A Scheduling-Based Approach to Multi-Agent Path Finding with Weighted and Capacitated Arcs

Roman Barták
Charles University
Prague, Czech Republic
bartak@ktiml.mff.cuni.cz

Jiří Švancara
Charles University
Prague, Czech Republic
Jiri.Svancara@mff.cuni.cz

Marek Vlk
Charles University & Czech Technical University in Prague
Prague, Czech Republic
vlk@ktiml.mff.cuni.cz

ABSTRACT

Multi-agent path finding (MAPF) deals with the problem of finding a collision-free path for a set of agents. The agents are located at nodes of a directed graph, they can move over the arcs, and each agent has its own destination node. It is not possible for two agents to be at the same node at the same time. The usual setting is that each arc has length one so at any time step, each agent either stays in the node, where it is, or moves to one of its neighboring nodes.

This paper suggests to model the MAPF problem using scheduling techniques, namely, nodes and arcs are seen as resources. The concept of optional activities is used to model which nodes and arcs an agent will visit. We first describe a model, where each agent can visit each node at most once. Then, we extend the model to allow agents re-visiting the nodes.

The major motivation for the scheduling model of MAPF is its capability to naturally include other constraints. We will study particularly the problems, where the capacity of arcs can be greater than one (more agents can use the same arc at the same time), and the lengths of arcs can be greater than one (moving between different pairs of nodes takes different times). These extensions make the model closer to reality than the original MAPF formulation. We compare the efficiency of models experimentally.

KEYWORDS

path finding; multiple agents; capacity constraints; scheduling; constraint programming

ACM Reference Format:
Roman Barták, Jiří Švancara, and Marek Vlk. 2018. A Scheduling-Based Approach to Multi-Agent Path Finding with Weighted and Capacitated Arcs. In *Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), Stockholm, Sweden, July 10–15, 2018,* IFAAMAS, 9 pages.

1 INTRODUCTION

There are many practical situations, where a set of agents (robots, cars, etc.) is moving in a shared environment, while each agent is heading for its desired goal position. The environment is usually represented by a graph, where agents can occupy nodes and move along the arcs [12]. The agents are moving cooperatively to avoid collisions and unsolvable congestions. The quality of the sequence of steps leading each agent to the goal position can be measured by some cost function such as makespan.

The problem described above is known as multi-agent path finding (MAPF) [6]. Some examples, where the MAPF problem is useful, include traffic optimization [5, 10], navigation [18], movement in computer games [20], etc.

The state of the art algorithms for the MAPF problem assume that all of the arc lengths are identical and that each node and arc can be occupied by at most one agent at any time. These limitations on the solved problem do not correspond to the reality in many cases. For example, some roads have a larger capacity than others. In this paper, we add new attributes to the problem specification that bring it closer to the real world.

In particular, we model the MAPF problem in the Constraint Programming (CP) formalism borrowing ideas from scheduling and routing problems. We see the nodes and arcs as resources with limited capacity, which equals one in the typical MAPF setting but can be larger in some applications. We use the concept of optional (alternative) activities [8] with specialized global constraints, such as *NoOverlap*, modeling resources. The motivation is supporting richer (in comparison to traditional MAPF) temporal and capacity constraints.

After formally introducing the classical MAPF problem, we will propose a core scheduling model that allows each agent to visit each node at most once (a single-layer model). We will then extend this model to support multiple visits of each node (a multi-layer model) that complies with the classical MAPF formulation. After that, we will generalize the model to support arcs of different lengths and capacities. While the extension of the proposed scheduling-based model is straightforward for this extended setting, we will also present the extension of the classical SAT-based model. The paper is concluded by experimental comparison of scheduling-based and SAT-based models with the goal to find how particular problem attributes influence efficiency of various models.

2 BACKGROUND ON MULTI-AGENT PATH FINDING

The MAPF problem is formulated by a graph and a set of agents sitting at certain nodes. The task is to find paths for agents from their origin nodes to their destination nodes while satisfying some constraints, namely, no two agents meet at the same node at the same time, and no two agents swap their positions at one step.

Formally we can define an instance of MAPF as ordered 4-tuple $(G, A, orig, dest)$, where $G = (V, E)$ is a directed graph and A is a set of agents. Functions $orig : A \longrightarrow V$ and $dest : A \longrightarrow V$ describe origin and destination nodes of an agent. For each agent $a \in A$, we denote $orig(a) \in V$ the origin location (node) of the agent and $dest(a) \in V$ its destination node.

Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. André, S. Koenig (eds.), July 10–15, 2018, Stockholm, Sweden. © 2018 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

The solution of a MAPF problem is a sequence of positions in time for each agent that satisfies the conditions that no two agents meet at the same node at the same time, and no two neighboring agents swap their positions at one step. The moves of the agents are discrete and synchronous. In this paper, we will focus on solutions that are makespan optimal – the total time until the last agent reaches its destination is minimized. This requirement of optimality makes the MAPF problem NP-hard [11].

The classical MAPF is usually solved by algorithms that can be divided into two categories:

(1) **Reduction-based solvers.** Many solvers reduce MAPF to another known problem such as SAT [17], integer linear programming [21], and answer set programming [3]. These approaches are based on fast solvers that work very well with unit cost parameters.
(2) **Search-based solvers.** On the other hand, many recent solvers are search-based. Some are variants of A* over a global search space – all possibilities how to place agents into the nodes of the graph [15]. Other make use of novel search trees [2, 13, 14].

All of the above approaches to solving the MAPF problem are designed and tested on graphs with unit-length arcs and unit-capacity arcs and nodes.

3 SCHEDULING MODEL

This section gives a scheduling-based model for the classical MAPF with unit lengths and unit capacities. For the sake of simplicity, we first describe the model restricted such that no agent can visit the same node more than once. Then, we will show how to eliminate this restriction.

The requirement that agents do not meet at a node at the same time leads to a classical expression of a unary (disjunctive) resource. In constraint programming, these disjunctive constraints are known to propagate badly and special global constraints modeling resources and efficient filtering algorithms have been proposed [19]. Hence it seems natural to exploit such constraints in a model, where the presence of an agent at a node is modeled as an activity. Activity can be conceived as an interval variable whose start time and end time are denoted by predicates $StartOf$ and $EndOf$, and the difference between the end time and the start time of the activity can be set using predicate $LengthOf$.

We will exploit the concept of *optional activities* [8], which is widely used in scheduling. Again, an optional activity can be conceived as an optional interval variable that can be set to be present or absent. The predicate $PresenceOf$ is used to determine whether or not the activity is present in the resulting schedule. The succeeding and preceding nodes in a path of an agent will be entailed by whether or not an activity corresponding to the arc and the agent is present. The activities must be connected via temporal constraint to define a path from the origin to the destination.

3.1 Single-Layer Model

Formally, for each agent $a \in A$ and each node $x \in V$, we introduce three optional activities $N[x,a]$, $N^{out}[x,a]$, and $N^{in}[x,a]$. The activity $N[x,a]$ corresponds to the time of an agent a spent at node x. The activities $N^{in}[x,a]$ and $N^{out}[x,a]$ describe the time spent in the incoming and outgoing arcs. Next, for each agent $a \in A$ and each arc $(x,y) \in E$, we introduce an optional activity $A[x,y,a]$. Notice that all the activities in the model are optional. Finally, we introduce an integer variable *MKSP* to denote the end of schedule (makespan).

The idea is that the path of an agent corresponds to the activities that are present in the solution and that in turn correspond to the nodes and arcs in the path. In the terminology of hierarchical scheduling, it can be conceived such that each activity $N^{out}[x,a]$ has activities $A[x,y,a]$ corresponding to the arcs outgoing from the node x as its children, and symmetrically, $N^{in}[x,a]$ has the activities $A[y,x,a]$ corresponding to the arcs incoming to the node x as its children. Hence, each activity $A[x,y,a]$ has two parents: $N^{out}[x,a]$ and $N^{in}[y,a]$ because the arc (x,y) is an outgoing arc for node x and an incoming arc for node y.

Formally, for each agent $a \in A$, the following logical constraints are introduced:

$$PresenceOf(N[orig(a),a]) = 1 \tag{1}$$

$$PresenceOf(N[dest(a),a]) = 1 \tag{2}$$

$$PresenceOf(N^{in}[orig(a),a]) = 0 \tag{3}$$

$$PresenceOf(N^{out}[dest(a),a]) = 0 \tag{4}$$

$$\forall x \in V \setminus \{orig(a)\} : PresenceOf(N[x,a]) \Leftrightarrow PresenceOf(N^{in}[x,a]) \tag{5}$$

$$\forall x \in V \setminus \{dest(a)\} : PresenceOf(N[x,a]) \Leftrightarrow PresenceOf(N^{out}[x,a]) \tag{6}$$

$$\forall x \in V \setminus \{orig(a)\} : Alternative\Big(N^{in}[x,a], \bigcup_{(y,x)\in E} A[y,x,a]\Big) \tag{7}$$

$$\forall x \in V \setminus \{dest(a)\} : Alternative\Big(N^{out}[x,a], \bigcup_{(x,y)\in E} A[x,y,a]\Big) \tag{8}$$

The definition of the $Alternative$ constraint, which gets an interval variable as the first argument and a set of interval variables as the second argument, is as follows. If the activity given as the first argument is present, then exactly one activity from the set of activities given as the second argument is present. In addition, it ensures that the start times and end times of the present activities are equal. Since this implication goes only in one direction, we have to impose the following implication constraints in order to find a correct path, for each agent $a \in A$:

$$\forall (x,y) \in E : PresenceOf(A[x,y,a]) \Rightarrow PresenceOf(N^{in}[y,a]) \tag{9}$$

$$\forall (x,y) \in E : PresenceOf(A[x,y,a]) \Rightarrow PresenceOf(N^{out}[x,a]) \tag{10}$$

The durations of activities N, N^{out}, and N^{in} are to be found, whereas the durations of activities $A[x,y,a]$ are fixed to 1, i.e., $LengthOf(A[x,y,a]) = 1$. Thanks to the $Alternative$ constraints, the processing times of activities N^{out} and N^{in} will span over the child activity A that will be present, and for the rest, the following constraints need to be added, for each agent $a \in A$:

$$StartOf(N[orig(a),a]) = 0 \tag{11}$$

$$EndOf(N[dest(a),a]) = MKSP \tag{12}$$

$$\forall x \in V \setminus \{orig(a)\} : StartOf(N[x,a]) = EndOf(N^{in}[x,a]) \tag{13}$$

$$\forall x \in V \setminus \{dest(a)\} : EndOf(N[x,a]) = StartOf(N^{out}[x,a]) \tag{14}$$

Note that the equality in constraint (12) is essential for the correctness of the models. Changing it to an inequality ($\leq$) would mean that the agent disappears after reaching its destination, and thus other agents may use that node, which is prohibited in MAPF.

We need to introduce the constraint precluding the agents from occurring at the same node at the same time, that is, for each node $x \in V$, we add:

$$NoOverlap\left(\bigcup_{a \in A} N[x, a]\right) \tag{15}$$

The *NoOverlap* constraint on a set of activities states that it constitutes a chain of non-overlapping activities, any activity in the chain being constrained to end before the start of the next activity in the chain. The *NoOverlap* constraint uses non-strict inequalities. However, if an agent leaves a node at time t, another agent is allowed to enter the same node no sooner than at time $t+1$. In fact, the times spent by agents at nodes are mostly zero (agents go through the nodes without waiting there). Hence, the *NoOverlap* constraint is given a so-called *transition distance* matrix TDM, which expresses a minimal delay that must elapse between two successive activities. More precisely, $TDM(N_1, N_2)$ gives a minimal allowed time difference between $StartOf(N_2)$ and $EndOf(N_1)$. Thus, the constraint (15) is given a TDM containing value 1 for each ordered pair of activities from the constraint, which ensures that the time distance between two consecutive visits of a node is at least one.

To prevent agents from using an arc at the same time (swap), we add, for each pair of distinct nodes $x, y \in V$ connected by arc:

$$NoOverlap\left(\bigcup_{a \in A} \{A[x, y, a], A[y, x, a]\}\right) \tag{16}$$

In this case, the transition distance matrix is not needed at all because the default values are 0.

Finally, the objective is to minimize the makespan, i.e., min *MKSP*.

Soundness

Proposition 3.1. *The single-layer model finds a solution to the MAPF problem where no agent visits the same node more than once if and only if such a solution exists.*

Proof. The solution of the single-layer model consists of selection of activities and their time allocation. The activities corresponding to origins and destinations of agents must be selected due to constraints (1) and (2). The constraints (5)-(10) ensure that if a node is used on some path then there must be exactly one incoming and one outgoing arc selected (except for the origin, where no incoming arc is used due to (3), and for the destination where no outgoing arc is selected due to (4)). No activity outside the path is selected as such activities would have to form a loop due to constraints (5)-(10), but that would violate the temporal constraints (13) and (14). Each path starts at time zero (11) and finishes at time *MKSP* (12). Finally, activities in nodes are not overlapping (15), and agents cannot use the same arc at the same time (16). □

3.2 Multi-Layer Model

In order to let the agents visit the same nodes repeatedly, we simply create a copy of the original graph and add extra arcs to enable transitions between the two graphs. The copies of the original graph will be henceforth referred to as *layers*. Assuming there is a node in which an agent might want to occur at most ℓ-times, we create ℓ such layers. We will show later how ℓ is chosen.

Formally, all the activities in the model are extended with one dimension, corresponding to the layer to which the activity belongs. Now we use $N[x, a, k]$, $N^{in}[x, a, k]$, $N^{out}[x, a, k]$, and $A[x, y, a, k]$, where $k \in \{1, \ldots, \ell\}$ corresponds to the layer. To allow the transitions between two consecutive layers, we introduce for $k \in \{1, \ldots, \ell - 1\}$: $A[x, x, a, k]$, which corresponds to transiting an agent a from a node x at layer k to the node x at layer $k+1$. The duration of activity $A[x, x, a, k]$ is set to 0, that is, $LengthOf(A[x, x, a, k]) = 0$. Note that going through arc $A[x, x, a, k]$ is in fact not a move but merely a transition to another layer. Due to the length 0, an agent can transit an arbitrary number of layers instantly, which is necessary to reach the destination in the final layer ℓ (even if the agent does not re-visit any node).

Also, we need to modify the constraints. The constraints (1)–(8) are updated, for each agent $a \in A$, to the following constraints:

$$PresenceOf(N[orig(a), a, 1]) = 1 \tag{17}$$

$$PresenceOf(N[dest(a), a, \ell]) = 1 \tag{18}$$

$$PresenceOf(N^{in}[orig(a), a, 1]) = 0 \tag{19}$$

$$PresenceOf(N^{out}[dest(a), a, \ell]) = 0 \tag{20}$$

$$\forall x \in V, \forall k \in \{1, \ldots, \ell\}, x \neq orig(a) \lor k \neq 1 :$$
$$PresenceOf(N[x, a, k]) \Leftrightarrow PresenceOf(N^{in}[x, a, k]) \tag{21}$$

$$\forall x \in V, \forall k \in \{1, \ldots, \ell\}, x \neq dest(a) \lor k \neq \ell :$$
$$PresenceOf(N[x, a, k]) \Leftrightarrow PresenceOf(N^{out}[x, a, k]) \tag{22}$$

$$\forall x \in V, \forall k \in \{1, \ldots, \ell\} :$$
$$Alternative(N^{in}[x, a, k], \{A[x, x, a, k-1]\} \cup \bigcup_{(y,x) \in E} A[y, x, a, k]) \tag{23}$$

$$\forall x \in V, \forall k \in \{1, \ldots, \ell\} :$$
$$Alternative(N^{out}[x, a, k], \{A[x, x, a, k]\} \cup \bigcup_{(x,y) \in E} A[x, y, a, k]) \tag{24}$$

Note that in order to ease the notational clutter, we neglect the special cases where the transition arc is not defined, i.e., there is no transition arc incoming to the first layer ($A[x, x, a, 0]$) and no transition arc outgoing from the layer ℓ ($A[x, x, a, \ell]$). Hence the transition arcs are used (added in the union in constraints (23) and (24)) only if they are defined.

The constraints (9) and (10) are modified, and extra implications for transitions are added:

$$\forall (x, y) \in E, \forall k \in \{1, \ldots, \ell\} :$$
$$PresenceOf(A[x, y, a, k]) \Rightarrow PresenceOf(N^{in}[y, a, k]) \tag{25}$$

$$\forall (x,y) \in E, \forall k \in \{1,\ldots,\ell\}: \quad PresenceOf(A[x,y,a,k]) \Rightarrow PresenceOf(N^{out}[x,a,k]) \qquad (26)$$

$$\forall x \in V, \forall k \in \{1,\ldots,\ell-1\}: \quad PresenceOf(A[x,x,a,k]) \Rightarrow PresenceOf(N^{in}[x,a,k+1]) \qquad (27)$$

$$\forall x \in V, \forall k \in \{1,\ldots,\ell-1\}: \quad PresenceOf(A[x,x,a,k]) \Rightarrow PresenceOf(N^{out}[x,a,k]) \qquad (28)$$

The constraints (11)–(14) are simply changed to the following constraints:

$$StartOf(N[orig(a),a,1]) = 0 \qquad (29)$$

$$EndOf(N[dest(a),a,\ell]) = MKSP \qquad (30)$$

$$\forall x \in V, \forall k \in \{1,\ldots,\ell\}, x \neq orig(a) \vee k \neq 1: \quad StartOf(N[x,a,k]) = EndOf(N^{in}[x,a,k]) \qquad (31)$$

$$\forall x \in V, \forall k \in \{1,\ldots,\ell\}, x \neq dest(a) \vee k \neq \ell: \quad EndOf(N[x,a,k]) = StartOf(N^{out}[x,a,k]) \qquad (32)$$

Finally, the constraints (15)–(16) are updated as follows:

$$NoOverlap\left(\bigcup_{\substack{a\in A \\ k\in\{1,\ldots,\ell\}}} N[x,a,k] \right) \qquad (33)$$

$$NoOverlap\left(\bigcup_{\substack{a\in A \\ k\in\{1,\ldots,\ell\}}} \{A[x,y,a,k], A[y,x,a,k]\} \right) \qquad (34)$$

Recall that the *NoOverlap* constraint over nodes was given the transition distance matrix *TDM* ensuring that the time distances between two consecutive visits of a node are at least one. In this case, however, we need to distinguish the time distance of two distinct agents, which must be at least 1, and the time distance of one agent in distinct layers, which must be allowed to be 0 in order for an agent to be able to transit an arbitrary number of layers instantly. More precisely, $TDM(N[x,a,k_1], N[x,a,k_2]) = 0$, and $TDM(N[x,a_1,k_1], N[x,a_2,k_2]) = 1$, for $a_1 \neq a_2$.

Constraint (34) does not need any transition distance matrix as the default values 0 are desired.

3.3 Algorithm

We presented a constraint model of the problem for a given number of layers. We will show now how to bound the number of layers for a given makespan and then we will present an algorithm for finding the minimal makespan.

Number of Layers

Proposition 3.2. *Let p_{min} be the shortest path from the origin node to the destination node of an agent, and let MB be an arbitrary upper bound on makespan (can be the optimal makespan). Then, to solve correctly any instance of MAPF problem, it suffices to construct the model with $\frac{MB-p_{min}}{2}+1$ layers. Moreover, this bound cannot be improved.*

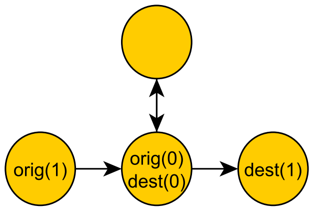

Figure 1: Illustration for the proof of Proposition 3.2.

Proof. The minimum number of nodes that every agent must go through (excluding its origin node) in order to reach its destination node is at least p_{min}. Hence, the maximum number of steps that an agent can spend on repeating already visited nodes is $MB - p_{min}$. However, in order to visit the same node again, an agent must go away from that node and go back to that node, which takes at least two steps. Thus, to get the worst case, an agent has to do cycles of length two to keep visiting the same node. Hence, in the worst case, an agent can revisit the same node as many as $\frac{MB-p_{min}}{2}$ times. Since at least one layer is always required even without repetitions, we obtain the upper bound $\frac{MB-p_{min}}{2}+1$ on the necessary number of layers.

To show that this bound cannot be improved, we construct a problem such that the necessary number of layers equals the bound (Figure 1). Consider an agent 0 having its destination node equal to its origin node ($orig(0) = dest(0)$), hence $p_{min} = 0$, and agent 1 passing that node so that the agent 0 has to jump back and forth (out of his origin node and back). Then the agent 0 occurs in the destination node (including the initial configuration) exactly $\frac{MB-p_{min}}{2}+1 = \frac{2-0}{2}+1 = 2$ times. □

When we know how many layers are needed, we can design the algorithm for solving the MAPF problem. Because all the domains of variables in Constraint Programming (CP) are finite, we need to first obtain an upper bound *UB* on makespan. Hence, before constructing any model, we first run a polynomial-time algorithm Push and Swap (PaS) [9], which finds arbitrary solution provided the problem is solvable (and assuming $|A| \leq |V|-2$), and thus we obtain a valid *UB*. Now, we could calculate the necessary number of layers according to Proposition 3.2, but because *UB* obtained from PaS is very loose, the number of layers, as well as the sizes of the domains, would be prohibitively large. That is why we start with one layer and increase the number of layers until we find a solution. This solution might not be makespan-optimal, but it provides better *UB* which is then used in the final call of the solver that produces a makespan-optimal solution.

The pseudocode of our approach is depicted in Algorithm 1. Calling CP(UB, ℓ) stands for the creation of a model for the problem with ℓ layers and with the upper bound on makespan *UB*, and solving it with a CP solver. The result of calling CP, as well as calling PaS, is the makespan of the solution, or fail if infeasible. In case of CP, the solution is makespan-optimal with respect to the number of layers.

Algorithm 1 Solving MAPF

```
1:  function SolveMAPF
2:      $UB \leftarrow PaS$
3:      if $UB = fail$ then
4:          return $infeasible$
5:      end if
6:      $\ell \leftarrow 1$
7:      $RET \leftarrow \mathrm{CP}(UB, \ell)$
8:      while $RET = fail$ do
9:          $\ell \leftarrow \ell + 1$
10:         $RET \leftarrow \mathrm{CP}(UB, \ell)$
11:     end while
12:     $\ell \leftarrow \frac{RET - p_{min}}{2} + 1$
13:     return $\mathrm{CP}(RET, \ell)$
14: end function
```

Soundness

Proposition 3.3. *Algorithm 1 finds a makespan-optimal solution to the MAPF problem if and only if a solution exists.*

Proof. Correctness and completeness follow directly from Propositions 3.1 and 3.2, using the fact that the constraints for the multi-layer model are modified such that the non-conflicting paths from the origin node of each agent, which corresponds to the activity $N[orig(a), a, 1]$, to its destination node, which corresponds to the activity $N[dest(a), a, \ell]$, are found whenever a solution exists for the given number of layers. Since the CP solver always returns the makespan-optimal solution for a given number of layers and using the fact that the problem is ultimately modeled with the sufficient number of layers according to Proposition 3.2, the obtained solution is optimal. □

Note that we also tried different variants of the multi-layered model. For example, we tested the models where the transitions of agents between layers are synchronized, and hence the *NoOverlap* constraints are imposed only within one layer instead of over all layers. However, these models turned out to be less efficient in that they require a higher number of layers to maintain optimality.

4 GENERALIZED MAPF

Recall that the main motivation for using the scheduling-based model was its applicability to more general problems. One such generalization of MAPF is labeling the arcs with lengths (weights), which determine the duration of going from one node to another, and with capacities determining the number of agents that can occur at one arc at the same time (referred to as *occupancy*). Now we work with an arc-weighted graph $G = (V, E, w, occ)$, where $w(x, y)$ indicates the duration of moving an agent over the arc (x, y), and $occ(\{x, y\})$ stands for how many agents can use the pair of arcs (x, y) and (y, x) at the same time.

Using non-directional occupancy allows us to model the original MAPF problem with prohibited swaps of agents (by setting occupancy to 1), which would not be possible with directional capacities. The motivation is that a road capacity is shared in both directions, while the travel time can be different (e.g., uphill/downhill).

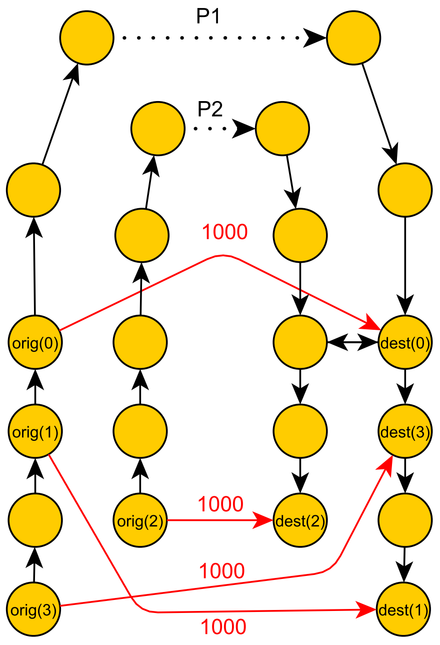

Figure 2: Pathological example for SM-OPT.

4.1 Scheduling-Based Approach

We model the generalization of MAPF using the multi-layer model described in the previous section, where the *NoOverlap* constraints over arcs (34) are substituted with cumulative functions [8] with capacities set to corresponding $occ(\{x, y\})$, and the duration of activities corresponding to arcs are set to the weights of arcs, i.e., $LengthOf(A[x, y, a, k]) = w(x, y)$.

Let us refer to the Algorithm 1 using the model with these generalizations as SM-OPT. It is easy to verify that the propositions from the previous section can be used also for the generalized MAPF. Clearly, increasing *occ* can only improve the makespan. However, the bound on makespan *MB* and the shortest path of an agent p_{min} in Proposition 3.2 must be measured with respect to the lengths of arcs. Measuring *MB* and p_{min} w.r.t. the number of arcs would be incorrect. To see this, consider the situation in Figure 2.

Each origin node of any agent is directly connected to its destination node with one arc of a very large length, say 1000, which is depicted in red. The other black solid arcs are unit-length, while the dotted arrows P1 and P2 stand for very long paths consisting of unit-length arcs (and the corresponding number of nodes). Suppose the length of the shortest path from $orig(i)$ to $dest(i)$, for $i \neq 0$, is 500 (by the paths P1 and P2), and the length of the shortest path from $orig(0)$ to $dest(0)$ is 496. Hence, $p_{min} = 496$, and the minimal makespan is 500, which is achieved via paths P1 and P2 with the caveat that agent 0 reaches its destination first and then he has to jump back and forth to the neighboring node so as to clear the way for the other agents.

What we get from the PaS algorithm in this example is the solution where each agent goes over the direct arc to the destination, hence the obtained makespan is 1000. If we treated the result w.r.t. the number of arcs, that is, $MB = 1$ and $p_{min} = 1$, we would get

$\frac{1-1}{2} + 1 = 1$, so that the algorithm would settle for the solution with just one layer, which is far from optimum (we can set arbitrarily long arcs instead of 1000).

Now, if we use the Proposition 3.2 correctly with the makespan bound MB and shortest path p_{min} measured w.r.t. the length of arcs, we could compute the number of layers as $\frac{500-496}{2} + 1 = 3$, which is exactly the number that is necessary because agent 0 occurs in its destination exactly three times. This example also confirms that the bound on the number of layers is tight.

However, as we obtain the solution of makespan 1000 from the PaS algorithm as well as from the first run of CP(1000, 1), the SM-OPT computes the final number of layers as $\ell = \frac{1000-496}{2} + 1 = 253$, which is unnecessarily too much. The question how to improve this bound remains open.

4.2 Heuristic Approach

The main problem with the generalized MAPF is that the number of layers calculated using the Proposition 3.2 may be very high. In order to observe the potential of the scheduling-based approach, let us define SM-HEUR as Algorithm 1 modified such that it starts from one layer and increases the number of layers by one only until a feasible solution is found and this solution is returned to the user. Formally, the last call to the solver at line 13 is not realized and instead, the algorithm returns *RET* calculated in the loop. Clearly, this does not guarantee to find a makespan-optimal solution, and as the pathological case from Figure 2 shows, it can be arbitrarily far from the optimum.

4.3 SAT-Based Approach

While the generalizations described above are easy to implement in our scheduling-based model, the same generalizations can be very challenging (both in implementation and runtime) in other existing approaches. One of the most popular approaches is reducing the MAPF problem to a SAT formula [17]. We implemented such solver using the Picat language, which has been showed to be comparable with the state of the art SAT-based MAPF solver [1].

First, we start describing the SAT model of the classical MAPF problem with unit lengths and unit capacities. We define the two following sets of variables: $\forall x \in V, \forall a \in A, t \in \{0, \ldots, T\} : At(x, a, t)$ meaning that agent a is at node x at time step t; and $\forall (x, y) \in E, \forall a \in A, t \in \{0, \ldots, T-1\} : Pass(x, y, a, t)$ meaning that agent a goes through arc (x, y) at time step t. An arc (x, x) is added to E, thus $Pass(x, x, a, t)$ means that agent a stays at node x at time step t. To model the MAPF problem, we introduce the following constraints:

$$\forall a \in A : At(orig(a), a, 0) = 1 \tag{35}$$

$$\forall a \in A : At(dest(a), a, T) = 1 \tag{36}$$

$$\forall a \in A, \forall t \in \{0, \ldots, T\} : \sum_{x \in V} At(x, a, t) \le 1 \tag{37}$$

$$\forall x \in V, \forall t \in \{0, \ldots, T\} : \sum_{a \in A} At(x, a, t) \le 1 \tag{38}$$

$$\forall x \in V, \forall a \in A, \forall t \in \{0, \ldots, T-1\} : \quad At(x, a, t) \implies \sum_{(x,y) \in E} Pass(x, y, a, t) = 1 \tag{39}$$

$$\forall (x, y) \in E, \forall a \in A, \forall t \in \{0, \ldots, T-1\} : \quad Pass(x, y, a, t) \implies At(y, a, t+1) \tag{40}$$

$$\forall (x, y) \in E, \forall t \in \{0, \ldots, T-1\} : \quad \sum_{a \in A} Pass(x, y, a, t) + Pass(y, x, a, t) \le 1 \tag{41}$$

The constraints (35) and (36) ensure that the starting and goal positions of all agents are valid. The constraints (37) and (38) ensure that each agent occupies at most one node while every node is occupied by at most one agent. The correct movement in the graph is ensured by constraints (39)–(41). In order, they ensure that if an agent is in a node, it needs to leave by one of the outgoing arcs (39). If an agent is using an arc, it needs to arrive at the corresponding node in the next time step (40). And last, we forbid two agents to occupy two opposite arcs at the same time (no-swap constraint) (41).

To find the optimal makespan, we iteratively increase the upper bound on makespan T until a satisfiable formula is generated.

To introduce the generalization of the problem we change some of the constraints. To introduce arc lengths, we need to change the arrival time step in constraint (40). It needs to be set only for time steps that make sense in accordance with the length of the arc. To introduce arc occupancies we need to change the constraint (41) in the following way:

$$\forall (x, y) \in E, \forall t \in \{0, \ldots T - w(e)\} : \quad \sum_{\substack{a \in A \\ t' \in t, \ldots, t+w(e)-1}} Pass(x, y, a, t') + Pass(y, x, a, t') \le occ(\{x, y\}) \tag{42}$$

This constraint is needed to ensure that the number of agents that are moving over one arc in the same direction or over two opposite arcs simultaneously does not exceed its occupancy.

While these changes may seem straightforward written as equations, it is not very natural for SAT to work with numbers other than ones and zeros (the Picat solver transforms the above arithmetic constraints to clauses of a Boolean formula automatically). Furthermore, adding lengths to the arcs increases the number of time steps T needed to solve the problem drastically and therefore introduces many new variables to the generated formula.

5 RESULTS OF EXPERIMENTS

We implemented the scheduling approach in the IBM CP Optimizer version 12.8 [7]. The only parameter that we adjusted is `Workers`, which is the number of threads the solver can use and which we set to 1. For the SAT-based approach we used the Picat language and compiler version 2.2#3 [1]. The experiments were run on a PC with an Intel® Xeon™ CPU E5-2660 v2 running at 2.00 GHz with 16 GB of RAM. We used a cutoff time of 1000 seconds per problem instance.

5.1 Implementation Details

We first compute the all-pairs-shortest-path matrix *sp* using the *Floyd-Warshall* algorithm [4] as the preprocessing phase. We set

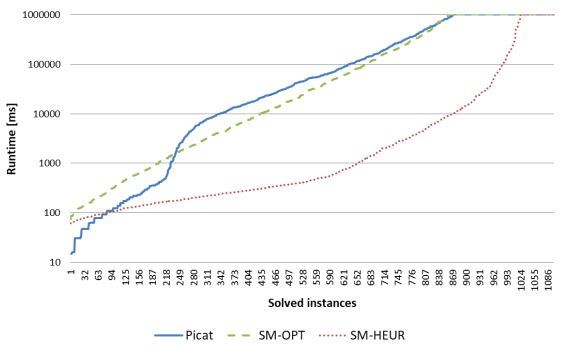

Figure 3: Comparison of all generated instances.

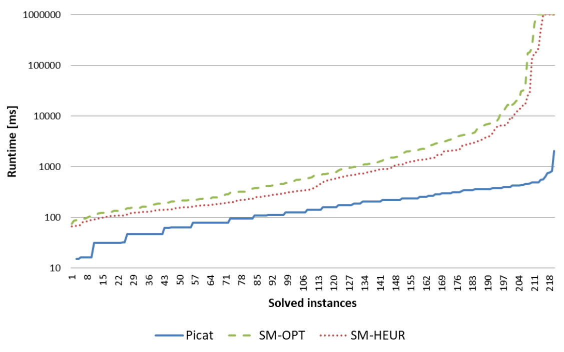

Figure 4: Instances with the maximum length of 1.

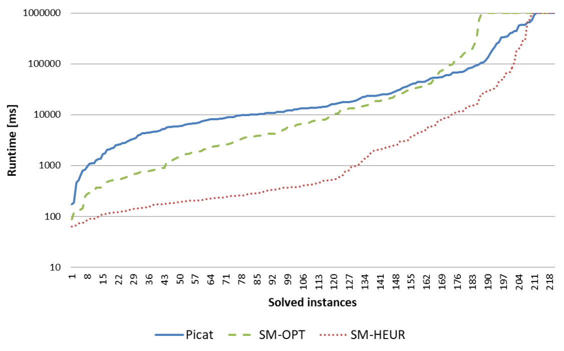

Figure 5: Instances with the maximum length of 50.

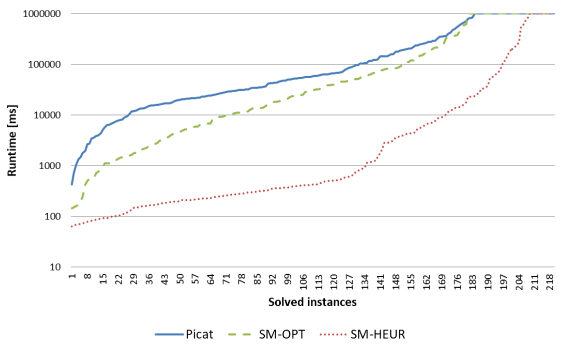

Figure 6: Instances with the maximum length of 100.

the lower bound on makespan to be the maximum, over all agents, of the shortest paths from the origin node to the destination node of the agent.

To represent the activities in the model, we use the *interval variables* of the CP Optimizer, which are designed for the scheduling problems and support specialized constraints such as $Alternative$ and $NoOverlap$. The bounds of the intervals and other time variables are limited using the sp matrix, namely, $\forall a \in A, \forall k \in \{1, \ldots, \ell\}$, $\forall x \in V$, the lower bound on start time (EST) of $N[x, a, k]$ is set to $sp(orig(a), x)$, and the upper bound on end time (LCT) of $N[x, a, k]$ is set to $UB - sp(x, dest(a))$. Further, if $EST > LCT$, it means that the node x cannot be passed through by agent a, and thus we omit creating variables associated with node x and agent a.

5.2 Problem Instances

The problem instances are created over strongly biconnected undirected graphs. These types of graphs ensure that the instance is always solvable as long as there are at least 2 agents less than the number of nodes [16]. To create the different complexity of the instances, we incrementally increase the number of nodes in the graph (from 20 nodes to 40 nodes with the increment of 5) as well as the number of agents in the graph (from 2 to 9 agents). Both the origin and destination positions of agents are randomly placed in the graph.

Further, we added lengths to the arcs. The length of each arc is chosen uniformly at random from the range $[1, W]$, where $W \in \{1, 50, 100, 200, 300\}$. Occupancy is a global attribute for the instance and is also incrementally increased (from 1 to the number of agents in the instance). Altogether we generated 1100 instances.

5.3 Results

The charts show the number of problems solved (x-axis) within a given time (y-axis). Hence, a curve that is closer to the bottom right represents a better method. All of the charts have a logarithmic scale on the y-axis.

We compare three methods described above: SM-OPT, SM-HEUR, and the SAT-based model (labeled as "Picat"). Recall that SM-HEUR does not guarantee to find an optimal solution.

The overall comparison of SM-OPT, SM-HEUR, and Picat is depicted in Figure 3. In Figures 4–8, we show comparisons for the given maximum length of arcs W to see how the lengths of arcs affect the efficiency of the approaches.

It can be noted that when all arcs are unit-length, which is closest to the original MAPF problem, Picat is positively faster than the scheduling-based methods (Figure 4). However, with increasing upper bound on arc length (and thus increasing the differences in individual arc lengths in an instance), the scheduling-based methods are becoming faster. This can be seen with the maximum length of 50 (Figure 5), where SM-OPT is comparable with Picat, and with the maximum length of 100 (Figure 6), where SM-OPT is faster than Picat.

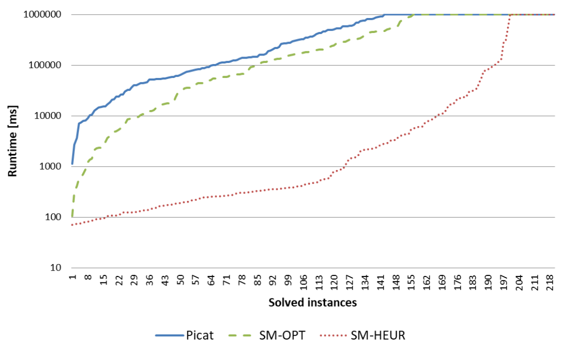

Figure 7: Instances with the maximum length of 200.

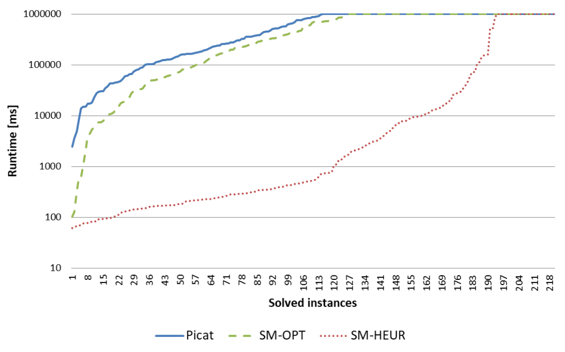

Figure 8: Instances with the maximum length of 300.

max length	1	50	100	200	300
1 *s*	0.59	5.38	7.50	inf	inf
10 *s*	0.89	1.47	2.73	3.88	4.00
100 *s*	0.94	0.94	1.15	1.29	1.73
1000 *s*	0.96	0.89	0.99	1.09	1.11

Table 1: Ratio of solved instances of SM-OPT to Picat within the given time limit.

In addition, Table 1 shows the ratio of the number of solved instances by SM-OPT to the number of solved instances by Picat, within the selected time limit (1–1000 seconds), for the given upper bound on arc length. In the first row, inf means that Picat did not solve any instance in one second.

The results clearly confirm the hypothesis that with the increasing length of arcs, the advantage of SM-OPT over Picat is increasing, which is even more apparent for lower time limits.

Figure 9 shows the comparison on instances with the maximum length of 1 and arc occupancy only of 1, which is the classical MAPF problem. The similarity of the charts in Figure 9 and in Figure 4 shows that the occupancy parameter does not have a significant impact on the efficiency of any approach.

The results also show that the SM-HEUR is by orders of magnitude faster than SM-OPT, while the number of problems where it found a sub-optimal solution is 71 out of 860. However, the average (over these 71 sub-optimal answers) increase in the makespan is by

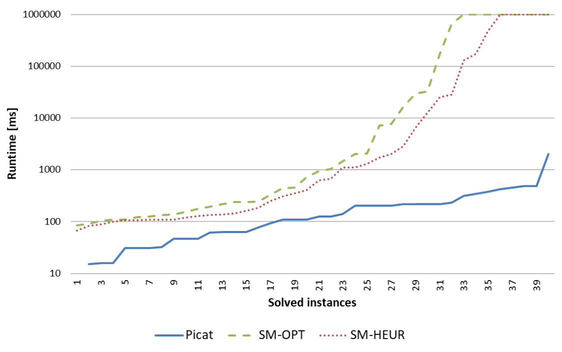

Figure 9: Instances with the maximum length of 1 and maximum arc occupancy of 1.

25.34 % from the optimum. This should be an incentive for further research on the number of re-visits of an agent at a node.

Another advantage of our technique over the SAT-based approach is that even if it does not finish in time, it can find at least some solution. Out of the 240 instances that SM-OPT did not solve in the given time limit, it found a feasible solution in 195 cases, i.e., it did not find any solution only for 45 instances. This contrasts with Picat which either finds an optimal solution or nothing. Thus, Picat did not find any solution for 230 instances.

6 CONCLUSIONS

We showed how to model a MAPF problem as a scheduling problem, where nodes and arcs are seen as resources used by the agents. First, we modeled the classical problem often used in the literature (unit lengths and capacities). Then we extended this problem by introducing arc length and arc occupancy limits to simulate real-world conditions. We showed that this extension is easy to model in our scheduling approach, but it is more challenging for a classical SAT-based approach.

We compared the discussed approaches experimentally. As expected, the classical SAT approach outperforms the scheduling-based methods on the classical MAPF problem instances. However, with the increasing lengths of arcs, our scheduling-based techniques outperform the SAT-based approach.

The hardness of the problem can be linked to the number of layers needed for its solution. This is true for both the scheduling approach and the SAT approach. One timestep is equivalent to one layer in the SAT approach and therefore we cannot improve on the number of layers needed. On the other hand, the number of layers in the scheduling approach is equivalent to the number of returns to a single node. We created a formula that estimates the number of layers needed. In many cases, however, this number is greatly overestimated. A future work should lead to a better formula for the number of layers, probably one that is not based on the makespan estimate, but rather on some other value.

ACKNOWLEDGMENTS

Research is supported by the Czech-Israeli Cooperative Scientific Research Project 8G15027, by SVV project number 260 453, and by the Charles University, project GA UK No. 158216.

REFERENCES

[1] Roman Barták, Neng-Fa Zhou, Roni Stern, Eli Boyarski, and Pavel Surynek. 2017. Modeling and Solving the Multi-agent Pathfinding Problem in Picat. In *29th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE Computer Society, 959–966. https://doi.org/10.1109/ICTAI.2017.00147

[2] Eli Boyarski, Ariel Felner, Roni Stern, Guni Sharon, Oded Betzalel, David Tolpin, and Solomon Eyal Shimony. 2015. ICBS: The Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding. In *Proceedings of the Eighth Annual Symposium on Combinatorial Search, SOCS 2015, 11-13 June 2015, Ein Gedi, the Dead Sea, Israel.*, Levi Lelis and Roni Stern (Eds.). AAAI Press, 223–225. http://www.aaai.org/ocs/index.php/SOCS/SOCS15/paper/view/10974

[3] Esra Erdem, Doga Gizem Kisa, Umut Öztok, and Peter Schüller. 2013. A General Formal Framework for Pathfinding Problems with Multiple Agents. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA.*, Marie desJardins and Michael L. Littman (Eds.). AAAI Press. http://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/view/6293

[4] Robert W. Floyd. 1962. Algorithm 97: Shortest path. *Commun. ACM* 5, 6 (1962), 345. https://doi.org/10.1145/367766.368168

[5] Donggyun Kim, Katsutoshi Hirayama, and Gyei-Kark Park. 2014. Collision Avoidance in Multiple-Ship Situations by Distributed Local Search. *JACIII* 18, 5 (2014), 839–848. https://doi.org/10.20965/jaciii.2014.p0839

[6] Daniel Kornhauser, Gary L. Miller, and Paul G. Spirakis. 1984. Coordinating Pebble Motion on Graphs, the Diameter of Permutation Groups, and Applications. In *25th Annual Symposium on Foundations of Computer Science, West Palm Beach, Florida, USA, 24-26 October 1984*. IEEE Computer Society, 241–250. https://doi.org/10.1109/SFCS.1984.715921

[7] Philippe Laborie. 2009. IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009, Pittsburgh, PA, USA, May 27-31, 2009, Proceedings (Lecture Notes in Computer Science)*, Willem Jan van Hoeve and John N. Hooker (Eds.), Vol. 5547. Springer, 148–162. https://doi.org/10.1007/978-3-642-01929-6_12

[8] Philippe Laborie, Jerome Rogerie, Paul Shaw, and Petr Vilím. 2009. Reasoning with Conditional Time-Intervals. Part II: An Algebraical Model for Resources.. In *FLAIRS conference*. 201–206.

[9] Ryan Luna and Kostas E. Bekris. 2011. Push and Swap: Fast Cooperative Path-Finding with Completeness Guarantees. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, Toby Walsh (Ed.). IJCAI/AAAI, 294–300. https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-059

[10] Nathan Michael, Jonathan Fink, and Vijay Kumar. 2011. Cooperative manipulation and transportation with aerial robots. *Auton. Robots* 30, 1 (2011), 73–86. https://doi.org/10.1007/s10514-010-9205-0

[11] Daniel Ratner and Manfred K. Warmuth. 1990. NxN Puzzle and Related Relocation Problem. *J. Symb. Comput.* 10, 2 (1990), 111–138. https://doi.org/10.1016/S0747-7171(08)80001-6

[12] Malcolm Ross Kinsella Ryan. 2008. Exploiting Subgraph Structure in Multi-Robot Path Planning. *J. Artif. Intell. Res.* 31 (2008), 497–542. https://doi.org/10.1613/jair.2408

[13] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artif. Intell.* 219 (2015), 40–66. https://doi.org/10.1016/j.artint.2014.11.006

[14] Guni Sharon, Roni Stern, Meir Goldenberg, and Ariel Felner. 2013. The increasing cost tree search for optimal multi-agent pathfinding. *Artif. Intell.* 195 (2013), 470–495. https://doi.org/10.1016/j.artint.2012.11.006

[15] Trevor Scott Standley. 2010. Finding Optimal Solutions to Cooperative Pathfinding Problems. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, Maria Fox and David Poole (Eds.). AAAI Press. http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1926

[16] Pavel Surynek. 2009. A novel approach to path planning for multiple robots in bi-connected graphs. In *2009 IEEE International Conference on Robotics and Automation, ICRA 2009, Kobe, Japan, May 12-17, 2009*. IEEE, 3613–3619. https://doi.org/10.1109/ROBOT.2009.5152326

[17] Pavel Surynek. 2012. Towards Optimal Cooperative Path Planning in Hard Setups through Satisfiability Solving. In *PRICAI 2012: Trends in Artificial Intelligence - 12th Pacific Rim International Conference on Artificial Intelligence, Kuching, Malaysia, September 3-7, 2012. Proceedings (Lecture Notes in Computer Science)*, Patricia Anthony, Mitsuru Ishizuka, and Dickson Lukose (Eds.), Vol. 7458. Springer, 564–576. https://doi.org/10.1007/978-3-642-32695-0_50

[18] Jur van den Berg, Jack Snoeyink, Ming C. Lin, and Dinesh Manocha. 2009. Centralized path planning for multiple robots: Optimal decoupling into sequential plans. In *Robotics: Science and Systems V, University of Washington, Seattle, USA, June 28 - July 1, 2009*, Jeff Trinkle, Yoky Matsuoka, and José A. Castellanos (Eds.). The MIT Press. http://www.roboticsproceedings.org/rss05/p18.html

[19] Petr Vilím, Roman Barták, and Ondřej Čepek. 2005. Extension of $O(n \log n)$ Filtering Algorithms for the Unary Resource Constraint to Optional Activities. *Constraints* 10, 4 (2005), 403–425. https://doi.org/10.1007/s10601-005-2814-0

[20] Ko-Hsin Cindy Wang and Adi Botea. 2008. Fast and Memory-Efficient Multi-Agent Pathfinding. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008*, Jussi Rintanen, Bernhard Nebel, J. Christopher Beck, and Eric A. Hansen (Eds.). AAAI, 380–387. http://www.aaai.org/Library/ICAPS/2008/icaps08-047.php

[21] J. Yu and S. M. LaValle. 2013. Planning optimal paths for multiple robots on graphs. In *2013 IEEE International Conference on Robotics and Automation, ICRA 2013*. 3612–3617. https://doi.org/10.1109/ICRA.2013.6631084