



Variants of Independence Detection in SAT-Based Optimal Multi-agent Path Finding

Pavel Surynek¹(✉), Jiří Švancara², Ariel Felner³, and Eli Boyarski⁴

¹ National Institute of Advanced Industrial Science and Technology (AIST),
Tokyo, Japan

pavel.surynek@aist.go.jp

² Faculty of Mathematics and Physics, Charles University, Prague, Czechia

jiri.svancara@mff.cuni.cz

³ Ben Gurion University, Beer-Sheva, Israel

felner@bgu.ac.il

⁴ Bar-Ilan University, Ramat-Gan, Israel

eli.boyarski@gmail.com

Abstract. In multi-agent path finding (MAPF) on graphs, the task is to find paths for distinguishable agents so that each agent reaches its unique goal vertex from the given start while collisions between agents are forbidden. A cumulative objective function is often minimized in MAPF. The main contribution of this paper consists in integrating independence detection technique (ID) into a compilation-based MAPF solver that translates MAPF instances into propositional satisfiability (SAT). The independence detection technique in search-based solvers tries to decompose a given MAPF instance into instances consisting of small groups of agents with no interaction across groups. After the decomposition phase, small instances are solved independently and the solution of the original instance is combined from individual solutions to small instances. The presented experimental evaluation indicates significant reduction of the size of instances translated to the target SAT formalism and positive impact on the overall performance of the solver.

Keywords: Multi-agent path-finding (MAPF) · Independence detection (ID) · Propositional satisfiability (SAT) · Cost optimality · Makespan optimality · Sum-of-costs optimality · SAT encodings · Path-finding on grids

1 Introduction

Multi-agent path finding (MAPF) represents a task of finding collision free paths for a set of mobile agents where each agent is assigned unique start and goal positions [13, 19, 22, 27]. An environment with agents is often abstracted by *undirected graph* in the literature [17, 36]. Agents in this abstraction are represented by items placed in vertices of the graph while edges represent passable regions. At most one agent can be placed in each vertex to model physical space occupancy by agents. Agents can traverse a single edge at each time step.

Various movement schemes exist for MAPF on graphs. Often an agent can move into an unoccupied neighbor not entered by another agent [28] – this will be called *move-to-unoccupied* variant. This variant requires at least one vertex in the graph unoccupied to be able to perform some movements at all. Other movement schemes include chains of agents moving simultaneously with only the leader entering the unoccupied vertex or cases with no vacant vertex in the graph where rotations along non-trivial cycles are allowed [32]. We base our presentation on the *move-to-unoccupied* variant. Let us note that techniques shown in this paper are generic across all these movement schemes.

The MAPF problem and its variants are strongly practically motivated. Applications range from navigation of multiple mobile robots [5, 8] through traffic optimization [12, 15] to movement planning in computer games [34]. We refer the reader to various studies such as [19, 20] for the detailed survey of applications.

1.1 Optimality in MAPF

We address *optimal* MAPF in which we search for paths that are optimal with respect to a given objective. The two basic cumulative objectives studied in the literature are *sum-of-costs* [19, 25] and *makespan* [29].

The **sum-of-costs objective** assumes that unit costs are assigned to move and wait actions. The cost of plan is the sum of action costs along all the paths and over all agents. The aim is to obtain a plan with the minimum cost.

Under the **makespan objective**, the aim is to obtain a plan that can be executed in as short as possible time while each movement consumes 1 unit of time. In other words, we need the longest path out of all the paths to be as short as possible.

As we will show later, there may be situations where the increase in the sum-of-costs leads to a shorter makespan and vice versa.

A feasible solution of MAPF can be found in polynomial time [13, 35]. Adding any of the discussed objectives renders the decision version of MAPF (a *yes/no question* if a given MAPF has a solution of specified makespan/sum-of-costs) to be NP-complete [16, 28, 32].

We will keep the further description around the sum-of-costs variant but it is important to note that the presented techniques apply for the makespan variant as well.

1.2 Contributions to SAT-Based MAPF

Techniques for solving MAPF optimally include translation of the decision version into *propositional formula* [10, 11]. The formula is *satisfiable* if and only if the instance of MAPF is solvable for a given value of the objective function. Assuming that satisfiability of such formula is a non-decreasing function of the value of objective function, the optimum can be obtained by querying the satisfiability multiple times. A trivial strategy of increasing the value of objective function by one turned out to be a good choice [30] in many cases (thanks to the non-uniform difficulty of each query).

Satisfiability of the formula can be decided by an off-the-shelf SAT solver [2, 6] which is one of the advantages of the SAT-based approach. All advanced techniques developed in recent SAT solvers [4] can be employed for solving MAPF in this way.

The most significant bottleneck of all existing SAT-based algorithms for MAPF is the large size and combinatorial difficulty of the target propositional formula that grow significantly with the increasing number of agents as well as with growing size of the underlying graph. This kind of growth of combinatorial difficulty has already been addressed by Standley [24] in his search-based optimal MAPF solving algorithm. Standley described various variants of a method called *independence detection* that tries to determine the smallest possible groups of agents for which paths can be found independently of other groups.

Our contribution consists in integrating two variants of independence detection – *simple independence detection* (SID) and *independence detection* (ID) – with MDD-SAT – the most recent SAT-based MAPF solver [30]. As there are differences in how the original Standley’s search-based algorithm and SAT-based approach work, we suggested modifications to ID to be compatible with the SAT-based approach. Our new solvers are called MDD-SAT+SID and MDD-SAT+ID following the notation of [24]. Conducted experiments demonstrate similar performance benefit as in the case of original application of SID and ID in the search-based approach.

This paper is an extension of [31]. We describe in more detail the encoding of MAPF to a Boolean formula. In addition to [31] we also present experimental evaluation of MDD-SAT+SID. The paper is organized as follows. After the formal introduction of the MAPF problem a brief exposition of related work is done. Then, the Boolean encoding of MAPF is presented, also the original SID and ID are recalled and their integration with the SAT-based approach is presented. Finally, an experimental evaluation with grids and large maps is presented.

2 MAPF Definition

An arbitrary **undirected graph** can be used to model the environment where agents are moving. Let $G = (V, E)$ be such a graph where $V = \{v_1, v_2, \dots, v_n\}$ is a finite set of vertices and $E \subseteq \binom{V}{2}$ is a set of edges.

The placement of agents in the environment is modeled by assigning them vertices of the graph. Let $A = \{a_1, a_2, \dots, a_m\}$ be a finite set of *agents*. Then, an arrangement of agents in vertices of graph G will be fully described by a *location* function $\alpha: A \rightarrow V$; the interpretation is that an agent $a \in A$ is located in a vertex $\alpha(a)$. At most **one agent** can be located in each vertex; that is α is uniquely invertible.

Definition 1 (MAPF). An instance of *multi-agent path-finding* problem is a quadruple $\Sigma = [G = (V, E), A, \alpha_0, \alpha_+]$ where location functions α_0 and α_+ define the initial and the goal arrangement of a set of agents A in G respectively. \square

The dynamicity of the model assumes a discrete time divided into time steps. An arrangement α_i at the i -th time step can be transformed by a transition action which

instantaneously moves agents in the non-colliding way to form a new arrangement α_{i+1} . The transition between α_i and α_{i+1} must satisfy the following *validity conditions*:

- (1) $\forall a \in A$ either $\alpha_i(a) = \alpha_{i+1}(a)$ or $\{\alpha_i(a), \alpha_{i+1}(a)\} \in E$ holds
(agents move along edges or wait at their current location),
- (2) $\forall a \in A \alpha_i(a) \neq \alpha_{i+1}(a) \Rightarrow \alpha_i^{-1}(\alpha_{i+1}(a)) = \perp$
(agents move to vacant vertices only), and
- (3) $\forall a, b \in A a \neq b \Rightarrow \alpha_{i+1}(a) \neq \alpha_{i+1}(b)$
(no two agents enter the same target/unique invertibility of resulting arrangement).

The task in MAPF is to transform α_0 using above valid transitions to α_+ . An illustration of MAPF and its solution is depicted in Fig. 1.

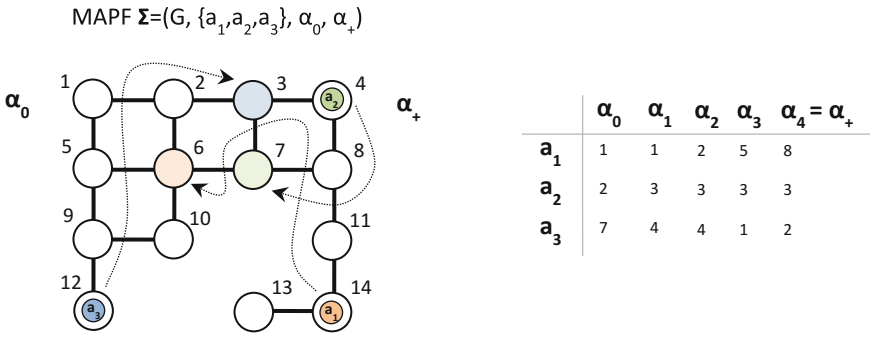


Fig. 1. An example of a MAPF instance from [31] with three agents a_1 , a_2 , and a_3 (left). A solution of the instance is shown (right).

Definition 2 (MAPF solution). A *solution* for MAPF instance $\Sigma = [G, A, \alpha_0, \alpha_+]$ is a sequence of arrangements $[\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_\mu]$ where $\alpha_\mu = \alpha_+$ and α_{i+1} is a result of valid transition from α_i for every $i = 1, 2, \dots, \mu - 1$. □

Makespan μ is the total number of time steps until the last agent reaches its destination. *Sum-of-costs* denoted ξ is the sum of path costs per individual agents. Each action (including wait) of an agent before it reaches its goal has unit cost.

2.1 Makespan vs. Sum-of-Costs

There exists an instance in which all the sum-of-costs optimal solutions are not makespan optimal. Similarly, none of the makespan optimal solution is sum-of-costs optimal there (see Fig. 2 for illustration).

In the SAT-based optimal MAPF solver described below, a proper relation between makespan and sum-of-costs need to be found as both objectives are bounded during search. We need to ensure that smallest cost found under the given makespan bound is optimal (see [30] for more detailed discussion).

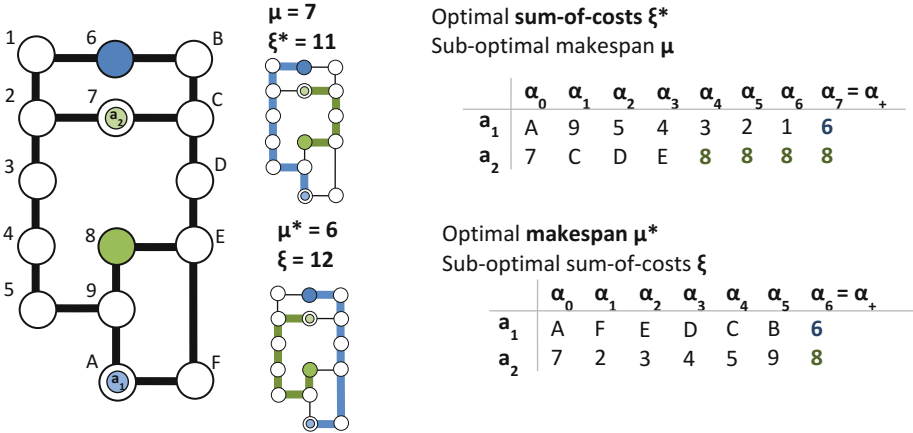


Fig. 2. An instance of the MAPF problem from [31] in which no *makespan* optimal solution is sum-of-costs optimal and no *sum-of-costs* optimal solution is makespan optimal.

3 Related Work

Many other successful algorithms exist for the optimal MAPF solving. The state-of-the-art search-based algorithms (though there is no universal winner) include *increasing cost tree search* - ICTS [19], *conflict base search* - CBS [20], and *improved CBS* - ICBS [7]. These algorithms excel in setups with relatively few agents on large maps.

Another research direction is represented by methods based on reduction of the MAPF problem to another formalism. Except the SAT as a target formalism, successful attempts to reduce MAPF to *constraint optimization problem* [18], *inductive logic programming* [33], and *answer set programming* [9] have been made. These approaches (the SAT approach including) can be generally characterized by a high performance in MAPFs with small underlying graph densely populated with agents. This is a natural outcome of the maturity of solvers used to solve hard combinatorial problems in the target formalism.

Recently new research directions driven by applications have been identified in the MAPF context. For example, it is not always necessary to distinguish between individual agents – see [14] for detailed survey.

4 SAT Encoding for Optimal Sum-of-Costs

In this paper, we follow the algorithm solving sum-of-cost optimal MAPF via reduction to SAT presented in [30].

The basic approach in solving MAPF via SAT is to create a *time expansion graph* (denoted TEG) [29]. A TEG is a directed acyclic graph (DAG). First, the set of vertices of the underlying graph G are duplicated for all time-steps from 0 up to the given

bound μ . Then, possible actions (move along edges or wait) are represented as directed edges between successive time steps. Formally a TEG is defined as follows:

Definition 3 (TEG). *Time expansion graph of depth μ* for underlying graph (V, E) is a digraph (V_μ, E_μ) where $V_\mu = \{u_j^t | t = 0, 1, \dots, \mu \wedge u_j \in V\}$ and $E_\mu = \{(u_j^t, u_k^{t+1}) | t = 0, 1, \dots, \mu - 1 \wedge (\{u_j, u_k\} \in E \vee j = k)\}$. \square

The encoding for MAPF introduces propositional variables and constraints for a single time-step t in order to represent any possible arrangement of agents at time t . Given a desired makespan μ , the formula represents the question of whether there is a solution in the TEG of μ time steps. The search for optimal makespan is done by iteratively incrementing μ ($\mu = 0, 1, 2, \dots$) until a satisfiable formula is obtained.

To find the optimal sum-of-costs solution, we use similar technique as with optimal makespan solution. The sequence of decision problems is whether there exists a solution of a given sum-of-cost ξ . However, encoding this decision problem is more challenging than the makespan case, because one needs to both bound the sum-of-costs, but also to predict how many time expansions are needed. We address this challenge by using two key techniques described next: (1) Cardinality constraint for bounding ξ and (2) Bounding the Makespan.

4.1 Cardinality Constraint for Bounding ξ

The SAT literature offers a technique for encoding a *cardinality constraint* [3, 21], which allows calculating and bounding a numeric cost within the formula. Formally, for a bound $\lambda \in \mathbb{N}$ and a set of propositional variables $X = \{x_1, x_2, \dots, x_k\}$ the cardinality constraint $\leq_\lambda \{x_1, x_2, \dots, x_k\}$ is satisfied iff the number of variables from the set X that are set to TRUE is $\leq \lambda$.

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

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

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

To find a μ value that meets R1, we require the following definitions. Let $\xi_0(a_i)$ be the 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 was called the sum of individual costs (SIC) [19]. ξ_0 is an admissible heuristic for optimal sum-of-costs search algorithms, since ξ_0 is a lower bound on the minimal sum-of-costs. ξ_0 is calculated by relaxing the problem by omitting the other agents. Similarly, we define $\mu_0 = \max_{a_i \in A} \xi_0(a_i)$. μ_0 is length of the *longest* of the shortest individual paths and

is thus a lower bound on the minimal makespan. Finally, let Δ be the extra cost over SIC (as done in [19]). That is, let $\Delta = \xi - \xi_0$.

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

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

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

Algorithm 1. SAT consult illustrating the increase in Δ .

```

MAPF-SAT (MAPF  $\Sigma = (G = (V, E), A, \alpha_0, \alpha_+)$ )
   $\mu_0 = \max_{a_i \in A} \xi_0(a_i)$ ,  $\Delta = 0$ 
  while Solution not found do
     $\mu = \mu_0 + \Delta$ 
    for each agent  $a_i$  do
      Build  $TEG_i(\mu)$ 
    end
    Solution = Consult-SAT-Solver( $\Sigma, \mu, \Delta$ )
    if Solution not found then
       $\Delta ++$ 
    end
  end
  return (Solution)
end

```

This algorithm clearly terminates 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. The unsolvability of an MAPF instance can be checked separately by a polynomial-time complete sub-optimal algorithm such as PUSH-AND-ROTATE [35].

4.3 Efficient Use of the Cardinality Constraint

The complexity of encoding a cardinality constraint depends linearly in the number of constrained variables [21, 23]. Since each agent a_i must move at least $\xi_0(a_i)$, we can reduce the number of variables counted by the cardinality constraint by only counting the variables corresponding to extra movements over the first $\xi_0(a_i)$ movement a_i makes. We implement this by introducing a TEG for a given agent a_i (labeled TEG_i).

TEG_i differs from TEG (Definition 3) in that it distinguishes between two types of edges: E_i and F_i . E_i are (directed) edges whose destination is at time step $\leq \xi_0(a_i)$. These are called standard edges. F_i denoted as extra edges are directed edges whose destination is at time step $\geq \xi_0(a_i)$. Figure 3 shows an underlying graph for agent a_1 (left) and the corresponding TEG_1 . Note that the optimal solution of cost 2 is denoted by the diagonal path of the TEG. Edges that belong to F_i are those that their destination is time step 3 (dotted lines). The key in this definition is that the cardinality constraint would only be applied to the extra edges, that is, we will only bound the number of extra edges (they sum up to Δ) making it more efficient. There are various possibilities to define what happens to an agent when it reaches the goal (disappears, waits etc.). In all cases, edges in TEGs corresponding to wait actions at the goal are not marked as extra. Importantly, our SAT approach is robust across all these variants.

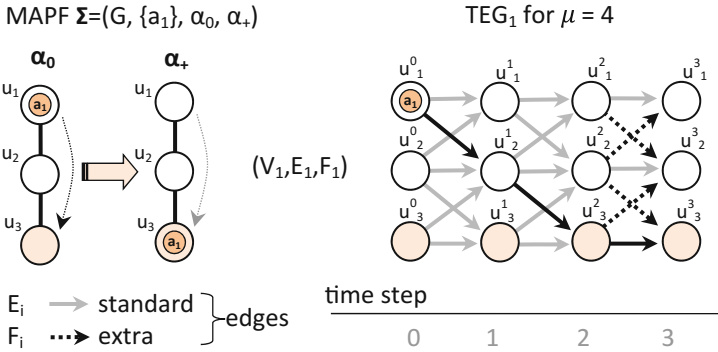


Fig. 3. A TEG for an agent that needs to go from u_1 to u_3 .

4.4 Detailed Description of the SAT Encoding

Agent a_i must go from its initial position to its goal within TEG_i . This simulates its location in time in the underlying graph G . That is, the task is to find a path from $a_0^0(a_i)$ to $a_+^\mu(a_i)$ in TEG_i . The search for such a path will be encoded within the Boolean formula. Additional constraints will be added to capture all movement constraints such as *collision avoidance* etc. And, of course, we will encode the cardinality constraint that the number of extra edges must be exactly Δ .

We want to ask whether a sum-of-costs solution of ξ exists. For this we build TEG_i for each agent $a_i \in A$ of depth $\mu_0 + \Delta$. We use V_i to denote the set of vertices in TEG_i

that agent a_i might occupy during the time steps. Next we introduce the Boolean encoding (denoted BASIC-SAT) which has the following Boolean variables:

1. $\chi_j^t(a_i)$ for every $t \in \{0, 1, \dots, \mu\}$ and $u_j^t \in V_i$ – Boolean variable of whether agent a_i is in vertex v_j at time step t .
2. $\mathcal{E}_{j,k}^t(a_i)$ for every $t \in \{0, 1, \dots, \mu - 1\}$ and $(u_j^t, u_k^{t+1}) \in (E_i \cup F_i)$ – Boolean variable that model transition of agent a_i from vertex v_j to v_k through any edge (standard or extra) between time steps t and $t + 1$ respectively.
3. $C^t(a_i)$ for every $t \in \{0, 1, \dots, \mu - 1\}$ such that there exist $u_j^t \in V_i$ and $u_k^{t+1} \in V_i$ with $(u_j^t, u_k^{t+1}) \in F_i$ – Boolean variables that model cost of movements along extra edges (from F_i) between time steps t and $t + 1$.

We 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 $T_\mu = \{0, 1, \dots, \mu - 1\}$. Several groups of constraints are introduced for each agent $a_i \in A$ as follows:

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

$$\chi_j^t(a_i) \Rightarrow \bigvee_{(u_j^t, u_k^{t+1}) \in (E_i \cup F_i)} \mathcal{E}_{j,k}^t(a_i) \quad (1)$$

$$\bigwedge_{(u_j^t, u_k^{t+1}), (u_j^t, u_l^{t+1}) \in (E_i \cup F_i) \wedge k < l} \neg \mathcal{E}_{j,k}^t(a_i) \vee \neg \mathcal{E}_{j,l}^t(a_i) \quad (2)$$

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

$$\mathcal{E}_{j,k}^t(a_i) \Rightarrow \chi_j^t(a_i) \wedge \chi_k^{t+1}(a_i) \quad (3)$$

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

$$\mathcal{E}_{j,k}^t(a_i) \Rightarrow \bigwedge_{a_l \in A \wedge a_l \neq a_i \wedge u_k^{t+1} \in V_l} \chi_k^{t+1}(a_l) \quad (4)$$

- C4: No two agents can appear in the same vertex at the same time step (although the previous constraint ensures that an agent does not collide with an agent currently residing in a vertex it does not prevent simultaneous entering of the same vertex by multiple agents). That is the following constraint is added for every $t \in T_\mu$ and pair of agents $a_i, a_l \in A$ such that $i \neq l$.

$$\bigwedge_{u_j^t \in V_i \cap V_i} \neg \chi_j^t(a_i) \vee \neg \chi_j^t(a_i) \quad (5)$$

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

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

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

$$\leq_\Delta \left\{ C^t(a_i) \mid i = 1, 2, \dots, n \wedge t = 0, 1, \dots, \mu - 1 \wedge \left\{ (u_j^t, u_k^{t+1}) \in F_i \right\} \neq \emptyset \right\} \quad (7)$$

The resulting Boolean formula that is a conjunction of C1 ... C7 will be denoted as $\mathcal{F}_{BASIC}(\Sigma, \mu, \Delta)$ and is the one that is consulted by Algorithm 1.

The following proposition summarizes the correctness of our encoding.

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

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

As discussed in [30], the limitation of BASIC-SAT encoding is its size which is implied by the size of the time expanded graph. To mitigate this limitation Surynek et al. took inspiration from another successful search-based solver called *increasing cost tree search* (ICTS) [19]. Vertices whose sum of distances from $a_0^0(a_i)$ and $a_+^\mu(a_i)$ in TEG_i is greater than μ can never be visited by a_i in any optimal solution or else a_i would not have enough time steps to reach $a_+^\mu(a_i)$. Omitting those vertices from TEGs that are too far in the aforementioned sense would not compromise soundness of the solving process but would lead to a smaller formula. In [30], this version of TEGs where unreachable vertices are omitted is called MDD and corresponding formula is denoted as $\mathcal{F}_{MDD}(\Sigma, \mu, \Delta)$. When referring to MDD-SAT solver we assume the version with MDDs.

Using MDDs can rule out many vertices that would be normally considered in standard time expansions. Experiments confirmed that MDDs enabled using the SAT-based approach even for large MAPF instances for which the size of encodings without MDD was prohibitive.

5 Independence Detection

Our major aim is to increase performance of the SAT-based MAPF solver by reducing the number of agents needs to be considered at once. This has been successfully done in search based methods via a technique called *independence detection*.

In this section, we will describe the original method of independence detection proposed by Standley (2010). The main idea behind this technique is that difficulty of MAPF solving 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 at low computational effort, and then combine them.

The simple approach, called *simple independence detection* (SID), assigns each agent to a group so that every group consists of exactly one agent. Then, for each of these groups, an optimal solution is found independently. Every pair of these solutions is evaluated and if the two groups' solutions are in conflict (that is, when collision of agents belonging to different group occurs), the groups are merged and replanned together. If there are no conflicting solutions, the solutions can be merged to a single solution of the original problem. This approach can be further improved by avoiding merging of groups.

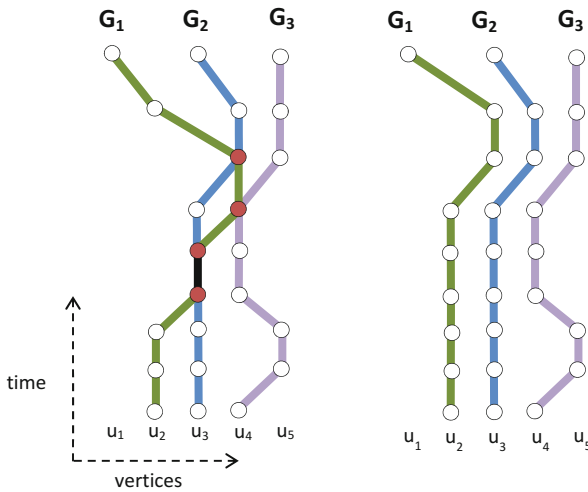


Fig. 4. A schematic illustration from [31] of path replanning within the independence detection technique. A path for the group G_1 conflicted with paths of other two groups (left part). Then path for G_1 has been successfully replanned (right part).

Generally, each agent has more than one possible optimal path. However, SID considers only one of these paths. The improvement of SID known as independence detection (ID) is as follows. Let's have two conflicting groups G_1 and G_2 . First, try to replan G_1 so that the new solution has the same cost and the steps that are in conflict with G_2 are forbidden.

If no such solution is possible, try to similarly replan G_2 . If this is not possible, merge G_1 and G_2 into a new group. In case either of the replanning was successful, that group needs to be evaluated with every other group again. This can lead to infinite cycle. Therefore, if two groups were already in conflict before, merge them without trying to replan.

Algorithm 2. MAPF solving algorithm based on **independence detection** (ID) technique. Planning for groups is always done to have least number of conflicts w.r.t. conflict avoidance table.

```

assign each agent to a group;
 $\text{plan}$  a path for each group by A*;
fill conflict avoidance table;
while conflicting groups exist
   $G_1, G_2 =$  conflicting groups;
  if  $G_1, G_2$  not conflicted before
     $\text{replan}$   $G_1$  by A* with illegal moves based on  $G_2$ ;
    if failed to replan  $G_1$ 
       $\text{replan}$   $G_2$  by A* with illegal moves based on  $G_1$ ;
    endif
  endif
  if no alternate paths for  $G_1, G_2$ 
    merge  $G_1$  and  $G_2$ ;
     $\text{plan}$  a path for new group by A*;
  endif
  update conflict avoidance table;
end
return combined paths of all groups;

```

Standley uses ID in combination with the A* algorithm. While planning, it is preferred to find paths that create the least possible amount of conflicts with other groups that have already planned paths. For this purpose, the conflict avoidance table is created (see Algorithm 2 for pseudo-code).

The table stores moves of agents in other groups. In case A* has a choice between several nodes with the same minimal $f()$ cost, the one with least amount of conflicts is expanded first. This technique yields an optimal solution that has a minimal number of conflicts with other groups. This property is useful when replanning of a group's solution is needed.

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 Integrating SID and ID into MDD-SAT

SID can be integrated into the SAT-based framework as a top-level algorithm where MDD-SAT merely serves as a procedure for optimal MAPF solving restricted on an individual group. Hence, no modification of the core MDD-SAT procedure is needed.

ID 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 occupied positions in the *conflict avoidance table*. In the yes/no SAT environment we either manage to avoid occupied positions or not while in the negative case there is no easy tool how to control the number of conflicts.

The SAT-based version of ID works in similar way to the original version of Standley but instead of resolving conflicts between a pair of conflicting groups G_1 and G_2 it resolves conflict of group G_1 with all other groups. If this attempt is successful, G_1 is independent on others and the process can continue with resolving conflicts between remaining groups (see Fig. 4 where G_1 has been made independent).

If the attempt to resolve conflict between G_1 and G_2 by making G_1 independent fails, the same is tried for G_2 . If the attempt for G_2 fails too groups are merged. The pseudo-code is shown as Algorithm 3.

In contrast to original ID we strictly require avoidance with respect to the conflict avoidance table instead of stating it as a preference only. This is technically done by omitting the conflicting vertices in the MDD. The SAT approach does not allow to express a preference like in the search based algorithm. This is the reason why ID in the SAT-based solver differs from the original one.

Algorithm 3. Independence detection in the SAT-based framework. Conflict avoidance is strictly required.

```

assign each agent to a group;
plan a path for each group  $G_1, \dots, G_k$  by MDD-SAT;
fill conflict avoidance table;
while conflicting groups exist
   $G_1, G_2 =$  conflicting groups;
  if  $G_1, G_2$  not conflicted before
    replan  $G_1$  by MDD-SAT with illegal moves based on
       $\{G_1, \dots, G_k\} - G_1$ ;
    if failed to replan  $G_1$ 
      replan  $G_2$  by MDD-SAT with illegal moves based on
         $\{G_1, \dots, G_k\} - G_2$ ;
    endif
  endif
  if no alternate paths for  $G_1, G_2$ 
    merge  $G_1$  and  $G_2$ ;
    plan a path for new group by MDD-SAT;
  endif
  update conflict avoidance table;
end
return combined paths of all groups;

```

7 Experiments

We performed experimental comparison of the proposed MDD-SAT+SID and MDD-SAT+ID solvers with other state-of-the-art solvers – namely with the previous best SAT-based solver MDD-SAT and also with search-based algorithms ICTS and ICBS.

The MDD-SAT+SID and MDD-SAT+ID have been implemented in C++ as an extension of an existing implementation of the MDD-SAT solver. A couple of minor improvements have been done in the original MDD-SAT encoding – some auxiliary propositional variables have been eliminated which reduced the size of the encoding and consequently saved runtime while generating formulae (this improvement affects both MDD-SAT and new MDD-SAT+SID, MDD-SAT+ID used in presented experiments).

We used `Glucose 3.0` [1] in variants of MDD-SAT which is a top performing SAT solver according to the recent SAT Competitions [4]. The complete implementation of the MDD-SAT solvers is available on-line to allow reproducibility of the presented results: <http://ktiml.mff.cuni.cz/~surynek/research/icaart2017>.

ICTS and ICBS have been implemented in C#. The original implementations of these algorithms have been used.

All the tests were run on Xeon 2 Ghz, and on Phenom II 3.6 Ghz, both with 12 Gb of memory.

The experimental setup followed the scheme used in the literature [22] which tests MAPF algorithms on 4-connected grids. Let us note however that all the suggested algorithms are designed and implemented for general undirected graphs (the fact that grids are used in the experiments is not exploited to increase efficiency of solving in any way).

7.1 Small Grids Evaluation

The first series of experiments takes place on small square grids of sizes 8×8 , 16×16 , and 32×32 with 10% of vertices occupied by obstacles. In this setup of the environment, we increased population of agents from 1 and observed the runtime of all the solvers until no solver was able to solve the instance within the given time limit of 300 s (this was 20 agents for 8×8 grid, and 40 and 60 for 16×16 and 32×32 grids respectively).

Ten randomly generated instances per number of agents were used. The initial positions were generated by choosing a subset of vertices randomly. The goal arrangement has been generated as a long random walk from the initial state following valid moves – this ensured solvability of all the tested instances.

To be able to communicate results of experiments more easily we intuitively distinguish three different categories of instances with respect to the density of agents as follows. The behavior of solvers is then discussed with respect to these categories:

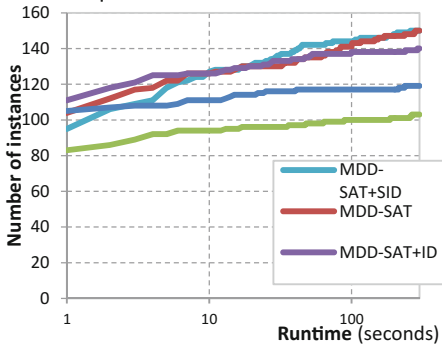
- **Low density** – few interactions among agents, paths for individual agents can be planned independently.
- **Medium density** – some interaction among agents are inevitable but there exist multiple groups of agents that are independent of each other.
- **High density** – majority of agents are interdependent and form one large group.

The small grid experiment contains instances from all these three cases. The hypothesis is that the SID and ID technique will be helpful in instances with medium density of agents while ID is expected to reach benefit in higher densities of agents. We also expect that in the case of low density of agents there will be some benefit of SID and ID since many agents will just follow their shortest paths towards goals in such a case. As in low and medium density cases the complexity of the formula is not proportional to the difficulty of the instance.

Furthermore, we expect rather negative effect of using SID and ID in instances with high density of agents. This is because of the fact that most agents will be gradually merged into a large group while the process of merging represents an overhead in such a case.

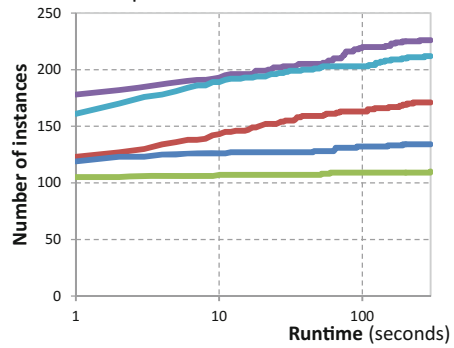
Solved instances

Grid 8x8 | 10% obstacles



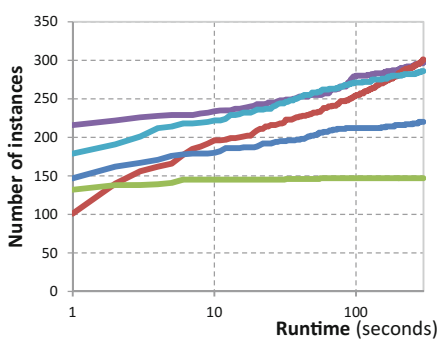
Solved instances

Grid 16x16 | 10% obstacles



Solved instances

Grid 32x32 | 10% obstacles



Sorted Runtimes | Grid 32x32

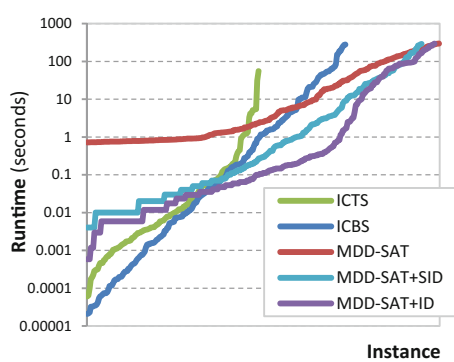


Fig. 5. Results of experiments on **small grid maps** of sizes 8×8 , 16×16 , and 32×32 . Figures show how many instances were solved within the given runtime and sorted runtimes (right bottom part). Clearly versions of MDD-SAT dominate in the test over search based algorithms ICTS and ICBS except few quickly solvable cases. Moreover, MDD-SAT+ID and MDD-SAT+SID outperforms MDD-SAT in cases with low to medium density of agents. MDD-SAT+ID and MDD-SAT+SID exhibit similar performance while ID shows its advantage in instances requiring more time.

Experimental result for the small grids (see Fig. 5) confirmed the hypothesis. MDD-SAT+SID/ID win in low to medium density of agents. For the higher density of agents, both MDD-SAT+SID/ID tend to be eventually outperformed by the original MDD-SAT. If SID and ID are compared then we can see that ID has more significant benefit than SID in most cases.

7.2 Large Maps – Dragon Age

We also experimented on three structurally different large maps from Dragon Age: Origins [26] – *ost003d*, *den520d*, and *brc202d* (see Fig. 6). Our choice of maps is driven by the choice of authors in the previous literature [20, 30].

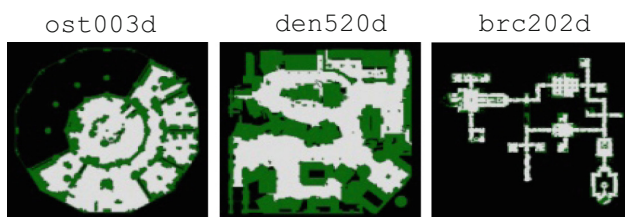


Fig. 6. Illustration of large Dragon Age maps *ost003d* (size 194×194), *den520d* (size 257×256), and *brc202d* (size 481×530).

We used setup with 16 and 32 agents randomly paced agents which represents low to medium density. Let us note that a case with high density of agents in the map of that size is currently out of reach of any existing algorithm.

To obtain problems of various difficulties the distance of agents from initial positions to their goals has been varied in the range 8, 16, 24, ..., 320.

For each distance 10 random instances were generated in which initial positions were selected randomly and then random walk has been performed until all the agents reach at least the given distance from its initial position.

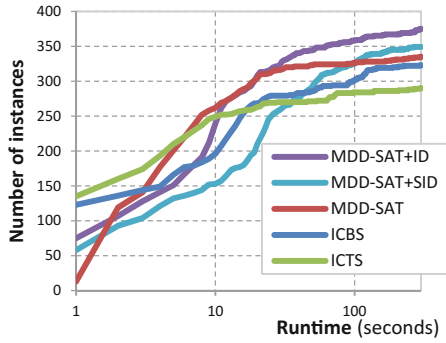
The hypothesis for large maps is that MDD-SAT+SID/ID should dominate generally with some expected advantage of ID which in fact is the same hypothesis as in the case of small grids because here we have only the low-medium density case. However, as there are important structural differences between the three tested maps which impact is hardly predictable. Intuitively, SID/ID should have been more beneficial in *ost003d* and *den520d* maps since in these maps there is more room to find alternative paths.

Results for the three Dragon Age maps are shown in Figs. 7, 8, and 9. Again the number of instances solved in the given runtime is shown. The difficulty (runtime) grows with the growing distance of agents from their goals in this setup.

It can be read from these results that MDD-SAT+ID tends to outperform MDD-SAT in more difficult instances. In these instances, the interaction among agents in non-trivial but on the other hand the interdependence among agents is tractable by ID.

Solved instances

Ost003d | 16 agents



Solved instances

Ost003d | 32 agents

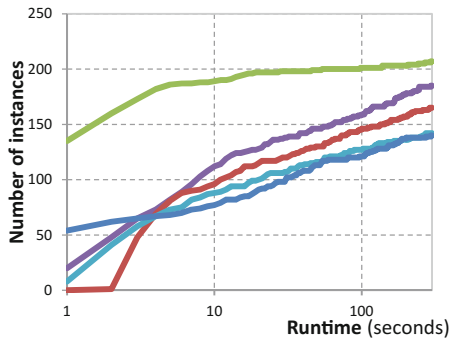
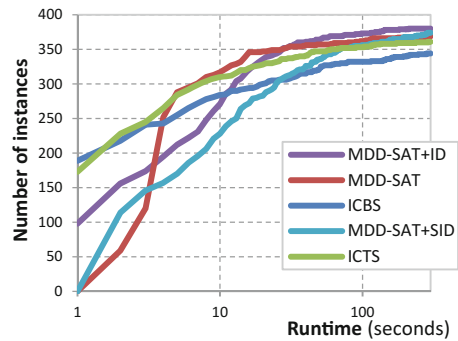


Fig. 7. Results of experiments on Dragon Age map *ost003d*. MDD-SAT+ID outperforms MDD-SAT in harder instances while MDD-SAT+SID performs worse than MDD-SAT. All MDD-SAT versions are dominated by ICTS.

Solved instances

Den520d | 16 agents



Solved instances

Den520d | 32 agents

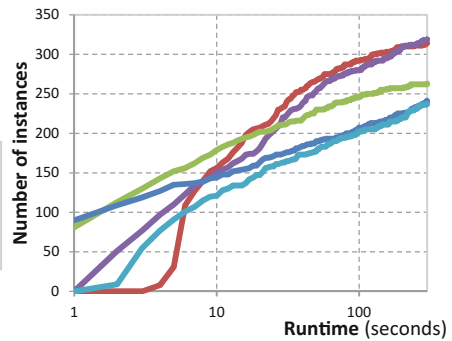


Fig. 8. Results of experiments on Dragon Age map *den520d*. ID brings minor benefit in harder instances while SID has merely a negative effect.

Surprising results have been obtained for MDD-SAT+SID which performed generally worse than MDD-SAT. SID hence was unsuccessful in independence detection enough to produce any performance benefit in MDD-SAT expect the case of very easy instances.

The intuitive hypothesis was not confirmed completely since surprisingly MDD-SAT is better than MDD-SAT+ID in easier instances of medium density category usually and the performance of MDD-SAT+SID remained behind expectation. Our initial intuitive hypothesis did estimate well the effort needed for merging groups that eventually represents a big overhead in case of large maps. Hence, MDD-SAT+ID can

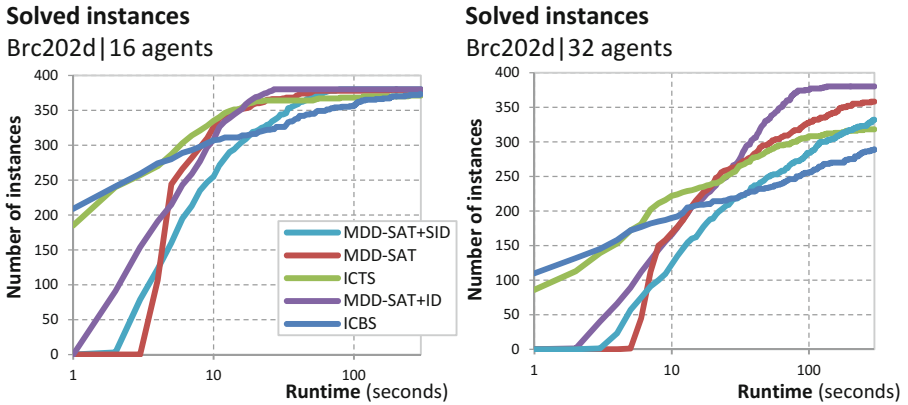


Fig. 9. Results of experiments on Dragon Age map *brc202d*. ID brings significant improvement in harder instances with 32 agents. SID again has rather a negative effect in MDD-SAT.

show its benefit after the difficulty of the formula representing the entire MAPF instance prevails over the difficulty of group merging.

Another surprising result was obtained in *brc202d* map where MDD-SAT+ID was a very clear winner in harder instances with 32 agents.

Moreover, we cannot say that SAT-based approach represented by MDD-SAT and MDD-SAT+SID/ID is a universal winner as there are cases where ICTS and ICBS dominate (*ost003d* with 32 agents is such an example).

7.3 Discussion

It can be generally observed that ID brings worthwhile improvement to MDD-SAT solver which by itself performs very well. The simple version of independence detection SID provides worse benefit than ID and in large instances its effect is even negative.

Experimental results indicate that there is a certain range of the density of agents though not precisely determined in our evaluation in which ID is beneficial while outside this range it cases an overhead.

The implementation of ID within the MDD-SAT+ID solver did not use any special reasoning about what groups of agents should be merged or not. The groups were processed in the ordering given by the original ordering of agents. We expect that more careful reasoning about merging can bring yet more improvements.

8 Conclusion

Integration of the existing technique of independence detection (ID) into the SAT-based approach to MAPF has been presented. Performed experimental evaluation shows that using ID in the SAT-based approach to MAPF has a significant performance

benefit. This can be especially observed in instances with medium density of agents where interactions among agents are non-trivial but there is still chance to find sufficiently many small independent groups of agents.

The new solver called MDD-SAT+ID is a combination of an existing SAT-based MAPF solver MDD-SAT and ID. We have shown that MDD-SAT+ID is a new state-of-the-art in the optimal SAT-based MAPF solving for the presented categories of the MAPF problem. Moreover, the new MDD-SAT+ID performs well also with respect to the best search based solvers like ICTS and ICBS though we cannot say that MDD-SAT+ID is a universal winner.

The future research of the presented topic will focus on the following aspects: (i) The classification of density of agents was intuitive only in the presented experimental evaluation. Hence the immediate future work is to develop concepts for more precise classification of the density and interaction among agents. (ii) ID is not beneficial across all instances and sometimes represents an overhead. Hence, having a more rigorous classification of the density of agents, we can develop a mechanism for automated deciding whether to use ID or not according to the classified density of agents. (iii) Currently we take groups of agents to be merged in the same order as they appear in the input. A more careful consideration of which groups to merge may lead to further performance improvements.

Acknowledgements. This paper is supported by the joint grant of the Israel Ministry of Science and the Czech Ministry of Education Youth and Sports number 8G15027, and Charles University under the SVV project number 260 333.

We would like thank anonymous reviewers for their constructive comments of [31] which helped us to prepare this extended version of the paper.

References

1. Audemard, G., Simon, L.: The Glucose SAT Solver (2013). <http://labri.fr/perso/lsimon/glucose/>. Accessed Oct 2016
2. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009), pp. 399–404. IJCAI (2009)
3. Bailleux, O., Boufkhad, Y.: Efficient CNF encoding of boolean cardinality constraints. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 108–122. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45193-8_8
4. Balint, A., Belov, A., Heule, M., Jarvisalo, M.: SAT 2015 competition (2015). <http://www.satcompetition.org/>. Accessed Oct 2016
5. van den Berg, J., Snoeyink, J., Lin, M.C., Manocha, D.: Centralized path planning for multiple robots: optimal decoupling into sequential plans. In: Proceedings of Robotics: Science and Systems V, University of Washington. The MIT Press (2010)
6. Biere, A., Heule, M., van Maaren, H., Walsh, T.: Handbook of Satisfiability. IOS Press, Amsterdam (2009)

7. Boyarski, E., Felner, A., Stern, R., Sharon, G., Tolpin, D., Betzalel, O., Shimony, S.: ICBS: improved conflict-based search algorithm for multi-agent pathfinding. In: Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015), pp. 740–746. IJCAI (2015)
8. Čáp, M., Novák, P., Vokřínek, J., Pěchouček, M.: Multi-agent RRT: sampling-based cooperative pathfinding. In: International conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2013), pp. 1263–1264. IFAAMAS (2013)
9. Erdem, E., Kisa, D.G., Öztok, U., Schüller, P.: A general formal framework for pathfinding problems with multiple agents. In: Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI 2013). AAAI Press (2013)
10. Huang, R., Chen, Y., Zhang, W.: A novel transition based encoding scheme for planning as satisfiability. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010). AAAI Press (2010)
11. Kautz, H., Selman, B.: Unifying SAT-based and graph-based planning. In: Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI 1999), pp. 318–325. Morgan Kaufmann (1999)
12. Kim, D., Hirayama, K., Park, G.-K.: Collision avoidance in multiple-ship situations by distributed local search. *J. Adv. Comput. Intell. Intell. Inform. (JACIII)* **18**(5), 839–848 (2014)
13. Kornhauser, D., Miller, G.L., Spirakis, P.G.: Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In: Proceedings of the 25th Annual Symposium on Foundations of Computer Science (FOCS 1984), pp. 241–250. IEEE Press (1984)
14. Ma, H., Koenig, S., Ayanian, N., Cohen, L., Hoenig W., Kumar, T.K.S., Uras, T., Xu, H., Tovey, C., Sharon, G.: Overview: generalizations of multi-agent path finding to real-world scenarios. In: IJCAI-16 Workshop on Multi-Agent Path Finding (WOMPF) (2016)
15. Michael, N., Fink, J., Kumar, V.: Cooperative manipulation and transportation with aerial robots. *Auton. Robot.* **30**(1), 73–86 (2011)
16. Ratner, D., Warmuth, M.K.: NxN puzzle and related relocation problems. *J. Symb. Comput.* **10**(2), 111–138 (1990)
17. Ryan, M.R.K.: Exploiting Subgraph structure in multi-robot path planning. *J. Artif. Intell. Res. (JAIR)* **31**, 497–542 (2008)
18. Ryan, M.R.K.: Constraint-based multi-robot path planning. In: Proceedings ICRA 2010, pp. 922–928. IEEE Press (2010)
19. Sharon, G., Stern, R., Goldenberg, M., Felner, A.: The increasing cost tree search for optimal multi-agent pathfinding. *Artif. Intell.* **195**, 470–495 (2013)
20. Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Conflict-based search for optimal multi-agent pathfinding. *Artif. Intell.* **219**, 40–66 (2015)
21. Marques-Silva, J., Lynce, I.: Towards robust CNF encodings of cardinality constraints. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 483–497. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74970-7_35
22. Silver, D.: Cooperative pathfinding. In: Proceedings of the 1st Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2005), pp. 117–122. AAAI Press (2005)
23. Sinz, C.: Towards an optimal CNF encoding of boolean cardinality constraints. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 827–831. Springer, Heidelberg (2005). https://doi.org/10.1007/11564751_73
24. Standley, T.: Finding optimal solutions to cooperative pathfinding problems. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010), pp. 173–178. AAAI Press (2010)

25. Standley, T., Korf, R.E.: Complete algorithms for cooperative pathfinding problems. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011), pp. 668–673. IJCAI (2011)
26. Sturtevant, N.R.: Benchmarks for grid-based pathfinding. *IEEE Trans. Comput. Intell. AI Games* **4**(2), 144–148 (2012)
27. Surynek, P.: A novel approach to path planning for multiple robots in biconnected graphs. In: Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA 2009), pp. 3613–3619. IEEE Press (2009)
28. Surynek, P.: An optimization variant of multi-robot path planning is intractable. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010), pp. 1261–1263. AAAI Press (2010)
29. Surynek, P.: Compact representations of cooperative path-finding as SAT based on matchings in bipartite graphs. In: Proceedings of the 26th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2014), pp. 875–882. IEEE Computer Society (2014)
30. Surynek, P., Felner, A., Stern, R., Boyarski, E.: Efficient SAT approach to multi-agent path finding under the sum of costs objective. In: Proceedings of 22nd European Conference on Artificial Intelligence (ECAI 2016), pp. 810–818. IOS Press (2016)
31. Surynek, P., Švancara, J., Felner, A., Boyarski, E.: Integration of independence detection into SAT-based optimal multi-agent path finding: a novel SAT-based optimal MAPF solver. In: Proceedings of the 9th International Conference on Agents and Artificial Intelligence (ICAART 2017). SciTe Press (2017)
32. Yu, J., LaValle, S.M.: Structure and intractability of optimal multirobot path planning on graphs. In: Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI 2013). AAAI Press (2013)
33. Yu, J., LaValle, S.M.: Planning optimal paths for multiple robots on graphs. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2013), pp. 3612–3617. IEEE Press (2013)
34. Wang, K.C., Botea, A.: Fast and memory-efficient multi-agent pathfinding. In: Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS 2008), pp. 380–387. AAAI Press (2008)
35. de Wilde, B., ter Mors, A., Witteveen, C.: Push and rotate: a complete multi-robot pathfinding algorithm. *J. Artif. Intell. Res. (JAIR)* **51**, 443–492 (2014)
36. Wilson, R.M.: Graph puzzles, homotopy, and the alternating group. *J. Comb. Theory Ser. B* **16**, 86–96 (1974)