MISTA 2017

Scheduling Models for Multi-Agent Path Finding

Roman Barták · Jiří Švancara · Marek Vlk

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. This paper suggests to model the MAPF problem using scheduling techniques, namely, nodes are seen as unary resources. We present three models of the problem. One model is motivated by network flows, another model uses classical unary resource constraints together with path constraints, and the last model works with optional activities. We compare the efficiency of models experimentally.

1 Introduction

There exist numerous practical situations, where a set of agents is moving in a shared environment, each agent having its own destination. For example, traffic junctions and large warehouses are typical examples of congested environments, where agents are moving between locations while sharing paths. In the era of autonomous systems, it is important to have efficient solutions for coordinating such agents.

The above problem is known as multi-agent path finding (MAPF) or cooperative path finding (CPF) [8]. The problem can be formalized as a (directed) graph, where agents are initially distributed at some nodes, each agent having a destination node to reach, and the task is to find a plan of movements for each agent to reach the destination node while not being at the same node as another agent at the same time. A frequent abstraction assumes that agents are moving synchronously and distances between the nodes are identical. Then, at each time step, each agent either moves to a neighboring node or stays in the current node. Grid worlds (such as the famous Lloyd 15-puzzle) are satisfying this assumption. This model makes it natural to use solving techniques based on Boolean satisfiability or state-space search, which are currently two leading approaches to solve MAPF. On the other hand, such an abstraction might

Roman Barták, Jiří Švancara, Marek Vlk
Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic
E-mail: surname@ktiml.mff.cuni.cz

be too restrictive as distances between the nodes might be important (and different) in some practical applications.

In this paper, we suggest models of MAPF that borrow ideas from scheduling and routing problems. We see the nodes (and possibly also the arcs) as resources with limited capacity, which is one in this particular setting but could be larger in future applications. We model the movements of agents using various techniques, namely as network flows, as paths, and as optional activities. The motivation is supporting richer (in comparison to traditional MAPF) temporal and capacity constraints, which makes the models closer to reality. On the other side, there is one extra restriction of our current models with respect to traditional MAPF formulation - the models are designed such that no agent visits the same node more than ones.

2 Background on Multi-Agent Path Finding

The MAPF problem is formulated by a graph and a set of packages (agents) sitting at certain nodes. The task is to transport packages to their destination nodes – each package moves itself – while satisfying some capacity constraints, namely no two packages meet at the same node at the same time. The difference from usual MAPF definition is that in the rest of the paper, we will also assume that no package enters any node more than once.

Let $G = (V, E, w)$ be a directed arc-weighted graph and P be a set of packages. The weight $w(a)$ indicates the duration of moving a package over the arc a. In many MAPF formulations, this duration is expected to be one. For each package p we denote $orig(p) \in V$ the original location (node) of the package and $dest(p) \in V$ its destination node. Let $InArcs(x)$ be the set of incoming arcs to x and $OutArcs(x)$ be the set of outgoing arcs from x. Formally,

$$InArcs(x) = \{(y, x) \mid (y, x) \in E\},$$
$$OutArcs(x) = \{(x, y) \mid (x, y) \in E\}$$

The solution for MAPF problem as described above is a sequence of positions in time for each package that satisfies the condition that no two packages are at the same node at the same time. In this paper, we will focus on solutions that are makespan optimal – the total time until the last package reaches its destination is minimized.

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 [10], inductive logic programming [12] 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 [9]. Other make use of novel search trees [7, 2, 6].

3 Flow Model

The *Flow model* is motivated by the model for the closely related problem of multiple-origin multiple-destination problem [1]. The model consists of two parts, a logical one and a numerical one. The logical part describes a valid path for each package using the idea of network flows. The numerical part describes temporal and resource constraints, namely that paths for different packages do not overlap in time and space.

3.1 The Logical Part (Modeling Paths)

For each package $p \in P$ and for each arc $a \in E$ we introduce a Boolean decision variable $Used[a, p]$ that indicates whether or not arc a is used to transport package p. For each package $p \in P$ and for each vertex $x \in V$ a Boolean variable $Flow[x, p]$ indicates whether or not the transport of package p goes through the vertex x.

To model a transport path for a package we specify the flow preservation constraints. These constraints describe that each package must leave its origin and must arrive at its destination, and if the package goes through some vertex then it must enter the vertex and leave it (both exactly once). In the case of origin, the package only leaves it and, similarly, in the case of destination, the package only enters it. Formally, for each package $p \in P$ we introduce the following flow preservation constraints (recall that domains of all the variables are Boolean, that is, $\{0, 1\}$):

$$\forall a \in InArcs(orig(p)) : Used[a, p] = 0 \tag{1}$$

$$\forall a \in OutArcs(dest(p)) : Used[a, p] = 0 \tag{2}$$

$$Flow[orig(p), p] = 1 \tag{3}$$

$$Flow[dest(p), p] = 1 \tag{4}$$

$$\forall x \in V \setminus \{orig(p)\} : \sum_{a \in InArcs(x)} Used[a, p] = Flow[x, p] \tag{5}$$

$$\forall x \in V \setminus \{dest(p)\} : \sum_{a \in OutArcs(x)} Used[a, p] = Flow[x, p] \tag{6}$$

3.2 The Numerical Part (Modeling Nodes as Resources)

The numerical part specifies non-overlapping constraints, namely two packages do not meet at the same node at the same time, and travel time between the nodes that is expressed by weights of arcs. To model the time interval when a package $p \in P$ stays in a node $x \in V$, we introduce two numerical variables $InT[x, p]$ and $OutT[x, p]$ modeling the time when the package enters the node and when it leaves the node respectively. We can describe the travel time of package p between the nodes x and y through the arc a as follows:

$$Used[a, p] \Rightarrow OutT[x, p] + w(a) = InT[y, p]. \tag{7}$$

If the package p is going through the node x then the package cannot enter the node before it leaves it:

$$InT[x, p] \leq OutT[x, p]. \tag{8}$$

Let $sp(x, y)$ be the length of the shortest path from node x to node y. Then we can calculate bounds of the time variables as follows:

$$\forall x \in V \setminus \{orig(p)\} : Flow[x, p] \Rightarrow OutT[orig(p), p] + sp(orig(p), x) \leq InT[x, p] \quad (9)$$

$$\forall x \in V \setminus \{dest(p)\} : Flow[x, p] \Rightarrow OutT[x, p] + sp(x, dest(p)) \leq InT[dest(p), p] \quad (10)$$

Let *MKSP* be the time when each package must be in its destination - it corresponds to makespan of the schedule. Then we set the times in package's origin and destination as follows:

$$InT[orig(p), p] = 0 \quad (11)$$

$$OutT[dest(p), p] = MKSP \quad (12)$$

Finally, to model that two packages p_1 and p_2 do not meet at the same node x, we need to specify that their times of visit do not overlap:

$$(Flow[x, p_1] \wedge Flow[x, p_2]) \Rightarrow (OutT[x, p_1] < InT[x, p_2] \vee OutT[x, p_2] < InT[x, p_1]) \quad (13)$$

3.3 Model Soundness

It is easy to prove that the Flow model is sound, that is, every consistent instantiation of variables defines a solution to the MAPF problem. The constraints (1)-(6) define a single path from origin to destination for each package, i.e., the variables *Flow* and *Used* are equal to one for nodes and arcs used on the path and equal to zero for all other nodes and arcs. The origin and destination must be on the path due to constraints (3) and (4). The path must continue from origin due to (6) and must reach the destination due to (5). The path cannot start and cannot finish in any other node due to constraints (5) and (6). The flow constraints allow a loop to be formed in the graph, but such loops are forbidden by temporal constraints (7) and (8). Each package starts its tour at time zero (11) and finishes at time *MKSP* (12) and two packages cannot meet at the same node at the same time due to constraint (13). Hence each solution to the above constraint satisfaction problem defines conflict free paths for all packages.

4 Path Model

The disjunctive non-overlap constraint (13) from the *Flow model* is 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 have been proposed [11]. Hence it seems natural to exploit such constraints in a model, where the presence of a package at a node is modeled as an activity. These activities must be connected via temporal constraint to define a path from origin to destination.

Formally, for each package $p \in P$ and each node $x \in V$, we introduce an activity $N[x, p]$ describing time that the package p spends in the node x. We denote $StartOf(N[x, p])$ the start time of the activity - it corresponds to $InT[x, p]$ in the Flow model - and similarly $EndOf(N[x, p])$ denotes the end time of activity corresponding to $OutT[x, p]$ in the Flow model. The start time of activity corresponding to

the origin of the package is set to zero, while the end time of activity corresponding to the destination of the package is set to $MKSP$, which is the makespan of the schedule:

$$StartOf(N[orig(p), p]) = 0 \tag{14}$$
$$EndOf(N[dest(p), p]) = MKSP. \tag{15}$$

4.1 The Path Part

To model the path from origin to destination, we will use a double-link model describing predecessors and successors of activities. The real path will be completed to form a loop by assuming that the origin directly follows the destination. The activities (nodes) that are not used in the path will form self-loops (the node will be its own predecessor and successor).

Formally, for each package $p \in P$ and for each node $x \in V$ we will use two variables $Prev[x, p]$ and $Next[x, p]$ describing the predecessor and successor of node x on the path of package p. The domain of the variable $Prev[x, p]$ consists of all nodes y such that $(y, x) \in E$ plus the node x. Similarly, the domain of variable $Next[x, p]$ consists of nodes z such that $(x, z) \in E$ plus the node x. To ensure that the variables are instantiated consistently, we introduce the constraint:

$$Prev[x, p] = y \Leftrightarrow Next[y, p] = x. \tag{16}$$

To close the loop, we will use the following constraints:

$$Prev[orig(p), p] = dest(p) \tag{17}$$
$$Next[dest(p), p] = orig(p). \tag{18}$$

It remains to connect information about the path with the activities over the path, namely to properly connect times of the activities so they are ordered correctly in time. This will be realized by the constraint:

$$EndOf(N[x, p]) + w(x, Next[x, p]) = StartOf(N[Next[x, p], p]), \tag{19}$$

where $w(x, y)$ is the length of arc from x to y. We set

$$w(x, x) = -1 \tag{20}$$
$$w(dest(p), orig(p)) = -MKSP. \tag{21}$$

In order to prune the search space, we add for all $x \in V \setminus \{orig(p)\}$ the following constraints:

$$Next[x, p] \neq x \Rightarrow EndOf(N[orig(p), p]) + sp(orig(p), x) \leq StartOf(N[x, p]), \tag{22}$$

and for all $x \in V \setminus \{dest(p)\}$, we add:

$$Next[x, p] \neq x \Rightarrow EndOf(N[x, p]) + sp(x, dest(p)) \leq StartOf(N[dest(p), p]). \tag{23}$$

4.2 The Resource Part

For each node $x \in V$, we add the following constraint encoding that the visits of the node x are not overlapping:

$$NoOverlap(\bigcup_{p \in P} N[x, p]). \tag{24}$$

4.3 Model Soundness

Any solution to the Path constraint model defines a solution of the MAPF problem and vice versa. For each package, each node (activity) has some predecessor and successor and they are defined consistently thanks to constraint (16), i.e., if x is a predecessor of y then y is the successor of x. It means that all nodes of the graph are covered by loops. Moreover, the origin and destination nodes are part of the same loop due to constraints (17) and (18). All other loops must be of length one due to constraints (19) and (20). Note that durations of activities are only restricted to be positive numbers and as regular arcs also have positive lengths, the only way to satisfy the constraints (19) over the loop is to include an arc with a negative length. Only the arcs (x, x) and $(dest(p), orig(p))$ have negative lengths as specified in constraints (20) and (21). Finally, each path starts at time zero (14) and finishes at time *MKSP* (15) and no two paths overlap at any node at any time due to constraint (24). Note that activities that are not used at any path (they are part of loops of length one) are still allocated to unary resource modeling the node. The duration of such activities is one due to constraints (19) and (20). However, as their start and end times are not restricted by bounds 0 and *MKSP*, such activities can be shifted to future (after *MKSP*).

5 Opt Model

The *Path model* uses classical activities. Some of them are used on the packages' paths from origins to destinations, while others are not necessary (those that are part of loops of length one). These are dummy activities that are part of the model as we do not know in advance which activities will be necessary (which nodes will be visited). In scheduling there exists a concept of *optional activities* that is used to model exactly the same problem. We will exploit optional activities in the *Opt model*. Now, unlike in the Path model, we do not use variables $Next$ and $Prev$ in order to find the path, but the succeeding and preceding nodes will be entailed by whether or not an activity corresponding to the arc and the package is present. All the activities in this model are optional.

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

5.1 The Path Part

The idea is that the path of a package 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 $N^{out}[x,p]$ has the activities $A[x,y,p]$ corresponding to the arcs outgoing from the node x as its children, and symmetrically, $N^{in}[x,p]$ has the activities $A[y,x,p]$ corresponding to the arcs incoming to the node x as its children. Hence each activity $A[x,y,p]$ has two parents: $N^{out}[x,p]$ and $N^{in}[y,p]$ as the arc (x,y) is an outgoing arc for node x and an incoming arc for node y.

Formally, for each package $p \in P$, the following logical constraints are introduced:

$$PresenceOf(N[orig(p),p]) = 1 \tag{25}$$

$$PresenceOf(N[dest(p),p]) = 1 \tag{26}$$

$$PresenceOf(N^{in}[orig(p),p]) = 0 \tag{27}$$

$$PresenceOf(N^{out}[dest(p),p]) = 0 \tag{28}$$

$$\forall x \in V \setminus \{orig(p)\} : PresenceOf(N[x,p]) \Leftrightarrow PresenceOf(N^{in}[x,p]) \tag{29}$$

$$\forall x \in V \setminus \{dest(p)\} : PresenceOf(N[x,p]) \Leftrightarrow PresenceOf(N^{out}[x,p]) \tag{30}$$

$$\forall x \in V \setminus \{orig(p)\} : Alternative(N^{in}[x,p], \bigcup_{(y,x)\in E} A[y,x,p]) \tag{31}$$

$$\forall x \in V \setminus \{dest(p)\} : Alternative(N^{out}[x,p], \bigcup_{(x,y)\in E} A[x,y,p]) \tag{32}$$

The constraint *Alternative* enforces that 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; otherwise no activity from the set is present. In addition, it ensures that the start and end times of the present activities are equivalent. Since this implication goes only in one direction, we have to impose the following constraints in order to find the path:

$$\forall (x,y) \in E : PresenceOf(A[x,y,p]) \Rightarrow PresenceOf(N^{in}[y,p]) \tag{33}$$

$$\forall (x,y) \in E : PresenceOf(A[x,y,p]) \Rightarrow PresenceOf(N^{out}[x,p]) \tag{34}$$

The processing times of activities $A[x,y,p]$ are fixed to the weights of the arcs $w(x,y)$, whereas the processing times of activities N, N^{out}, and N^{in} are to be found. 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:

$$\forall x \in V \setminus \{orig(p)\} : StartOf(N[x,p]) = EndOf(N^{in}[x,p]) \tag{35}$$

$$\forall x \in V \setminus \{dest(p)\} : EndOf(N[x,p]) = StartOf(N^{out}[x,p]) \tag{36}$$

$$StartOf(N[orig(p),p]) = 0 \tag{37}$$

$$EndOf(N[dest(p),p]) = MKSP \tag{38}$$

Again, in order to prune the search space, we add the following constraints:

$$\forall x \in V \setminus \{orig(p)\} : EndOf(N[orig(p),p]) + sp(orig(p),x) \leq StartOf(N[x,p]) \tag{39}$$

$$\forall x \in V \setminus \{dest(p)\} : EndOf(N[x,p]) + sp(x,dest(p)) \leq StartOf(N[dest(p),p]) \tag{40}$$

5.2 The Resource Part

Exactly as in the Path model, we need to introduce the constraint precluding the packages from occurring at the same node at the same time, that is, for each node $x \in V$, we add:

$$NoOverlap(\bigcup_{p \in P} N[x, p]) \tag{41}$$

5.3 Model Soundness

The solution of the Opt constraint model consists of selection of activities and their time allocation. The activities corresponding to origins and destinations of packages must be selected due to constraints (25) and (26). The constraints (29)-(34) 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 (27), and for the destination where no outgoing arc is selected due to (28)). No activity outside the path is selected as such activities would have to form a loop due to constraints (29)-(34), but that would violate the temporal constraints (35) and (36). Finally, each path starts at time zero (37) and finishes at time *MKSP* (38) and activities in nodes are not overlapping (41).

6 Experimental Results

We implemented the models in the IBM CP Optimizer version 12.7.1 [5]. The only parameters that we adjusted are `DefaultInferenceLevel`, which was set to `Extended`, and `Workers`, which we set to 1. The experiments were run on a Dell PC with an Intel® Core™ i7-4610M processor running at 3.00 GHz with 16 GB of RAM. We use a cutoff time of 100 seconds per problem instance.

6.1 Implementation Details

For all three models, we compute the all-pairs-shortest-path matrix *sp* using the *Floyd-Warshall* algorithm [4] as the preprocessing phase. We set the lower bound on makespan to be the longest path of the packages' shortest paths from their origins to their destinations, and the upper bound on makespan *UB* is set to be the sum of the shortest paths from the origins to the destinations of all the packages. Further, if for a package $p \in P$ and a node $x \in V$, $sp(orig(p), x) > UB$, it means that the node x cannot be passed through by the package p, and thus we omit creating variables associated with the node x and the package p.

To represent the activities in the Path model and the Opt model, we use the *Interval Variables* of the CP Optimizer, which are tailored for the scheduling problems and support specialized constraints such as Alternative and NoOverlap. The only issue is that the NoOverlap constraint works with non-strict inequalities, whereas if a package leaves a node at time t, another package is allowed to enter the same node no sooner than at time $t + 1$. In fact, the times spent by packages at nodes are mostly zero-length. Hence, the NoOverlap constraint is given a so-called *Transition Distance* matrix

containing all ones, which ensures that the time distances between two consecutive visits of a node are at least one. Consequently, instead of constraint (20) in the Path model, we set $w(x, x) = 0$, and as we omit the constraints (19) between the nodes $dest(p)$ and $orig(p)$, the constraints (21) can be also omitted.

The bounds of the intervals and other time variables are limited using the sp matrix. Note that in the Path model, all the intervals must be scheduled and non-overlapping even when the package is not passing through the associated node, so that we use the time upper bound $UB + |P|$.

As to the implementation of the constraints (19) in the Path model, one option is to use the specialized *Element* constraint. Another option is to use constraints in the form of implications for each possible value of $Next[x, p]$:

$$Next[x, p] = y \Rightarrow EndOf(N[x, p]) + w(x, y) = StartOf(N[y, p])$$

The implications turned out to be much more efficient than using the *Element* constraint so that the implications are used in the experiments.

We also tested the models without the constraints for pruning the search space (9)-(10), (22)-(23), and (39)-(40), which led to increase in average runtime for the Flow, Path, and Opt model roughly by 26 %, 37 %, and 20 %, respectively. For the Path model, we also tried adding the constraints $Next[x, p] = x \Leftrightarrow StartOf(N[x, p]) \geq UB$, which turned out to be counterproductive.

6.2 Problem Instances

The problem instances are simple four connected grid maps with unit-length edges. To ensure interaction between agents, impassable walls are introduced in the grid graph. These walls create two types of graphs - a grid that has an obstacle in the middle that the agents have to go around, and a grid that has a bottleneck that the agents have to squeeze through. To create different complexity of the instances we incrementally increase the grid size from 5 by 5 to 9 by 9 as well as we vary the number of agents from 2 to 9 for each size of the graph.

Different origin and destination positions are also included in the experiments. Both can be either randomly scattered across the whole graph or grouped in one place. This yields four different combinations of origin and destination positions. Each of the instances described above was generated multiple times. In total, we generated 1600 instances.

6.3 The Results

The Figure 1 shows the overall comparison in the form of a cactus graph. It shows the number of problems solved (x-axis) within a given time (y-axis). For simple instances, the Flow model is the best one. Then the middle complexity instances are solved best by the Path model, but the overall winner is the Opt model that can solve the largest number of instances. This is an interesting behavior, in particular, that the Flow model is better than the Path model for simpler and for more complex instances, but not for the middle-complexity instances.

We compared the models also based on parameters of the instances. Recall, that two types of worlds (maps) were generated - one with an obstacle to go around it

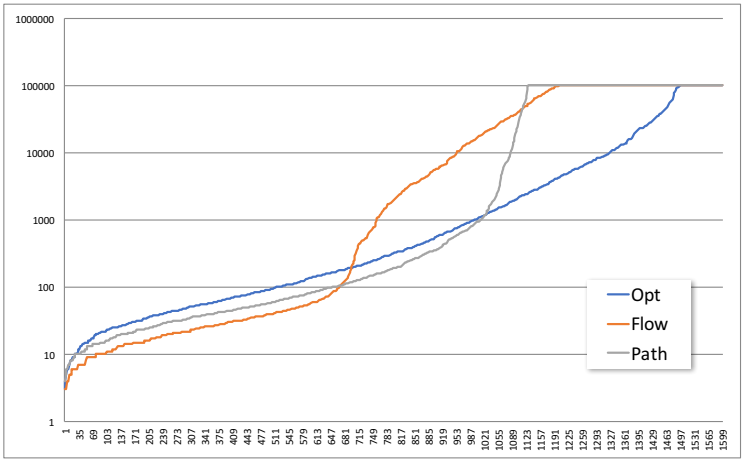

Fig. 1 Dependence of the number of problems solved on time (logarithmic scale; time measured in milliseconds).

and one with a bottleneck that the agents have to squeeze through. Figure 2 shows the comparison in the form a cactus graph. The Opt model is overall the best model independently of the map. The bottleneck maps seem to favor the Flow model over the Path though the trend for the obstacle maps seems similar and maybe if a larger cutoff time is used, the behavior of models will be similar.

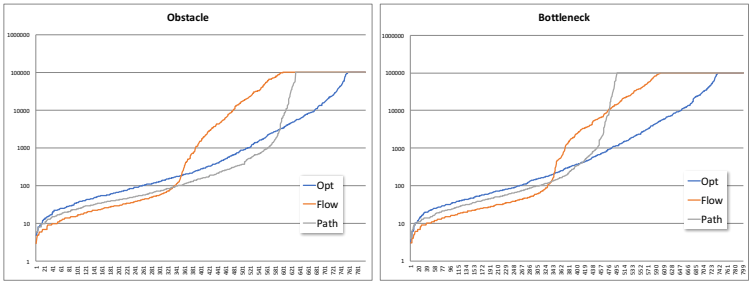

Fig. 2 Dependence of the number of problems solved on time for two types of maps (logarithmic scale; time measured in milliseconds).

We also studied the behavior of models based on the size of instances. The size can be measured by the size of the map or by the number of agents. Figure 3 shows the comparison of models for different sizes of maps. It is clear that for small maps, the Flow model works very well but as the size increases the Path model works better. Again, the Opt model demonstrates the most stable behavior. Regarding the number of agents, it seems that the behavior of models corresponds to the overall behavior and the number of agents does not favor any of the models. Figure 4 shows the comparison for selected numbers of agents.

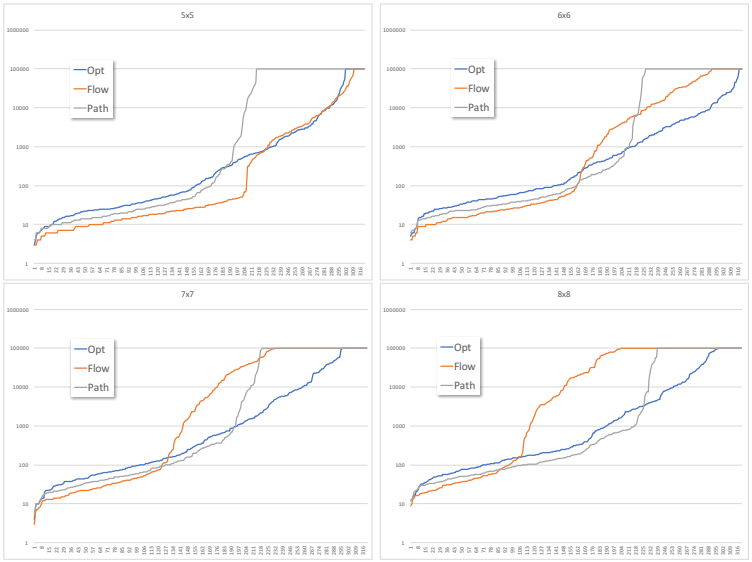

Fig. 3 Dependence of the number of problems solved on time for different sizes of maps (logarithmic scale; time measured in milliseconds).

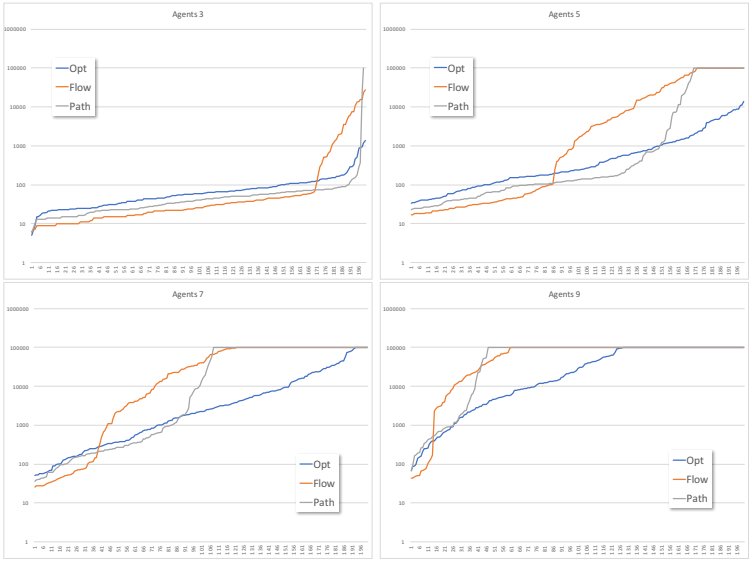

Fig. 4 Dependence of the number of problems solved on time for different numbers of agents (logarithmic scale; time measured in milliseconds).

7 Conclusions

In this paper, we proposed three scheduling models for multi-agent path finding problems. The major motivation was to exploit techniques developed for scheduling problems in a new area, where they have not been used so far. This should allow easier

solving of more realistic problems with various resource and temporal constraints such as non-uniform distances between the nodes and various capacities of nodes (and arcs). The model with optional activities seems the most stable, in particular when the problems are becoming larger. There is an interesting behavior of the Flow model, which is the best for small instances, then it is the worst model for middle-size instances, but the runtime increase seems smaller for larger instances in comparison to other models. This model is more influenced by the size of the graph than the other two models.

There is one significant restriction of the presented models - no agent (package) can return to any node. A future research can study how to extend the models to allow re-visits of the nodes, which is supported by existing solving approaches to MAPF.

Acknowledgements Research is supported by the Czech-Israeli Cooperative Scientific Research Project 8G15027 and by the Czech Science Foundation under the project P202/12/G061.

References

1. Roman Barták, Agostino Dovier, Neng-Fa Zhou, Multiple-Origin-Multiple-Destination Path Finding with Minimal Arc Usage: Complexity and Models. ICTAI 2016: 91-97 (2016)
2. Eli Boyarski, Ariel Felner, Roni Stern, Guni Sharon, David Tolpin, Oded Betzalel, Solomon Eyal Shimony, ICBS: Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding. IJCAI (2015)
3. Esra Erdem, Doga Gizem Kisa, Umut Öztok, Peter Schüller, A General Formal Framework for Pathfinding Problems with Multiple Agents. AAAI (2013)
4. Robert W. Floyd, Algorithm 97: shortest path, Communications of the ACM 5.6 (1962)
5. Philippe Laborie, IBM ILOG CP Optimizer for detailed scheduling illustrated on three problems, International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Springer, Berlin, Heidelberg, 148-162 (2009)
6. Guni Sharon, Roni Stern, Meir Goldenberg, Ariel Felner, The increasing cost tree search for optimal multi-agent pathfinding. Artif. Intell. 195: 470-495 (2013)
7. Guni Sharon, Roni Stern, Ariel Felner, Nathan R. Sturtevant, Conflict-based search for optimal multi-agent pathfinding. Artif. Intell. 219: 40-66 (2015)
8. David Silver, Cooperative Pathfinding. AIIDE 2005: 117-122 (2005)
9. Trevor Scott Standley, Finding Optimal Solutions to Cooperative Pathfinding Problems. AAAI (2010)
10. Pavel Surynek, Towards optimal cooperative path planning in hard setups through satisfiability solving, PRICAI, 564576, (2012)
11. Petr Vilím, Roman Barták, Ondřej Čepek, Extension of O(n log n) filtering algorithms for the unary resource constraint to optional activities, Constraints 10.4: 403-425, (2005)
12. Ko-Hsin Cindy Wang, Adi Botea, MAPP: a Scalable Multi-Agent Path Planning Algorithm with Tractability and Completeness Guarantees. J. Artif. Intell. Res. 42: 55-90 (2011)