# Multi-Agent Path Finding with Capacity Constraints

Pavel Surynek[1 [0000−0001−7200−0542]], T. K. Satish Kumar[2], and Sven Koenig[3]

[1] Faculty of Information Technology, Czech Technical University in Prague
Thákurova 9, 160 00 Praha 6, Czechia
pavel.surynek@fit.cvut.cz
http://users.fit.cvut.cz/surynek
[2] University of Southern California, Henry Salvatori Computer Science Center
941 Bloom Walk, Los Angeles, USA
skoenig@usc.edu
[3] University of Southern California, Information Sciences Institute
4676 Admiralty Way, Marina del Rey, USA
tkskwork@gmail.com

**Abstract.** In multi-agent path finding (MAPF) the task is to navigate agents from their starting positions to given individual goals. The problem takes place in an undirected graph whose vertices represent positions and edges define the topology. Agents can move to neighbor vertices across edges. In the standard MAPF, space occupation by agents is modeled by a capacity constraint that permits at most one agent per vertex. We suggest an extension of MAPF in this paper that permits more than one agent per vertex. Propositional satisfiability (SAT) models for these extensions of MAPF are studied. We focus on modeling capacity constraints in SAT-based formulations of MAPF and evaluation of performance of these models. We extend two existing SAT-based formulations with vertex capacity constraints: MDD-SAT and SMT-CBS where the former is an approach that builds the model in an eager way while the latter relies on lazy construction of the model.

**Keywords:** multi agent path finding, propositional satisfiability (SAT), capacity constraints, cardinality constraints

## 1 Introduction

In *multi-agent path finding* (MAPF) [9,18,19,21,24,28,33] the task is to navigate agents from given starting positions to given individual goals. The standard version of the problem takes place in undirected graph $G = (V, E)$ where agents from set $A = \{a_1, a_2, ..., a_k\}$ are placed in vertices with at most one agent per vertex. The initial configuration of agents in vertices of the graph can be written as $\alpha_0 : A \rightarrow V$ and similarly the goal configuration as $\alpha_+ : A \rightarrow V$. The task of navigating agents can be expressed as transforming the initial configuration of agents $\alpha_0 : A \rightarrow V$ into the goal configuration $\alpha_+ : A \rightarrow V$.

Movements of agents are instantaneous and are possible across edges into neighbor vertices assuming no other agent is entering the same target vertex. This formulation permits agents to enter vertices being simultaneously vacated by other agents. Trivial

case when a pair of agents swaps their positions across an edge is forbidden in the standard formulation. We note that different versions of MAPF exist where for example agents always move into vacant vertices [29]. We usually denote the configuration of agents at discrete time step $t$ as $\alpha_t : A \rightarrow V$. Non-conflicting movements transform configuration $\alpha_t$ *instantaneously* into next configuration $\alpha_{t+1}$. We do not consider what happens between $t$ and $t + 1$ in this discrete abstraction. Multiple agents can move at a time hence the MAPF problem is inherently parallel.

In order to reflect various aspects of real-life applications, variants of MAPF have been introduced such as those considering *kinematic constraints* [8], *large agents* [11], or *deadlines* [14] - see [13] for a more detailed survey.

Particularly in this work we are dealing with an extension of MAPF that generalizes the constraint of having at most one agent per vertex. There are many situations where we need to model nodes that could hold more than one agent at a time. Such situations include various graph-based evacuation models where for example nodes correspond to rooms in evacuated buildings [10] which naturally can hold more than one agent. Various spatial projections could also lead to having multiple agents per vertex such as upper projection of agents representing aerial drones where a single node corresponds to $x$,$y$-coordinate that could hold multiple agents at different $z$-coordinates [12]. Generally the need to consider nodes capable of containing multiple agents appears in modeling of multi-agent motion planning task at higher levels of granularity.

### 1.1 Contributions

The contribution of this paper consists in showing how to generalize existing *propositional satisfiability* (SAT) [4] models of MAPF for finding optimal plans with general capacity constraints that bound the number of agents in vertices. Two existing SAT-based models are generalized: MDD-SAT [32] that builds the propositional model in an *eager way* and SMT-CBS [30,31] that builds the model in a *lazy way* inspired by satisfiability modulo theories (SMT) [16].

The eager style of building the propositional model means that all constraints are posted into the model in advance. Such model is *complete*, that is, it is solvable (satisfiable) if and only if the instance being modeled is solvable. In contrast to this, the lazy style does not add all constraints at once and works with *incomplete* models. The incomplete model preserve only one-sided implication w.r.t. solvability: if the instance being modeled is solvable then the incomplete model is solvable (satisfiable).

The SMT-CBS algorithm iteratively refines the incomplete model towards the complete one by eliminating conflicts. That is, a candidate solution is extracted from the satisfied incomplete model. The candidate is checked for conflicts - whether any of the MAPF rules is violated - for example if a collision between agents occurred. If there are no conflicts, we are finished as the candidate is a valid solution of the input MAPF instance. If a conflict is detected, then a constraint that eliminates this particular conflict is added to the incomplete model resulting in a new model and the process is repeated. A new candidate solution is extracted from the new model etc. Eventually the process may end up with a complete model after eliminating all possible conflicts. However, we hope that the process finishes before constructing a complete model and we solve the instance with less effort.

In the presented generalization with capacity constraints we need to distinguish between the eager and lazy variant. The capacity constraint concerning given vertex $v$ bounding the number of agents that can simultaneously occupy $v$ by some integer constant say 2 can be literally translated into the requirement that no 3 distinct agents can occupy $v$ at the same time. Such a constraint can be directly posted in the eager variant: we either forbid all possible triples of agents in $v$ or post the corresponding *cardinality constraint* [3,23].

The situation is different in the lazy variant. To preserve the nature of the lazy approach we cannot post the capacity bound entirely as conceptually at the low level we are informed only about a particular MAPF rule violation, say for example agents $a_1$, $a_5$ and $a_8$ occurred simultaneously in $v$ which is forbidden in given MAPF. The information that there is a capacity constraint on $v$ bounding the number of agents in $v$ by 2 may even not be accessible at the low level. Hence we can forbid simultaneous occurrence of only the given triple of agents, $a_1$, $a_5$ and $a_8$ in this case.

The paper is organized as follows. We first introduce the standard multi-agent path finding problem formally including commonly used objectives. Then we introduce two major existing SAT-based encodings. On top of this, we show how to extend these encodings with vertex capacities. Finally we evaluate extended models on standard benchmarks including open grids and large game maps.

## 2 Formal Definition of MAPF and Vertex Capacities

The *Multi-agent path finding* (MAPF) problem [24,18] consists of an undirected graph $G = (V, E)$ and a set of agents $A = \{a_1, a_2, ..., a_k\}$ such that $|A| \leq |V|$. Each agent is placed in a vertex so that at most one agent resides in each vertex. The placement of agents is denoted $\alpha : A \to V$. Next we are given initial configuration of agents $\alpha_0$ and goal configuration $\alpha_+$.

At each time step an agent can either *move* to an adjacent vertex or *wait* in its current vertex. 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 nor cross the same edge in opposite directions simultaneously. The following definition formalizes the commonly used movement rule in MAPF.

**Definition 1. Valid movement in MAPF.** *Configuration $\alpha'$ results from $\alpha$ if and only if the following conditions hold:*

*(i)* $\alpha(a) = \alpha'(a)$ *or* $\{\alpha(a), \alpha'(a)\} \in E$ *for all* $a \in A$ *(agents wait or move along edges);*
*(ii) for all* $a \in A$ *it holds* $\alpha(a) \neq \alpha'(a) \Rightarrow \neg(\exists b \in A)(\alpha(b) = \alpha'(a) \wedge \alpha'(b) = \alpha(a))$ *(no two agents cross an edge in opposite directions);*
*(iii) and for all* $a, a' \in A$ *it holds that* $a \neq a' \Rightarrow \alpha'(a) \neq \alpha'(a')$ *(no two agents share a vertex in the next configuration).*

Solving the MAPF instance is to find a sequence of configurations $[\alpha_0, \alpha_1, ..., \alpha_\mu]$ such that $\alpha_{i+1}$ results using valid movements from $\alpha_i$ for $i = 1, 2, ..., \mu - 1$, and $\alpha_\mu = \alpha_+$.

A version of MAPF with *vertex capacities* generalizes the above definition by adding capacity function $c : V \rightarrow \mathbb{Z}_+$ that assigns each vertex a positive integer capacity. The interpretation is that a vertex $v$ can hold up to the specified number of agents $c(v)$ at any time-step.

The definition of the valid movement will change only in point (iii) where instead of permitting at most one agent per vertex we allow any number of agents not exceeding the capacity of the vertex:

**Definition 2. Vertex capacities in MAPF.**

*(iii') for all $v \in V$ it holds that $|a \mid \alpha'(a) = v| \leq c(v)$ (the number of agents in each vertex does not exceed the capacity in the next configuration).*

Generalized vertex capacities relax the problem in fact as illustrated in Figure 1. Intuitively, capacities greater than one induce additional parking place in the environment which we hypothetise makes the problem easier to solve.

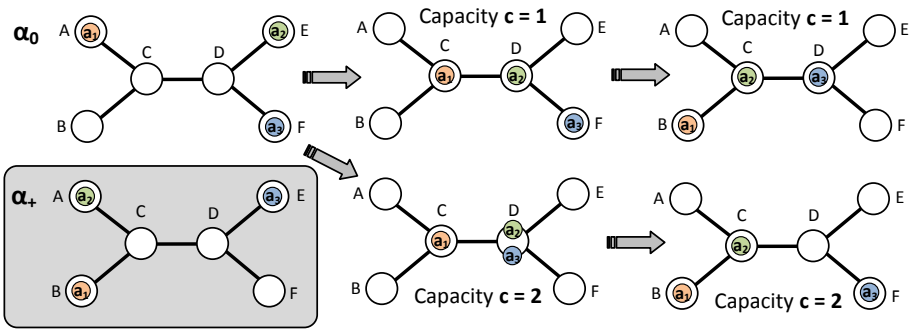


Fig. 1: Illustration of the standard MAPF ($c = 1$) and MAPF with generalized vertex capacity (uniform capacity $c = 2$ us used). With $c = 2$ two agents $a_2$ and $a_3$ can both enter vertex D. In contrast to this, $a_3$ must wait in vertex F in the standard MAPF.

### 2.1 Common Objectives in MAPF

We address here optimal MAPF solving hence we need to introduce objective functions more formally. In case of *makespan* [29] we just need to minimize $\mu$ in the aforementioned solution sequence. For introducing the *sum-of-costs* objective [7,26,21,19] we need the following notation:

**Definition 3. Sum-of-costs objective** *is the summation, over all agents, of the number of time steps required to reach the goal vertex. Denoted $\xi$, where $\xi = \sum_{i=1}^{k} \xi(path(a_i))$, where $\xi(path(a_i))$ is an individual path cost of agent $a_i$ connecting $\alpha_0(a_i)$ and $\alpha_+(a_i)$ calculated as the number of edge traversals and wait actions.* [4]

[4] The notation $path(a_i)$ refers to path in the form of a sequence of vertices and edges connecting $\alpha_0(a_i)$ and $\alpha_+(a_i)$ while $\xi$ assigns the cost to a given path.

Observe that in the sum-of-costs we accumulate the cost of wait actions for agents not yet reaching their goal vertices. For the sake of brevity we focus here on the sum-of-costs, but we note that all new concepts can be introduced for different cumulative objectives like the makespan.[5]

We note that finding a solution that is optimal (minimal) with respect to the sum-of-costs objective is NP-hard [17]. The same result holds for the variant with capacities as it is a straight generalization of the standard MAPF version.

## 3 Related Work

Let us now recall existing SAT-based optimal MAPF solvers. We here focus on aspects important for introducing capacities. We recall MDD-SAT, the sum-of-costs optimal solver based on *eager* SAT encoding [32], and SMT-CBS [31], the most recent SAT-based, or more precisely SMT-based, algorithm using lazy *encoding*.

### 3.1 SAT-based Approach

The idea behind the SAT-based approach is to construct a propositional formula $\mathcal{F}(\xi)$ such that it is satisfiable if and only if a solution of a given MAPF of sum-of-costs $\xi$ exists [29]. Moreover, the approach is constructive; that is, $\mathcal{F}(\xi)$ exactly reflects the MAPF instance and if satisfiable, solution of MAPF can be reconstructed from satisfying assignment of the formula. We say $\mathcal{F}(\xi)$ to be a *complete propositional model* of MAPF.

**Definition 4. (complete propositional model).** *Propositional formula $\mathcal{F}(\xi)$ is a* complete propositional model *of MAPF $\Sigma$ if the following condition holds:*

$$\mathcal{F}(\xi) \textit{ is satisfiable} \Leftrightarrow \Sigma \textit{ has a solution of sum-of-costs } \xi.$$

Being able to construct such formula $\mathcal{F}$ one can obtain optimal MAPF solution by checking satisfiability of $\mathcal{F}(0)$, $\mathcal{F}(1)$, $\mathcal{F}(2)$,... until the first satisfiable $\mathcal{F}(\xi)$ is met. This is possible due to monotonicity of MAPF solvability with respect to increasing values of common cumulative objectives like the sum-of-costs. In practice it is however impractical to start at 0; lower bound estimation is used instead - sum of lengths of shortest paths can be used in the case of sum-of-costs. The framework of SAT-based solving is shown in pseudo-code in Algorithm 1.

### 3.2 Details of the MDD-SAT Encoding

Construction of $\mathcal{F}(\xi)$ as used in the MDD-SAT solver relies on time expansion of underlying graph $G$. Having $\xi$, the basic variant of time expansion determines the maximum number of time steps $\mu$ (*makespan*) such that every possible solution of the given MAPF with the sum-of-costs less than or equal to $\xi$ fits within $\mu$ timesteps. Given $\xi$ we

[5] Dealing with objectives is out of scope of this paper. We refer the reader to [32] for more detailed discussion.

can calculate $\mu$ as $max_{i=1}^{k}\{\xi_0(a_i)\} + \xi - \xi_0$ where $\xi_0(a_1)$ is the length of the shortest path connecting $\alpha_0(a_i)$ and $\alpha_+(a_i)$; $\xi_0 = \sum_{i=1}^{k} \xi_0(a_i)$. The detailed justification of this equation is given in [32].

Time expansion itself makes copies of vertices $V$ for each timestep $t = 0, 1, 2, ..., \mu$. That is, we have vertices $v^t$ for each $v \in V$ and time step $t$. Edges from $G$ are converted to directed edges interconnecting timesteps in the time expansion. Directed edges $(u^t, v^{t+1})$ are introduced for $t = 1, 2, ..., \mu - 1$ whenever there is $\{u, v\} \in E$. Wait actions are modeled by introducing edges $(u^t, t^{t+1})$. A directed path in the time expansion corresponds to trajectory of an agent in time. Hence the modeling task now consists in construction of a formula in which satisfying assignments correspond to directed paths from $\alpha_0^0(a_i)$ to $\alpha_+^\mu(a_i)$ in the time expansion.

Assume that we have time expansion $TEG_i = (V_i, E_i)$ for agent $a_i$. Propositional variable $\mathcal{X}_v^t(a_j)$ is introduced for every vertex $v^t$ in $V_i$. The semantics of $\mathcal{X}_v^t(a_i)$ is that it is $TRUE$ if and only if agent $a_i$ resides in $v$ at time step $t$. Similarly we introduce $\mathcal{E}_u, v^t(a_i)$ for every directed edge $(u^t, v^{t+1})$ in $E_i$. Analogously the meaning of $\mathcal{E}_{u,v}^t(a_i)$ is that is $TRUE$ if and only if agent $a_i$ traverses edge $\{u, v\}$ between time steps $t$ and $t + 1$.

Constraints are added so that truth assignment are restricted to those that correspond to valid solutions of a given MAPF. Added constraints together ensure that $\mathcal{F}(\xi)$ is a *complete propositional model* for given MAPF.

We here illustrate the model by showing few representative constraints. We omit here constraints that concern objective function. For the detailed list of constraints we again refer the reader to [32].

Collisions among agents are eliminated by the following constraint for every $v \in V$ and timestep $t$ expressed on top of $\mathcal{X}_v^t(a_i)$ variables:

$$\sum_{a_i \in A \mid v^t \in V_i} \mathcal{X}_v^t(a_i) \leq 1 \tag{1}$$

There are various ways how to translate the constraint using propositional clauses. One efficient way is to introduce $\neg\mathcal{X}_v^t(a_i) \vee \neg\mathcal{X}_v^t(a_j)$ for all possible pairs of $a_i$ and $a_j$.

Next, there is a constraint stating that if agent $a_i$ appears in vertex $u$ at time step $t$ then it has to leave through exactly one edge $(u^t, v^{t+1})$. This can be established by following constraints:

$$\mathcal{X}_u^t(a_i) \Rightarrow \bigvee_{(u^t, v^{t+1}) \in E_i} \mathcal{E}_{u,v}^t(a_i), \tag{2}$$

$$\sum_{v^{t+1} \mid (u^t, v^{t+1}) \in E_i} \mathcal{E}_{u,v}^t(a_i) \leq 1 \tag{3}$$

Similarly, the target vertex of any movement except wait action must be empty. This is ensured by the following constraint for every $(u^t, v^{t+1}) \in E_i$:

$$\mathcal{E}_{u,v}^t(a_i) \Rightarrow \bigwedge_{a_j \in A \mid a_j \neq a_i \wedge v^t \in V_j} \neg\mathcal{X}_v^t(a_j) \tag{4}$$

**Algorithm 1:** Framework of SAT-based MAPF solving

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

Other constraints ensure that truth assignments to variables per individual agents form paths. That is if agent $a_i$ enters an edge it must leave the edge at the next time step.

$$\mathcal{E}^t_{u,v}(a_i) \Rightarrow \mathcal{X}^t_v(a_i) \wedge \mathcal{X}^{t+1}_v(a_i) \tag{5}$$

A common measure how to reduce the number of decision variables derived from the time expansion is the use of *multi-value decision diagrams* (MDDs) [21]. The basic observation that holds for MAPF is that an agent can reach vertices in the distance $d$ (distance of a vertex is measured as the length of the shortest path) from the current position of the agent no earlier than in the $d$-th time step. Analogical observation can be made with respect to the distance from the goal position.

Above observations can be utilized when making the time expansion of $G$. For a given agent, we do not need to consider all vertices at time step $t$ but only those that are reachable in $t$ timesteps from the initial position and that ensure that the goal can be reached in the remaining $\mu - t$ timesteps.

### 3.3 Resolving Conflicts Lazily in SMT-CBS

SMT-CBS is inspired by the search-based algorithm CBS [22,20] that uses the idea of resolving conflicts lazily; that is, a solution of MAPF instance is not searched against the complete set of movement constraints that forbids collisions between agents but with respect to initially empty set of collision forbidding constraints that gradually grows as new conflicts appear. SMT-CBS follows the high-level framework of CBS but rephrases the process into propositional satisfiability in a similar way as done in formula satisfiability testing in the *satisfiability modulo theory* paradigm [16,15,5].

The high-level of CBS searches a *constraint tree* (CT) using a priority queue in breadth first manner. CT is a binary tree where each node $N$ contains a set of collision avoidance constraints $N.constraints$ - a set of triples $(a_i, v, t)$ forbidding occurrence of agent $a_i$ in vertex $v$ at time step $t$, a solution $N.paths$ - a set of $k$ paths for individual agents, and the total cost $N.\xi$ of the current solution.

The low-level process in CBS associated with node $N$ searches paths for individual agents with respect to set of constraints $N.constraints$. For a given agent $a_i$, this is a standard single source shortest path search from $\alpha_0(a_i)$ to $\alpha_+(a_i)$ that avoids a set of vertices $\{v \in V | (a_i, v, t) \in N.constraints\}$ whenever working at time step $t$. For details see [19].

CBS stores nodes of CT into priority queue OPEN sorted according to the ascending costs of solutions. At each step CBS takes node $N$ with the lowest cost from OPEN and checks if $N.paths$ represent paths that are valid with respect to MAPF movements rules - that is, $N.paths$ are checked for collisions. If there is no collision, the algorithms returns valid MAPF solution $N.paths$. Otherwise the search branches by creating a new pair of nodes in CT - successors of $N$. Assume that a collision occurred between agents $a_i$ and $a_j$ in vertex $v$ at time step $t$. This collision can be avoided if either agent $a_i$ or agent $a_j$ does not reside in $v$ at timestep $t$. These two options correspond to new successor nodes of $N$: $N_1$ and $N_2$ that inherit the set of conflicts from $N$ as follows: $N_1.conflicts = N.conflicts \cup \{(a_i, v, t)\}$ and $N_2.conflicts = N.conflicts \cup \{(a_j, v, t)\}$. $N_1.paths$ and $N_1.paths$ inherit paths from $N.paths$ except those for agents $a_i$ and $a_j$ respectively. Paths for $a_i$ and $a_j$ are recalculated with respect to extended sets of conflicts $N_1.conflicts$ and $N_2.conflicts$ respectively and new costs for both agents $N_1.\xi$ and $N_2.\xi$ are determined. After this, $N_1$ and $N_2$ are inserted into the priority queue OPEN.

SMT-CBS compresses CT into a single branch in which the propositional model taken from MDD-SAT is iteratively refined. The high-level branching from CBS is deferred to the low level of SAT solving. In the MDD-SAT encoding collision avoidance constraints are omitted initially, only constraints ensuring that assignments form valid paths interconnecting starting positions with goals are preserved. This will result in an *incomplete propositional model* denoted $\mathcal{H}(\xi)$. The important component of SMT-CBS is a paths validation procedure that reports back the set of conflicts found in the current solution that are used for making model refinements.

SMT-CBS is shown in pseudo-code as Algorithm 2. The algorithm is divided into two procedures: SMT-CBS representing the main loop and SMT-CBS-Fixed solving the input MAPF for fixed cost $\xi$. The major difference from the standard CBS is that there is no branching at the high-level. The high-level SMT-CBS roughly correspond to the main loop of MDD-SAT. The set of conflicts is iteratively collected during the entire execution of the algorithm. Procedure *encode* from MDD-SAT is replaced with *encode-Basic* that produces encoding that ignores specific movement rules (collisions between agents) but in contrast to *encode* it encodes collected conflicts into $\mathcal{H}(\xi)$.

The conflict resolution in the standard CBS implemented as high-level branching is here represented by refinement of $\mathcal{H}(\xi)$ with disjunction (line 20). The presented SMT-CBS can eventually build the same formula as MDD-SAT but this is done lazily.

## 4 Handling Capacity Constraints in MAPF

To adapt the SAT-based approach for MAPF with capacities we need minor modifications only in both MDD-SAT and SMT-CBS. However in each algorithm the integration of capacity constraints is profoundly different. While in MDD-SAT we integrate

**Algorithm 2:** SMT-CBS algorithm for solving MAPF

**SMT-CBS** ($\Sigma = (G = (V, E), A, \alpha_0, \alpha_+)$)
  $conflicts \leftarrow \emptyset$
  $paths \leftarrow \{path^*(a_i)$ a shortest path from $\alpha_0(a_i)$ to $\alpha_+(a_i) | i = 1, 2, ..., k\}$
  $\xi \leftarrow \sum_{i=1}^{k} \xi(paths(a_i))$
  **while** $TRUE$ **do**
    $(paths, conflicts) \leftarrow$ SMT-CBS-Fixed($conflicts, \xi, \Sigma$)
    **if** $paths \neq$ *UNSAT* **then**
      **return** $paths$
    $\xi \leftarrow \xi + 1$
**SMT-CBS-Fixed**($conflicts, \xi, \Sigma$)
  $\mathcal{H}(\xi) \leftarrow$ encode-Basic($conflicts, \xi, \Sigma$)
  **while** $TRUE$ **do**
    $assignment \leftarrow$ consult-SAT-Solver($\mathcal{H}(\xi)$)
    **if** $assignment \neq UNSAT$ **then**
      $paths \leftarrow$ extract-Solution($assignment$)
      $collisions \leftarrow$ validate($paths$)
      **if** $collisions = \emptyset$ **then**
        **return** $(paths, conflicts)$
      **for** *each* $(a_i, a_j, v, t) \in collisions$ **do**
        $\mathcal{H}(\xi) \leftarrow \mathcal{H}(\xi) \cup \{\neg \mathcal{X}_v^t(a_i) \vee \neg \mathcal{X}_v^t(a_j)\}$
        $conflicts \leftarrow conflicts \cup \{[(a_i, v, t), (a_j, v, t)]\}$
    **return** *(UNSAT,conflicts)*

capacity constraints eagerly in the line with the original design of the algorithm (that is, the constraint in introduced as a whole), in SMT-CBS we integrate capacity constraint lazily which means part by part as new conflicts appear.

### 4.1 Details of the Encoding with Capacities

We need only a small modification of the MDD-SAT encoding to handle vertex capacities. We need to replace constraint (1) with the following constraint that is again posted for every vertex $v$ and time step $t$:

$$\sum_{a_i \in A \mid v^t \in V_i} \mathcal{X}_v^t(a_i) \leq c(v) \tag{6}$$

Unlike in the standard MAPF we need here a more sophisticated translation of the constraint to propositional clauses. Using the approach of forbidding individual $c(v) + 1$-tuples can be highly inefficient especially in cases when $c(v)$ is large. Therefore we use cardinality constraints encodings commonly used in SAT [3,23,25]. Generally the cardinality constraint over set of propositional variables $\{\mathcal{X}_1, \mathcal{X}_2, ..., \mathcal{X}_n\}$ permits at most a specified number of variables from the set to be $TRUE$, denoted $\leq_k \{\mathcal{X}_1, \mathcal{X}_2, ..., \mathcal{X}_n\}$ means that at most k variables from the set can be $TRUE$.

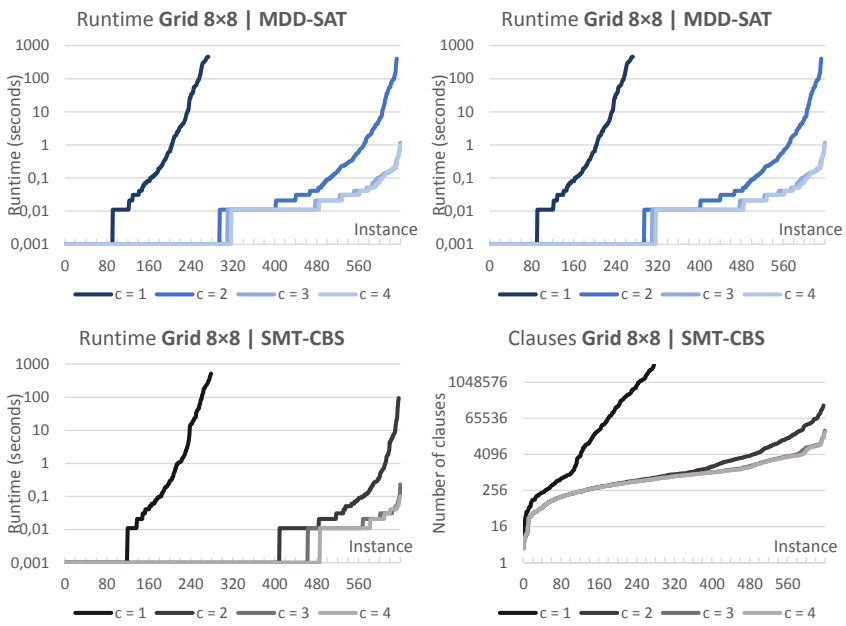


Fig. 2: Sorted runtimes and the number of clauses on the $8 \times 8$ grid. MDD-SAT and SMT-CBS are compared.

In our case of MAPF with capacities we need to introduce following cardinality constraints for every vertex $v$ and time step $t$. The practical implementation of cardinaliy constraints is done through encoding adder circuits inside the formula [23].

$$\leq_{c(v)}\{\mathcal{X}_v^t(a_i) \mid a_i \in A \wedge v^t \in V_i\} \tag{7}$$

### 4.2 Capacities in SMT-CBS

Capacities in SMT-CBS are resolved lazily as well. That is, the capacity constraint is not posted entirely as a cardinality constraint but instead individual sets of agents that violate the capacity are forbidden one by one as they appear. That is for example if a generalized conflict occurs with agents $a_{i_1}$, $a_{i_2}$, ..., $a_{i_m}$ in vertex $v$ (in other words if $m > c(v)$) we post a conflict elimination clause concerning the colliding set of agents: $\neg\mathcal{X}_v^t(a_{i_1}) \vee \neg\mathcal{X}_v^t(a_{i_2}) \vee ... \vee \neg\mathcal{X}_v^t(a_{i_m})$.

Hence in the SMT-CBS algorithm we modify only lines 20 and 21 that handle generalized vertex conflicts. Also we need to modify the validation procedure called at line 15 to reflect generalized vertex capacities.

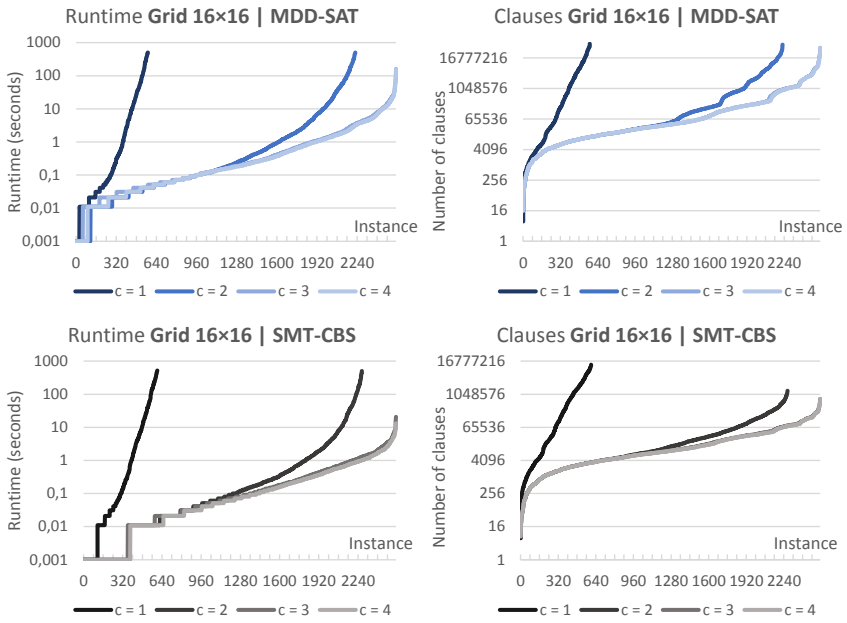


Fig. 3: Sorted runtimes and the number of clauses on the $16 \times 16$ grid. MDD-SAT and SMT-CBS are compared.

## 5 Experimental Evaluation

To evaluate the performance of capacity handling in context of SAT-based algorithms we performed an extensive evaluation on both standard synthetic benchmarks [6,21] and large maps from games [27].

### 5.1 Setup of Experiments and Benchmarks

We took the existing implementations of MDD-SAT and SMT-CBS written in C++. Both implementations are built on top of the Glucose 4 SAT solver [1,2]. In the implementations we modified the capacity constraint from the original *at-most-one* to generalized variants as mentioned above. All experiments were run on a Ryzen 7 CPU 3.0 Ghz under Kubuntu linux 16 with 16GB RAM. The timeout in all experiments was set to 500 seconds. Presented are only results finished within this time limit.

The second part of experimental evaluation took place on large 4-connected maps taken from *Dragon Age* [19,27]. We took three structurally different maps focusing on various aspects such as narrow corridors, large almost isolated rooms, or topologically complex open space. In contrast to small instances, these were only sparsely populated with agents. Initial and goal configuration were generated at random again. Up to 80 agents were used in these instances and uniform capacities of 1, 2, and 3. We measured the runtime on large maps.

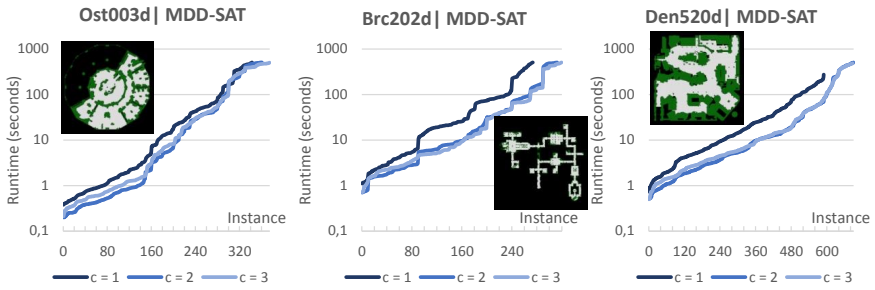


Fig. 4: Sorted runtimes of MDD-SAT on `ost003d`, `brc202d`, and `den520d` maps.

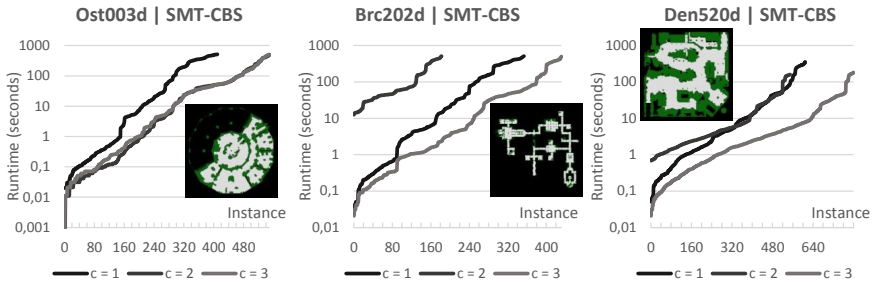


Fig. 5: Sorted runtimes of MDD-SAT on `ost003d`, `brc202d`, and `den520d` maps.

### 5.2 Results on Small Grids

Results obtained for small open grids are presented in Figures 2 and 3. We can see that in comparison with the standard MAPF capacities bring significant reduction of the difficulty of instances. This difference can be seen in both MDD-SAT and SMT-CBS. The starkest performance difference is between $c = 1$ and $c = 2$. The least performance difference is between $c = 3$ and $c = 4$. The similar picture can be seen in for the number of clauses.

### 5.3 Results on Large Maps

Results for large game maps are shown in Figures 4 and 5. A different picture can be seen here. Adding capacities does not cause any significant simplification except the `brc202d` map which consists of narrow corridors. The interpretation is that adding extra parking place via capacities may lead to simplification only when it is not available normally. Otherwise generalized capacity constraints lead to harder instances.

## 6 Discussion and Conclusion

We introduced multi-agent path finding problem with vertex capacity constraints. We modified two existing state-of-the-art SAT-based optimal MAPF solvers to reflect vertex

capacities, the MDD-SAT solver using the *eager* encoding and the SMT-CBS solver using the *lazy* encoding.

In both solvers we observed that adding an extra room by increasing the capacity of vertices dramatically reduces the difficulty of instances. However adding further capacity has less significant effect. In large maps using higher capacities even lead to performance degradation which we attribute to more complex constraints.

In the future work we would like to apply the MAPF formulation with capacities in the real-life multi-agent problems being solved by hierarchical approaches.

## References


1. Audemard, G., Lagniez, J., Simon, L.: Improving glucose for incremental SAT solving with assumptions: Application to MUS extraction. In: SAT. pp. 309–317 (2013)
2. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: IJCAI. pp. 399–404 (2009)
3. Bailleux, O., Boufkhad, Y.: Efficient CNF encoding of boolean cardinality constraints. In: CP. pp. 108–122 (2003)
4. Biere, A., Biere, A., Heule, M., van Maaren, H., Walsh, T.: Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications. IOS Press (2009)
5. Bofill, M., Palahí, M., Suy, J., Villaret, M.: Solving constraint satisfaction problems with SAT modulo theories. Constraints **17**(3), 273–303 (2012)
6. 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: IJCAI. pp. 740–746 (2015)
7. Dresner, K., Stone, P.: A multiagent approach to autonomous intersection management. JAIR **31**, 591–656 (2008)
8. Hönig, W., Kumar, T.K.S., Cohen, L., Ma, H., Xu, H., Ayanian, N., Koenig, S.: Summary: Multi-agent path finding with kinematic constraints. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017. pp. 4869–4873 (2017)
9. Kornhauser, D., Miller, G.L., Spirakis, P.G.: Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In: FOCS, 1984. pp. 241–250 (1984)
10. Kumar, K., Romanski, J., Hentenryck, P.V.: Optimizing infrastructure enhancements for evacuation planning. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. pp. 3864–3870. AAAI Press (2016)
11. Li, J., Surynek, P., Felner, A., Ma, H., Koenig, S.: Multi-agent path finding for large agents. In: AAAI. pp. 7627–7634. AAAI Press (2019)
12. Liu, S., Mohta, K., Atanasov, N., Kumar, V.: Towards search-based motion planning for micro aerial vehicles. CoRR **abs/1810.03071** (2018), http://arxiv.org/abs/1810.03071
13. Ma, H., Koenig, S., Ayanian, N., Cohen, L., Hönig, W., Kumar, T.K.S., Uras, T., Xu, H., Tovey, C.A., Sharon, G.: Overview: Generalizations of multi-agent path finding to real-world scenarios. CoRR **abs/1702.05515** (2017), http://arxiv.org/abs/1702.05515
14. Ma, H., Wagner, G., Felner, A., Li, J., Kumar, T.K.S., Koenig, S.: Multi-agent path finding with deadlines. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden. pp. 417–423 (2018)
15. Nieuwenhuis, R.: SAT modulo theories: Getting the best of SAT and global constraint filtering. In: Principles and Practice of Constraint Programming - CP 2010 - 16th International Conference, CP 2010, St. Andrews, Scotland, UK, September 6-10, 2010. Proceedings. pp. 1–2 (2010)

16. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll($T$). J. ACM **53**(6), 937–977 (2006)
17. Ratner, D., Warmuth, M.K.: Finding a shortest solution for the N x N extension of the 15-puzzle is intractable. In: AAAI. pp. 168–172 (1986)
18. Ryan, M.R.K.: Exploiting subgraph structure in multi-robot path planning. J. Artif. Intell. Res. (JAIR) **31**, 497–542 (2008)
19. Sharon, G., Stern, R., Felner, A., Sturtevant, N.: Conflict-based search for optimal multi-agent pathfinding. Artif. Intell. **219**, 40–66 (2015)
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. Sharon, G., Stern, R., Goldenberg, M., Felner, A.: The increasing cost tree search for optimal multi-agent pathfinding. Artif. Intell. **195**, 470–495 (2013)
22. Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Conflict-based search for optimal multi-agent path finding. In: AAAI (2012)
23. Silva, J., Lynce, I.: Towards robust CNF encodings of cardinality constraints. In: CP. pp. 483–497 (2007)
24. Silver, D.: Cooperative pathfinding. In: AIIDE. pp. 117–122 (2005)
25. Sinz, C.: Towards an optimal CNF encoding of boolean cardinality constraints. In: CP (2005)
26. Standley, T.: Finding optimal solutions to cooperative pathfinding problems. In: AAAI. pp. 173–178 (2010)
27. Sturtevant, N.R.: Benchmarks for grid-based pathfinding. Computational Intelligence and AI in Games **4**(2), 144–148 (2012)
28. Surynek, P.: A novel approach to path planning for multiple robots in bi-connected graphs. In: ICRA 2009. pp. 3613–3619 (2009)
29. Surynek, P.: Time-expanded graph-based propositional encodings for makespan-optimal solving of cooperative path finding problems. Ann. Math. Artif. Intell. **81**(3-4), 329–375 (2017)
30. Surynek, P.: Lazy modeling of variants of token swapping problem and multi-agent path finding through combination of satisfiability modulo theories and conflict-based search. CoRR **abs/1809.05959** (2018), http://arxiv.org/abs/1809.05959
31. Surynek, P.: Unifying search-based and compilation-based approaches to multi-agent path finding through satisfiability modulo theories. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI 2019. pp. 1177–1183. ijcai.org (2019)
32. Surynek, P., Felner, A., Stern, R., Boyarski, E.: Efficient SAT approach to multi-agent path finding under the sum of costs objective. In: ECAI. pp. 810–818 (2016)
33. Wang, K., Botea, A.: MAPP: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. JAIR **42**, 55–90 (2011)