality constraint* $\leq_\lambda \{x_1, x_2, ..., x_k\}$ is satisfied iff the number of variables from the set X that are set to TRUE is $\leq \lambda$. Actually we use the cardinality constraint to bound the number of edges each agent traverses *in addition* to the first $\xi_0(a_i)$ edges. So, that the cardinality constraint in fact ensures that the number of such extra moves is at most Δ. This is done by modifying the TEG, marking some edges are standard and others as extra (see Figure 1). We do so for efficiency reasons, following Surynek et al. (2016a).

Algorithm 1 summarizes the MDD-SAT algorithm. Δ is initialized as zero and in every iteration it is increased (Line 9). μ is set to $\mu_0 + \Delta$ (Line 5) and ξ to $x_0 + \Delta$ (Line 6). Next the formula $\mathcal{F}_\xi$ is built, representing the decision problem of asking whether there is a solution with sum-of-costs ξ and makespan μ. A SAT solver is tasked to check if $\mathcal{F}_\xi$ is solvable. If such a solution exists, it is returned. Otherwise

Δ and consequently ξ and μ are incremented by 1 and another iteration of building $\mathcal{F}_\xi$ and running the SAT solver is activated. This algorithm is complete but cannot detect unsolvability. However, this can be detected in polynomial time using other algorithms [Kornhauser *et al.*, 1984].

3 From Optimal to Suboptimal Solver

To convert SAT-MDD (Algorithm 1) to a suboptimal any solution algorithm, we simply remove the cardinality constraints from the construction of $\mathcal{F}_\xi$. Let $\mathcal{F}$ denote the resulting formula. Since $\mathcal{F}$ has all the constraints in $\mathcal{F}_\xi$ except the cardinality constraints, then clearly a satisfying assignment to $\mathcal{F}$ still represents a feasible solution (no collisions between agents etc.). Since $\mathcal{F}$ is less constrained than $\mathcal{F}_\xi$, we expect it to be solved faster. Indeed, we observed this in our preliminary experiments. Using $\mathcal{F}$ in Algorithm 1 instead of $\mathcal{F}_\xi$ looses, however sum-of-cost optimality.

Hence, replacing $\mathcal{F}_\xi$ with $\mathcal{F}$ in Algorithm 1 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 as **uMDD-SAT**. A key question is what is the suboptimality of the solutions uMDD-SAT returns? Is it really unbounded? We show later that even without the cardinality constraints, the suboptimality of the solutions outputted is bounded, due to how $\mathcal{F}$ is 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 runtime and suboptimality.

3.1 Bounded Suboptimal SAT-based Solver

The key to our bounded-suboptimal SAT-based solver is that it modified the Δ parameter used in construction of $\mathcal{F}_\xi$. In SAT-MDD, Δ is incremented by one in every iterations. Allowing Δ parameter to be less restrictive; that is, replace Δ with $\Delta' = \Delta + \delta$, where $\delta \geq 0$ is an integer value, produces formula of the same size but representing more solutions.[1] Since $\Delta' > \Delta$, we expect a formula with the sum-of-costs bounded by Δ' 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. Let us denote the formula $\mathcal{F}_\xi$ constructed for a TIG with μ layers (representing a makespan of μ) and Δ parameter as $\mathcal{F}(\mu, \Delta)$.

Proposition 2 *Let δ be a non-negative integer and let $\mathcal{F}(\mu_0 + \Delta, \Delta + \delta)$ be the first satisfiable formula encountered in the sequence of formulae $\mathcal{F}(\mu_0, \delta)$, $\mathcal{F}(\mu_0 + 1, 1 + \delta)$,...,$\mathcal{F}(\mu_0 + \Delta - 1, \Delta + \delta - 1)$, $\mathcal{F}(\mu_0 + \Delta, \Delta + \delta)$. Then solution represented by $\mathcal{F}(\mu_0 + \Delta, \Delta + \delta)$ has sum-of-costs $\xi \leq \xi_{opt} + \delta$ where ξ_{opt} is the optimal sum-of-costs for Σ.*

[1]The change from Δ to Δ' does not affect the number of clauses that represent the cardinality constraint, because we coded the cardinality constraints using a sequential counter, whose size is proportional to the number of propositional variable involved but not to the value of the bound [Sinz, 2005].

Algorithm 2: eMDD-SAT, an $(1+\epsilon)$-bounded suboptimal SAT-based MAFP solver

```
1  eMDD-SAT(MAPF Σ = (G = (V, E), A, α0, α+),ϵ)
2      μ0 = max_{ai∈A} ξ0(ai)
3      ξ0 = Σ_{ai∈A} ξ0(ai)
4      Δ ← 0
5      while Solution not found do
6          μ = μ0 + Δ
7          Δ' ← Δ + ϵ · (ξ0 + Δ)
8          F(μ, Δ') ← Build-Formula(Σ, μ, Δ')
9          Solution ← Consult-SAT-Solver(F(μ, Δ'))
10         if Solution not found then
11             Δ ← Δ + 1
12         end
13     end
14     return Solution
15 end
```

Proof: Formula $\mathcal{F}(\mu_0 + \Delta - 1, \Delta + \delta - 1)$ in the penultimate iteration was 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. But we also 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 formula $\mathcal{F}(\mu_0 + \Delta - 1, \Delta + \delta - 1)$ together with $\delta \geq 0$ implies that there is no solution of sum-of-costs $\xi_0 + \Delta - 1$ at all. Therefore, the optimal sum-of-costs is at least $\xi_0 + \Delta$. The solvability of $\mathcal{F}(\mu_0 + \Delta, \Delta + \delta)$ 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 2 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 an $(1 + \epsilon)$-bounded suboptimal algorithm.

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 $\delta(\Delta)$.*

Proof: Let $\delta(\Delta) = \epsilon \cdot (\xi_0 + \Delta)$. Hence the sum-of-costs of the solution returned by the algorithm is at most $(1+\epsilon) \cdot (\xi_0 + \Delta)$ while the optimum is at least $\xi_0 + \Delta$ hence the ratio between the sum-of-costs of returned solution and the sum-of-costs of the optimal one is at most $(1 + \epsilon)$. ■

The pseudo-code of the $(1 + \epsilon)$-bounded suboptimal SAT-based algorithm is presented as Algorithm 2. We refer to this algorithm as **eMDD-SAT**.

Note that a further minor improvement of the pseudo-code could be done which exploits the original optimization of the formula. Observe that in any solution to a MAPF problem it holds that $\mu \leq \xi \leq m \cdot \mu$. Therefore, if $\xi_0 + \Delta + \delta(\Delta) \geq \mu \cdot k$. then there is no need to add cardinality any constraints to $\mathcal{F}_\xi$, as the solution is guaranteed to be bounded by $\mu \cdot k$.

This inequality represents a limit of the degree of relax-

ation achievable by allowing more freedom over the cost bound imposed by the cardinality constraint. Hence the $(1+\epsilon)$-bounded suboptimal SAT-based algorithm tends to be near optimal anyway. Precisely, efectively the algorithm will be $\left(\frac{k\cdot(\mu_0+\Delta)}{\xi_0+\Delta}\right)$-bounded in the worst case.

4 Experimental Evaluation

We performed a large set of experiments to evaluate uMDD-SAT and eMDD-SAT, our suggested any solution and bounded suboptimal versions of MDD-SAT. We used various 4-connected grids as the underlying graphs.

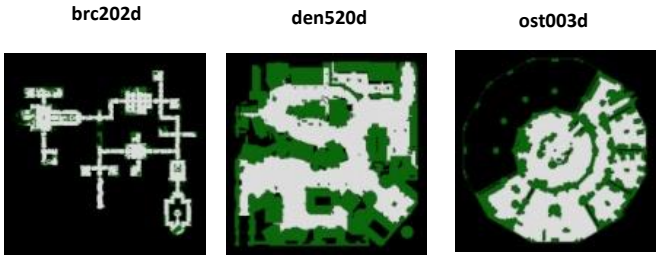

Figure 2: Dragon Age maps include: narrow corridors in `brc202d`, large open space in `den520d`, and open space with almost isolated rooms in `ost003d`.

(i) The first set of small densely populated instances consisted of girds of sizes 8×8, 16×16, and 32×32 with 10% nodes occupied by obstacles. To obtain instances of various difficulties the number of agents was varied from 1 to 32, 1 to 128, and 1 to 256 in case of 8×8, 16×16, and 32×32 grids respectively (the step was varied from 1 in the range of small units of agents to 16 in the range of hundreds of agents). Ten random instances were genereted for each number of agents by randomly choosing an initial position and then performing a random walk to set the target position.

(ii) Instances of the second testing set are based on three structurally different large maps taken from Sturtevant's repository [Sturtevant, 2012]. These are Dragon Age Origion (DAO) maps denoted as `brc202d`, `den520d`, and `ost003d` which are a standard benchmark for MAPF (see Figure 2). Again the number of agents was varied from 1 to 256 to obtain instances of various difficulties (the step ranged from 1 to 16) and 10 random instances were generated for each number of agents.

All tests were run on a machine with CPU Intel i7 3.2 Ghz, 8 GB RAM under Ubuntu Linux 15 and Windows 10 respectively. The timeout for all solvers has been set to 500 seconds.

4.1 Evaluation of the Unbounded Case

In this section we evaluate the performance of uMDD-SAT, our any solution suboptimal SAT-based solver. We compared uMDD-SAT with two suboptimal algorithms that are by design unbounded: PUSH-AND-SWAP [Luna and Bekris, 2011; de Wilde *et al.*, 2014], which is a polynomial time rule-based algorithm, and UNIROBOT[Surynek, 2015], which is a SAT-based algorithm that reduces MAPF with k agents to a problem of finding k vertex disjoint paths [Seymour, 1980]. We

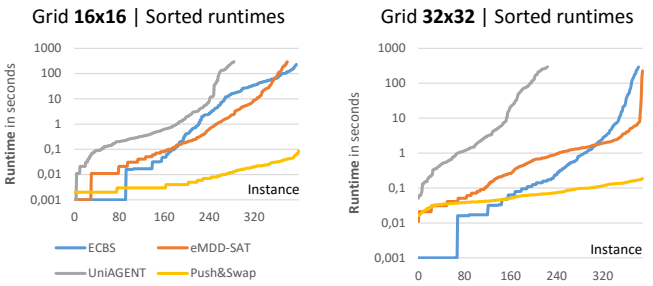

Figure 3: Runtimes of unbounded variants on grids of size 16×16, and 32×32.

also compared uMDD-SAT against ECBS [Barer *et al.*, 2014] a state-of-the-art bounded-suboptimal algorithm that is based on the CBS MAPF solver. To make the comparison with unbounded MAPF solver fair, we set the suboptimality bound of ECBS to a very large number (500).

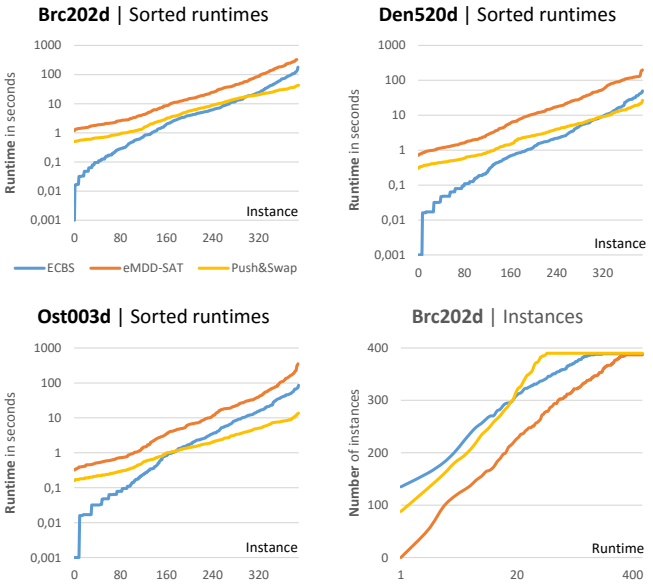

Figure 4: Runtimes of unbounded variants on DAO maps `brc202d`, `den520d`, and `ost003d`.

Runtime results of the experiments with the unbounded versions on small grids and DAO maps are shown in Figures 3 and 4 respectively. Runtimes for all testing instances that were below the limit of 500 seconds were sorted and shown in the figure (the x-axis corresponds to ordering of instances according to increasing runtime and the y-axis corresponds to runtime in seconds). The intuitive understanding of this presentation is that the faster algorithm has its line in the lower part of the figure.

Consider first the runtime results for the 16x16 and 32x32 grids (Figure 3). PUSH-AND-SWAP is the fastest algorithm in these small grids and UNIROBOT turned out to be worst performing algorithm. The comparison of ECBS and eMDD-SAT shows that in the easier instances (those that are sorted in the left-hand side of the x-axis), ECBS is faster, while for the

harder instnances (those on the right-hand side of the x-axis) eMDD-SAT performs better.

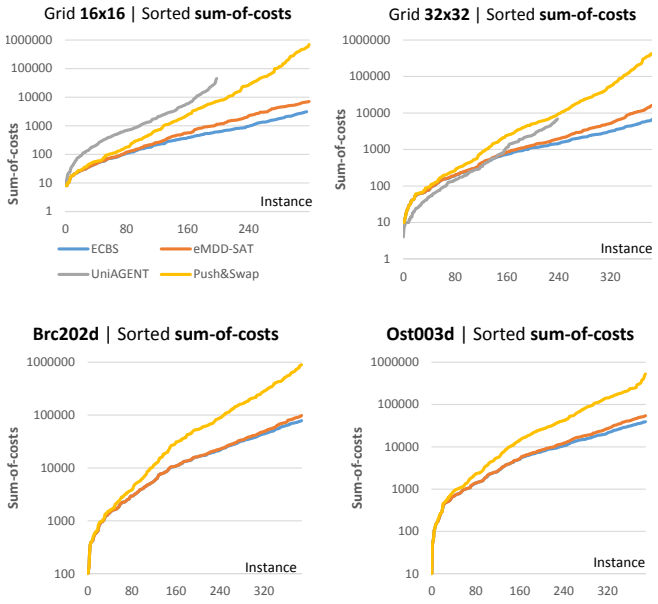

Figure 5: Sum-of-costs of unbounded variants on small grids and DAO maps.

Now consider the runtime results on the DAO maps (Figure 4), which are much larger than the aforementioned grids. Here too, UNIROBOT turned out to be the worst performing algorithm, and in general not applicable on large DAO maps. The bottom-right plot in Figure 4, which shows the number of instances solved by the remaining algorithms as a function of the runtime, that is, for a given x value the y value shows the number of instances solved given x seconds. All three algorithms (ECBS, PUSH-AND-SWAP, and uMDD-SAT) managed to solve all instances within the time limit, but uMDD-SAT was somewhat slower than the other two. Consider the other plots in Figure 4 we can conclude that in this domain ECBS was in general faster.

While the compared algorithms do not provide a bound on the sum-of-costs of their solutions, it may still be of interest in practice. We observed that the sum-of-costs of the solutions found by tested algorithms were significantly different from the optimum and from each other. Figure 5 shows the sum-of-costs of the solutions found for the DAO instances. This presentation is similar to the plots in the previous figures, but here the instances are sorted according ot their sum-of-costs. The interpretation is the same: lower curves corresponds to finding lower sum-of-costs. The results show that both UNIROBOT and PUSH-AND-SWAP generate worse solutions than ECBS and eMDD-SAT. The solutions quality returned by ECBS and eMDD-SAT are comparable, with slightly better solutions found by ECBS in some cases.

Altogether we can conclude that for unbounded suboptimal case uMDD-SAT is a reasonable option: perhaps not always the fastest or the one with the lowest sum-of-costs, but comparable to the state-of-the-art. This is encouraging, especially since if SAT solvers continue to become better, the performance of SAT-based algorithms such as uMDD-SAT will continue to improve.

4.2 Evaluation of the Bounded Case

Next, we conducted experiments to evaluate eMDD-SAT, our bounded-suboptimal MDD-SAT variant. Here we only compared againts ECBS as the other algorithms (PUSH-AND-SWAP and UNIROBOT) do not guarantee a bounded-suboptimal solution. The first set of experiments evaluate the behavior of both algorithms for different values of $1+\epsilon$, i.e., for different required suboptimality bounds. The same set of instances used for the unbounded experiments were also used here.

First, we measured the *success rate* of each algorithm, which is the the ratio of successfully solved instances under a predetermined time limit. The time limit in our experiments was 500 seconds. Figure 6 shows the algorithms' success rate (y-axis) as a function of the required suboptimality bound (the x-axis), which ranges from 1.1 to 1.0. Results for `32×32` with 100 agents and `ost003d` with 200 agents are shown. It can be observed that eMDD-SAT is better than ECBS for closer to optimal suboptimality bounds, outperforming ECBS starting at bound $(1+\epsilon) = 1.02$ and lower. For the 32x32 grids, which are more dense than the DAO map, the advantage of eMDD-SAT even begin earlier, again highlighting the advantage of SAT-based algorithms in harder problems. Next, we focus our evaluation on bound $(1+\epsilon) = 1.01$, to focus on the cases where eMDD-SAT is effective.

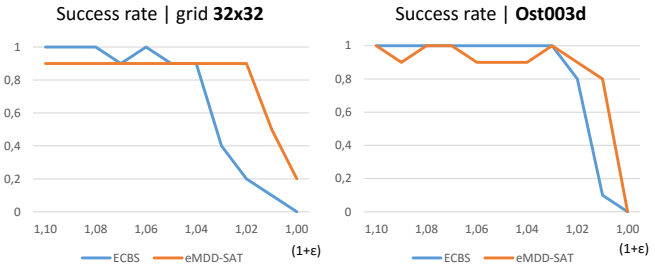

Figure 6: Success rate of the bounded variant with $(1+\epsilon)$ ranging from 1.1 to 1.0 (optimal case). Grid`32×32` contains 100 agents and DAO map `ost003d` 200 agents.

Results for $(1+\epsilon) = 1.01$ for small girds and DAO maps are presented in figures 7 and 8. In this we can observe that MDD-SAT tends to be faster in all small grids for the harder problems. In our analysis (results not shown for space limitation), we observed that these were the cases with higher density of agents.

Results for DAO maps indicate that in easier instances containing fewer agents ECBS is faster. However with the increasing difficulty of instances and density of agents the gap in performance is narrowed until eMDD-SAT starts to perform better in harder instances. This trend is best visible on the `ost003d` map.

Let us note that the maximum achievable ϵ by relaxing the cardinality constraint within the suboptimal eMDD-SAT approach for DAO maps is: $\epsilon = 1.47$ for `brc202d`, $\epsilon = 1.33$ for `den520d`, and $\epsilon = 1.12$ for `ost033d` all cases with

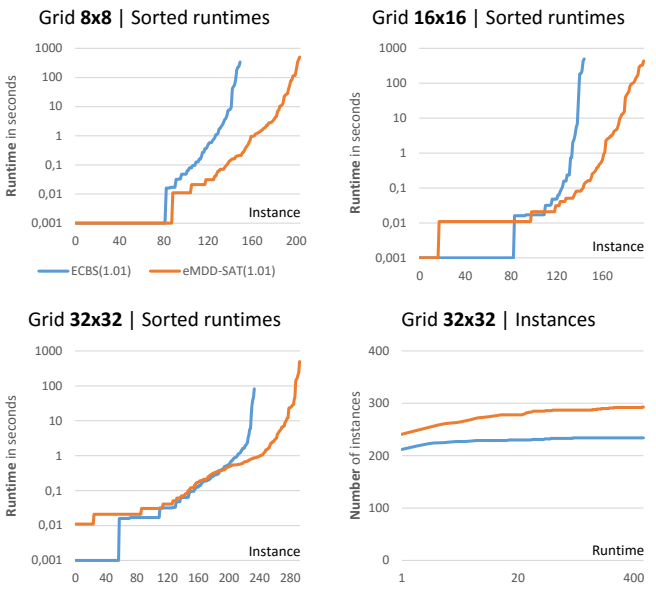

Figure 7: Runtimes of bounded variants ($\epsilon = 0.01$) on grids of size `8×8` and `16×16`, and `32×32`.

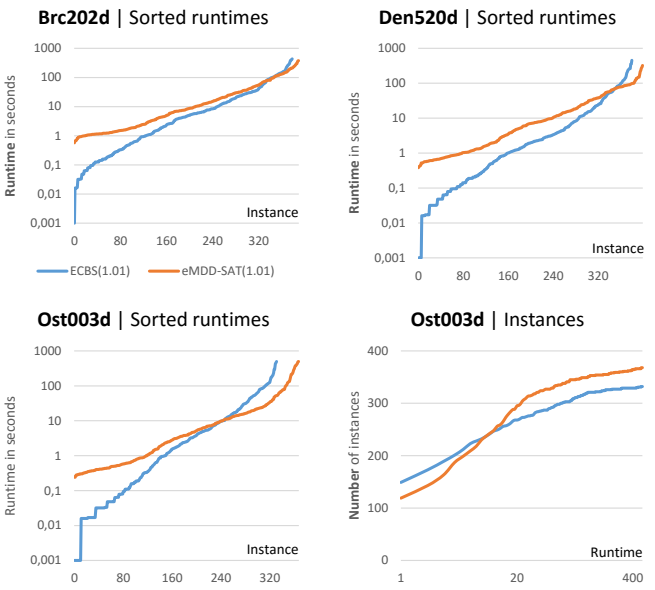

Figure 8: Runtimes of bounded variants ($\epsilon = 0.01$) on DAO maps `brc202d`, `den520d`, and `ost003d`.

200 agents. Setting these or greater bounds in eMDD-SAT is equivalent to complete removal of the cardinality constraint. That is, it is equivalent to running uMDD-SAT.

5 Conclusions

The SAT-based approach represented by eMDD-SAT has an advantage of using the learning mechanism built-in the external SAT solver. On the other hand, search based methods represented by ECBS are specially designed for solving MAPF and do not bring the overhead of a general purpose SAT solver. We attribute the good performance of the eMDD-SAT approach to clause learning mechanism.

This conclusion corresponds with the fact that advantage of eMDD-SAT appears in harder instances with long runs of the SAT solver where the clause learning mechanism has enough time to prune the search space efficiently. On the other hand the SAT-based approach has an overhead of building formula and communication with the external solver which negatively affects performance in sparsely occupied instances.

One of possible future research directions is to integrate learning mechanism into specialized MAPF solver which would eliminate the overhead of usage of the external SAT solver. Vertices represented within layers of MDD can be regarded as values of a multi-value decision variables representing positions of agents at individual time steps. Learning mechanism over such finite domain variables would be very similar to *nogood recording* known from modern CSP solvers [Dechter, 2003]. Reasoning about MAPF in the context of nogood recording and CSP would open door to higher level constraint propagation than that offered by SAT's unit propagation. Lastly, we believe that this work will open the way to developing bounded-suboptimal SAT-based algorithms for other planning problems.

References

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

[Barer *et al.*, 2014] Max Barer, Guni Sharon, Roni Stern, and Ariel Felner. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Symposium on Combinatorial Search (SoCS)*, 2014.

[Botea and Surynek, 2015] A. Botea and P. Surynek. Multi-agent path finding on strongly biconnected digraphs. In *AAAI*, pages 2024–2030, 2015.

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

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

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

[Dechter, 2003] Rina Dechter. *Constraint processing*. Elsevier Morgan Kaufmann, 2003.

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

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

[Kornhauser *et al.*, 1984] D. Kornhauser, G. Miller, and P. Spirakis. Coordinating pebble motion on graphs, the di-

ameter of permutation groups, and applications. In *FoCS*, pages 241–250, 1984.

[Luna and Bekris, 2011] R. Luna and K. E. Bekris. Push and swap: Fast cooperative path-finding with completeness guarantees. In *IJCAI*, pages 294–300, 2011.

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

[Sajid *et al.*, 2012] Qandeel Sajid, Ryan Luna, and Kostas Bekris. Multi-agent pathfinding with simultaneous execution of single-agent primitives. In *SOCS*, 2012.

[Seymour, 1980] P.D. Seymour. Disjoint paths in graphs. *Discrete Mathematics*, 29(3):293 – 309, 1980.

[Sharon *et al.*, 2013] G. Sharon, R. Stern, M. Goldenberg, and A. Felner. The increasing cost tree search for optimal multi-agent pathfinding. *Artif. Intell.*, 195:470–495, 2013.

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

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

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

[Sinz, 2005] C. Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In *CP*, 2005.

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

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

[Surynek *et al.*, 2016a] Pavel Surynek, Ariel Felner, Roni Stern, and Eli Boyarski. Efficient SAT approach to multi-agent path finding under the sum of costs objective. In *ECAI*, pages 810–818, 2016.

[Surynek *et al.*, 2016b] Pavel Surynek, Ariel Felner, Roni Stern, and Eli Boyarski. An empirical comparison of the hardness of multi-agent path finding under the makespan and the sum of costs objectives. In *Symposium on Combinatorial Search (SoCS)*, 2016.

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

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

[Surynek, 2014] P. Surynek. Compact representations of cooperative path-finding as SAT based on matchings in bipartite graphs. In *ICTAI*, pages 875–882, 2014.

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

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

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

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