Proceedings of the Twelfth International
Symposium on Combinatorial Search (SoCS 2019)

On SAT-Based Approaches for Multi-Agent Path Finding with the Sum-of-Costs Objective

Roman Barták, Jiří Švancara
Charles University, Faculty of Mathematics and Physics
Prague, Czech Republic
bartak@ktiml.mff.cuni.cz

Abstract

Multi-Agent Path Finding (MAPF) deals with the problem of finding collision-free paths for a set of agents. Each agent moves from its start location to its destination location in a shared environment represented by a graph. Reduction-based solving approaches for MAPF, for example, reduction to SAT, exploit a time-expanded layered graph, where each layer corresponds to specific time. Hence, these approaches are natural for minimizing Makespan (the shortest time until all agents reach their destinations). Modeling the other frequently used objective, namely Sum of Costs (SoC; the sum of paths lengths of all agents) is more difficult as the solution with the smallest SoC may not be reached in the time-expanded graph with the smallest Makespan. In this paper we suggest a novel approach to estimate the Makespan, that guarantees the existence of a SoC-optimal solution. We also propose a novel pre-processing technique reducing the number of variables in the SAT model. The approach is empirically compared with an existing reduction-based method as well as with the state-of-the-art search-based optimal MAPF solver.

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) (Silver 2005). The shared environment is often abstracted as a (directed) graph, where agents are initially distributed at some vertices, each agent having a destination vertex to reach. The task is to find a plan of movements for each agent to reach the destination vertex while not being at the same vertex as another agent at the same time. A frequent abstraction assumes that agents are moving synchronously and distances between the vertices are identical. Then, at each time step, each agent either moves to a neighboring vertex or stays in the current vertex. 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.

The MAPF problem has received a lot of attention from the research community since it is highly practically motivated in real life. Some examples, where the MAPF problem is useful, include traffic optimization (Kim, Hirayama, and Park 2014; Michael, Fink, and Kumar 2011), navigation (van den Berg et al. 2009), movement in computer games (Wang and Botea 2008), etc.

Based on the settings where the problem is used, we may have some requirements on the optimality of the found solution. The two most often used cost functions are Makespan (i.e., the minimal time when all agents reached their destination) and Sum of Costs (i.e., the number of all actions performed by all agents). Each of the cost functions has a practical use. Makespan optimal solutions minimize the time to completion of the whole task, even at the cost of taking more actions by some agent(s). On the other hand, Sum of Costs minimizes the total number of actions performed, where actions can be linked to some fuel consumption.

In this paper, we focus on finding the Sum of Costs optimal solutions to MAPF via reduction to a satisfiability (SAT) problem. The reduction approach is more commonly used to finding Makespan optimal solutions, where the translation is more natural. We based our novel models on an existing model that uses reduction to SAT to find the Sum of Costs optimal solutions (Surynek et al. 2016a), and we aim to improve its efficiency by a different approach to estimate the makespan necessary to reach the Sum of Costs optimum.

Problem Definition

Multi-Agent Path Finding (MAPF) deals with the problem of finding a collision-free paths for a set of agents. Formally, a *MAPF instance* is given as a pair (G, A), where $G = (V, E)$ is a graph with vertices V and edges E, and A is a set of agents. Each agent $a_i \in A$ is associated with its starting location $s_i \in V$ and its desired goal location $g_i \in V$ so we can denote the agent as a pair $a_i = (s_i, g_i)$.

The time is discretized to time steps and in each time step every agent can perform either a *move* action, that is, to go to a neighboring vertex or a *wait* action, that is, to stay in its current location. The MAPF task is to find a valid plan – a sequence of actions (or equivalently a sequence of locations) – for each agent.

Let π_i denote a plan for agent a_i, then $\pi_i(j)$ denotes a location where agent a_i is present at a time step j. A valid *solution of MAPF problem* is a plan

$$\pi = \bigcup_{a_i \in A} \pi_i$$

such that the following constraints are satisfied:

1. The plan for each agent is a valid path, that is, $\pi_i(1) = s_i$, $\pi_i(|\pi_i|) = g_i$, and if $\pi_i(j) = v$ and $\pi_i(j+1) = u$ then $(v, u) \in E$ or $v = u$.
2. No two agents occupy one node at the same time, that is, for all pairs of agents a_{i_1} and a_{i_2} at all time steps j it holds that $\pi_{i_1}(j) \neq \pi_{i_2}(j)$.
3. No two agents occupy one edge at the same time, that is, for all pairs of agents a_{i_1} and a_{i_2} at all time steps j it holds that $\pi_{i_1}(j) \neq \pi_{i_2}(j+1) \vee \pi_{i_1}(j+1) \neq \pi_{i_2}(j)$.

Note that this definition allows agents to move along a fully occupied cycle as long as it contains three or more vertices. Other definitions, where a vertex needs to be empty before an agent can move to it, are also used (Kornhauser, Miller, and Spirakis 1984a). This setting is often called pebble motion. All of the techniques presented in this paper can be easily modified to either version; we shall stick to the presented definition.

In addition to having a valid solution, it is often required to find an optimal solution in terms of some cost function. The two most often used functions are *Sum of Costs* (SoC) (Sharon et al. 2011) and *Makespan* (Mks) (Surynek 2014).

Let $|\pi_i|$ denote the number of actions agent a_i performs before reaching g_i for the last time[1]. Then we define Makespan and Sum of Costs as follows:

$$Mks(\pi) = \max_{i=1 \dots n} |\pi_i|$$

$$SoC(\pi) = \sum_{i=1}^{n} |\pi_i|$$

Both of these objective functions have real-life motivation and they are not equivalent. Optimizing either of the Makespan or Sum of Costs objective function can yield different solutions (Surynek et al. 2016b) – see Figure 1 for an example.

It is also important to note that, while there are many polynomial-time algorithms that can find a feasible solution (Kornhauser, Miller, and Spirakis 1984b; de Wilde, ter Mors, and Witteveen 2014; Surynek 2009), the task to find either a Makespan optimal solution or a Sum of Costs optimal solution is an NP-Hard problem (Ratner and Warmuth 1990; Yu and LaValle 2013).

[1]An agent might be forced to leave its goal node to allow pass of another agent. Therefore, the last entrance to the goal node is used to define the plan length. Wait actions are included in the plan.

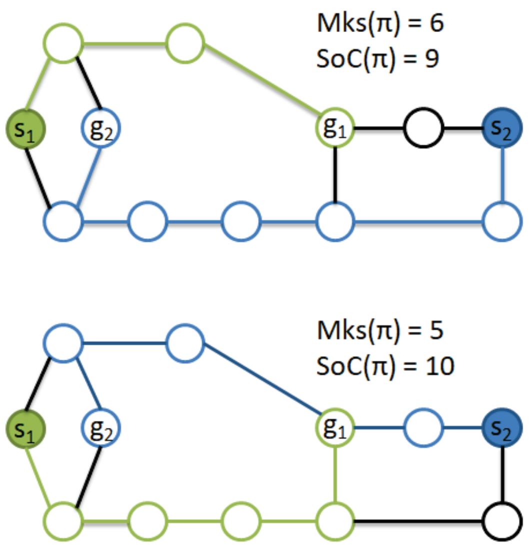

Figure 1: MAPF instance with two agents, where optimizing Sum of Costs and Makespan objective functions yield different plans. The start and goal location of both agents are highlighted as well as the found path for each agent. The top plan minimizes Sum of Costs. The bottom plan minimizes Makespan.

Makespan Optimal Model

As the plan length is unknown in advance, reduction-based approaches to solve planning problems use the method of solving the problem with restricted plan length and, in case of failure, increasing the length limit (Kautz and Selman 1992). This makes it natural to look for shortest plans so we shall also start by describing the classical Makespan optimal translation of MAPF to SAT.

Lets assume that we are looking for a solution to the MAPF problem with makespan T. We define the following two sets of variables: $\forall x \in V, \forall a \in A, \forall t \in \{0, \dots, T\}$: $At(x, a, t)$ meaning that agent a is at vertex x at time step t; and $\forall (x, y) \in E, \forall a \in A, \forall t \in \{0, \dots, T-1\}$: $Pass(x, y, a, t)$ meaning that agent a goes through an edge (x, y) at time step t. An auxiliary edge (x, x) is added to E, thus $Pass(x, x, a, t)$ means that agent a stays at vertex x at time step t. To model the MAPF problem, we introduce the following constraints:

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

$$\forall a \in A : At(g_a, a, T) = 1 \tag{2}$$

$$\forall a \in A, \forall t \in \{0, \dots, T\} : \sum_{x \in V} At(x, a, t) \leq 1 \tag{3}$$

$$\forall x \in V, \forall t \in \{0, \dots, T\} : \sum_{a \in A} At(x, a, t) \leq 1 \tag{4}$$

$$\forall x \in V, \forall a \in A, \forall t \in \{0, \dots, T-1\} :$$

$$At(x, a, t) \implies \sum_{(x,y) \in E} Pass(x, y, a, t) = 1 \tag{5}$$

$$\forall (x,y) \in E, \forall a \in A, \forall t \in \{0,\dots,T-1\}: \quad Pass(x,y,a,t) \implies At(y,a,t+1) \tag{6}$$

$$\forall (x,y) \in E : x \neq y, \forall t \in \{0,\dots,T-1\}: \quad \sum_{a \in A} (Pass(x,y,a,t) + Pass(y,x,a,t)) \leq 1 \tag{7}$$

The constraints (1) and (2) ensure that the starting and goal positions of all agents are valid. The constraints (3) and (4) ensure that each agent occupies at most one vertex and every vertex is occupied by at most one agent. The correct movement in the graph is forced by constraints (5)–(7). In sequence, they ensure that if an agent is in certain vertex, it needs to leave it by one of the outgoing edges (5). If an agent is using an edge, it needs to arrive at the corresponding vertex in the next time step (6). Finally, we forbid two agents to occupy two opposite edges at the same time (no-swap constraint) (7).

To find the optimal makespan, we iteratively increase the makespan T until a satisfiable formula is generated. This clearly provides the makespan-optimal solution as the iterative approach guarantees that no solution with a smaller makespan exists.

To better visualize the above-described constraints, we can use the notion of a *time-expanded graph*. See Figure 2 for reference. We create T copies of the original vertices from graph G and add edges (u_i, v_{i+1}), iff there is an edge $(u,v) \in E$ in the original graph (these edges correspond to move actions), and we also add edges (u_i, u_{i+1}) for each vertex (these edges correspond to wait actions). This creates a time-expanded graph G_T. Agents are moving over this graph in such a way that they start in the 0-th layer and at each timestep they move to the next layer. Constraints (3) and (4) ensure that agents do not collide in vertices. No swap condition is provided by constraint (7).

It is possible to speed up the computation by using a better lower bound for the makespan instead of starting with $T = 1$. A straightforward lower bound is to compute for each agent a_i the shortest path SP_i from agent's start location s_i to agent's goal location g_i. The lower bound for T is then the longest of these shortest paths $LB(Mks) = \max_{i \in A} SP_i$.

Another way to enhance the computation is to do a pre-processing for the variables $At(x,a,t)$. These variables correspond to an agent being present at some location at a time. However, for some locations, we can determine, that the specific agent can not be present at the specific time, because we know where the agent needs to be present at times 0 and T. Specifically for agent a_i, if vertex v is distance d away from start location s_i, we know that the agent a_i can not be present in vertex v in times $0,\dots,(d-1)$ simply because it can not travel the whole distance in time. Similarly, if vertex v is distance d away from goal location g_i, agent a_i can not be present in vertex v in times $T-d+1,\dots,T$.

As a result, we can also do a pre-processing for the variables $Pass(x,y,a,t)$. If we determined that an agent can not be present at vertex v at time t, we can conclude that neither incoming or outgoing edges of that vertex can be used in time t.

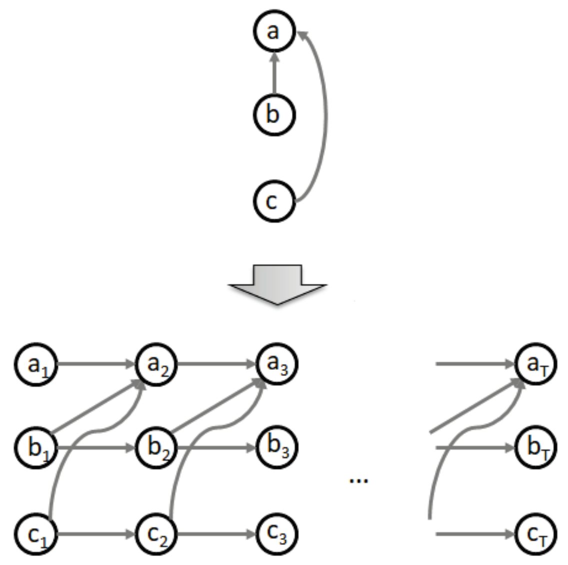

Figure 2: Example of a graph on three vertices being transformed to time-expanded graph with T layers

Sum of Costs Optimal Models

Solving the MAPF problem Makespan optimally via reduction to SAT is quite straightforward in the sense that once we find the correct number of layers in the time-expanded graph, we are guaranteed that this is the Makespan-optimal solution. On the other hand, if we use the same approach to finding the Sum of Costs optimal solution, the first solvable formula is not guaranteed to provide the SoC-optimal solution. Recall Figure 1, indeed we would first find the Makespan optimal solution with $Mks(\pi) = 5$ and $SoC(\pi) = 10$. However, by adding another layer to the time-expanded graph, we are able to find a better solution with respect to the Sum of Costs (which is actually the SoC-optimal solution in this example). In general, it is not clear how many extra layers need to be added to guarantee that the found solution is optimal. This exact problem is addressed in the following two models.

To design models for SoC optimization, we can keep the core model with all of the constraints (1) – (7) as they describe proper paths for agents and they are not related to any particular objective.

In addition, we assume that we can encode a constraint

$$SoC(\pi) \leq UB(SoC) \tag{8}$$

which ensures that the found plan π has Sum of Costs at most $UB(SoC)$, which is some upper bound.

The constraint (8) allows us to use a dichotomic branch-and-bound technique to optimize the value $SoC(\pi)$

$$Minimize\_SoC(LB(SoC), UB(SoC)) \tag{9}$$

which, given a lower bound $LB(SoC)$ and an upper bound $UB(SoC)$, ensures that the found plan π has the minimal Sum of Costs in the interval $\langle LB(SoC), UB(SoC) \rangle$. This can be achieved by iteratively using the constraint (8) and halving the specified interval. Therefore, this technique is just a syntactic shortcut for repetitive usage of constraint (8) with more precise bounds.

Model 1

The pioneering SAT-based model to compute the Sum of Costs optimal solution (Surynek et al. 2016a) focuses on bounding both Makespan (i.e., the number of layers in time-expanded graph) and Sum of Costs together using constraint (8). The bounds are set in such a way that the first solvable formula generated corresponds to the optimal Sum of Costs solution.

Algorithm 1 Model 1

```
function MODEL 1
    $\forall a_i \in A : SP_i = shortest\_path(s_i, g_i)$
    $LB(Mks) = \max_{i \in A} SP_i$
    $LB(SoC) = \sum_{i \in A} SP_i$
    $\delta \leftarrow 0$
    while No Solution do
        solve_MAPF($LB(Mks) + \delta, LB(SoC) + \delta$)
        $\delta \leftarrow \delta + 1$
    end while
end function
```

See Algorithm 1 for description of the model. We first start by finding the shortest path for each agent while ignoring all other agents. The maximum length of these paths is a valid lower bound for Makespan, as was mentioned before. For similar reasons, the sum of the lengths of these paths is a valid lower bound for Sum of Costs – each agent is guaranteed to travel at least this distance. The function solve_MAPF(T, C) generates the SAT model with constraints (1) – (7), Makespan T, and using C as the $UB(SoC)$ in constraint (8).

Let δ be the extra movement that is allowed to the agents. At first, we start with $\delta = 0$ and try to solve the MAPF problem with the lower bounds. If this is possible then we know it must be the optimal solution, since both values (for Makespan and Sum of Costs) are lower bounds. If there is no solution with these restrictions, we increase δ by 1. Notice that this increment adds an extra layer to time-expanded graph and also allows some agent to make one extra step. It has been shown that this is sufficient to find the Sum of Costs optimal plan and furthermore it will be the first solvable formula (Surynek et al. 2016a). We will now present a different proof of soundness of the approach that opens doors for our novel model.

Theorem 1. *If there exists a solution with the Sum of Costs $LB(SoC) + \delta$ then this solution can be found in a time-expanded graph with $LB(Mks) + \delta$ layers.*

Proof. Assume that there exists a solution with the Sum of Costs $LB(SoC) + \delta$, where $LB(SoC) = \sum_{i \in A} SP_i$. It means that this plan uses δ extra actions in addition to $LB(SoC)$ actions that are necessary for each agent to cover its shortest path. These extra actions can be used by any of the agents. If agent i uses k extra actions then we need a time-expanded graph with $SP_i + k$ (or more) layers to model its path. Hence, the largest time-expanded graph is needed if all of the extra actions are used by the agent with the longest of the shortest paths (ie. $\max_{i \in A} SP_i$), which is exactly equal to $LB(Mks)$. It means that a time-expanded graph with $LB(Mks) + \delta$ layers is enough to model all paths. □

Model 1 incrementally increases δ both for Sum of Costs and for Makespan. If it finds a satisfiable model, then according to Theorem 1, this model gives the Sum of Costs optimal solution (if there was a better SoC-optimal solution then it would be found for a smaller makespan).

Model 2

By observing the behavior of Model 1, we noticed that at the end, it may generate larger time-expanded graph than it needs for the Sum of Costs optimal solution (the Sum of Costs optimal solution can be found in a time-expanded graph with a smaller makespan). Moreover, smaller time-expanded graphs are refuted one by one by using too tight upper bound for Sum of Costs. This brought us to the idea of skipping the iterative increase of δ by one and, rather, going directly to the makespan that guarantees the existence of the Sum of Costs optimal solution, even at the expense of overestimating the number of layers of the time-expanded graph required. The new model is called Model 2.

Algorithm 2 Model 2

```
function MODEL 2
    $\forall a_i \in A : SP_i = shortest\_path(s_i, g_i)$
    $LB(Mks) = \max_{i \in A} SP_i$
    $LB(SoC) = \sum_{i \in A} SP_i$
    $\gamma \leftarrow 0$
    while No Solution do
        $SoC \leftarrow$ opt_MAPF($LB(Mks) + \gamma$,
                $LB(SoC), |A| * LB(Mks) + \gamma$)
        $\gamma \leftarrow \gamma + 1$
    end while
    $\delta \leftarrow SoC - LB(SoC)$
    opt_MAPF($LB(Mks) + \delta, LB(SoC), SoC$)
end function
```

See Algorithm 2 for reference. Again, we start by finding the shortest paths for each agent and computing the lower bounds for both Makespan and Sum of Costs. We then find the Makespan optimal solution for the problem just as it was described in the previous section. In this step, there is no restriction on Sum of Costs. This solution gives us some Sum of Costs, that we will use as an upper bound.

Computing δ in this algorithm gives information how many extra steps all of the agents used in the found solution. Following the idea of Theorem 1, $LB(Mks) + \delta$ is the number of layers in the time-expanded graph that guarantees to find the Sum of Costs optimal solution. Finding this optimal solution is the last step in the algorithm.

The function opt_MAPF(T, L, U) generates the SAT model with constraints (1) – (7), Makespan T, and using

L and U as $LB(SoC)$ and $UB(SoC)$ respectively for the dichotomic branch-and-bound search using (9). If the model is satisfiable, the function returns the minimal value of SoC within the interval $\langle L, U\rangle$.

In particular, when finding the Makespan optimal solution, we let the interval be $\langle LB(SoC), |A| * LB(Mks) + \gamma\rangle$. The upper bound lets each agent from A perform as many action as possible in the currently given Makespan. This means that there is no restriction on how many steps each agent can take.

When we are finding the Makespan optimal solution to get an upper bound on Sum of Costs, there is no need to find the optimal solution. In fact, any solution provided by some polynomial sub-optimal solver would suffice. However, these solutions tend to overestimate the solution quite a bit and, therefore, produce much higher δ, resulting in a bigger time-expanded graph. The trade-off is so big that in the algorithm, we find not only a Makespan optimal solution, but from all of the Makespan optimal solution, we select the one with the minimal Sum of Costs.

This approach can be summed in a sequence of three optimizations – first, we optimize the Makespan, then, for that Makespan we optimize the Sum of Costs, and lastly, from these upper and lower bounds, we create a sufficiently large time-expanded graph, on which we once again optimize Sum of Costs, which is guaranteed to be the globaly optimal Sum of Costs of that problem.

Reduction of Used Variables

An improvement that can be applied to both of the described models focuses on decreasing the number of variables that enter the SAT solver. Once again recall the time-expanded graph used to solve the MAPF problem Makespan optimally. A possible way to look at the visualization is that the agents are not moving over one time-expanded graph, but rather, each agent has its own time-expanded graph. These graphs are then interconnected by the constraints that prohibit collisions.

When computing Makespan optimal solutions, we need for all agents to have the time-expanded graph of the same size (same number of layers), since we do not know how much of movement each agent needs to perform. On the other hand, when optimizing Sum of Costs, we do not necessarily need the same number of layers for each agent.

Assume that there are agents a_i and a_j with their respective shortest paths SP_i and SP_j and furthermore SP_i is the longest of all of the shortest paths, while SP_j is much shorter. Now recall Model 1 and Model 2, where optimizing Sum of Costs means, that even agent a_j is allowed to move in time-expanded graph with $SP_i + \delta$ layers, however, there is allowed only δ extra movement for all of the agents combined. Even if all of that movement was used up by agent a_j, there would still be $(SP_i - SP_j) > 0$ extra layers in the time-expanded graph that could not be used. This creates superfluous variables that the SAT solver needs to work with.

The improvement applied on Model 1 and Model 2 then works as follow. When computing function *solve_MAPF*, we create for each agent a_i a separate time-expanded graph TEG_i with $SP_i + \delta$ layers. The time-expanded graphs are interconnected by the constraints (1) – (7) as usual. The last thing to solve is the goal location g_i of agent with smaller SP_i. When an agent reaches its goal, it needs to stay in that goal (i.e. it does no disappear) and all the other agents needs to avoid it. We can simply achieve this by forbidding vertex g_i for all other agents in timesteps greater than $SP_i + \delta$.

Note specifically in Model 2 that we use this enhancement only when finding the Sum of Costs optimal solution (i.e. the second call to solving MAPF in Algorithm 2). We do not want to use this enhancement while finding Makespan optimal solution, because it does not guarantee to find one.

Using this enhancement on Model 1 and Model 2 we create Model 1+ and Model 2+.

Experiments

Implementation

Each of the described models was implemented using the Picat language (Picat language and compiler version 2.2#3), which is a logic-based programming language similar to Prolog. The main advantage, and the reason this tool was used, is that the necessary constraints are easily represented and then automatically translated to a propositional formula. See Figure 3 for an example of code in the Picat language. Notice the similarity between constraints (1) – (7) and the code itself.

In addition, the language provides tools to solve the generated formula and to add the constraint (8) while applying the branch-and-bound technique (9). Moreover, it has been shown that the solver using Picat language is comparable with the state-of-the-art SAT-based MAPF solver (Barták et al. 2017).

Instances

To test the described models, we generated pseudo-random instances. The instances are 4-connected grid maps with increasing sizes from 8×8 to 16×16 with an increment of 2. To introduce some influence between the agents, 20% of the cells in the grid were marked as an impassable obstacle. Representation of one of the maps can be seen in Figure 4.

An increasing number of agents were placed into the created grids. If the grid was of size $W \times W$ then the number of agents was in a range of W to $2W$ with an increment of 2. The start and goal locations were chosen randomly in such a fashion, that no two agents are to start in the same location or to end in the same location. Each of such setting was created five times. Together, this yields 175 unique instances.

Results

All of the generated instances were solved by each of the described model. In addition, we used Conflict Based Search (CBS) (Sharon et al. 2012) solver as a state-of-the-art Sum of Costs optimal solver. The timeout of each instance was set to 600 seconds. Since all of the solvers compute the same problem and the quality of the solution is the same, the main property we are interested in is the time it takes each model to compute it. This result can be seen in Figure 5.

```
import sat.

path_for_delta(N,E,As,K,M) =>
    ME = M - 1,

    B = new_array(M,K,N),
    C = new_array(ME,K,E),

    % Initialize the first and last states
    foreach(A in 1..K)
        (V,FV) = As[A],
        B[1,A,V] = 1,
        B[M,A,FV] = 1
    end,

    B :: 0..1,
    C :: 0..1,

    % Each agent occupies up to one vertex at each time.
    foreach (T in 1..M, A in 1..K)
        sum([B[T,A,V] : V in 1..N]) #=< 1
    end,

    % No two agents occupy the same vertex at any time.
    foreach(T in 1..M, V in 1..N)
        sum([B[T,A,V] : A in 1..K]) #=< 1
    end,

    % if an edge is used in one direction
    % it can not be used in the other direction (no swap)
    foreach(T in 1..ME, EID in 1..E)
     oposit_edges(EID, E, EList),
     sum([C[T,A,W] : A in 1..K, W in EList]) #=< 1
    end,

    % if an agent is in a node
    % it needs to move through one of the edges from that node
    foreach(T in 1..ME, A in 1..K, V in 1..N)
     out_edges(V,E,EList),
        B[T,A,V] #=> sum([C[T,A,W] : W in EList]) #= 1
    end,

    % if agent is using an edge
    % it must arrive to the connected node in next timestep
    foreach(T in 1..ME, A in 1..K, EID in 1..E)
        edge(EID,_,V),
        C[T,A,EID] #=> B[T+1,A,V] #= 1
    end,

    solve(B).
```

Figure 3: Example of a Picat code solving MAPF in the Makespan optimal way.

The instances are ordered based on how long it took to complete them, therefore, the lower the line in the results graph, the better. We can see that CBS is hands-downs fastest for about 90 instances, however, then there is a rapid increase in computation time and it becomes the slowest solver. The four reduction based models seem to be ordered quite well with model Model 2+ being the fastest, closely followed by model Model 2 and Model 1 and Model 1+ being the slowest.

A more detailed analysis of the performance can be seen in Table 1. The most instances solved were by Model 2 and Model 2+, 137 and 139 respectively. All of the other solvers solved similar number of instances – 95 to 97. If we disregard CBS for the moment, we can see that Model 2+ was fastest on most of the instances, while the other reduction based solvers were fastest on only few instances.

To see the difference between the computation times, we compute IPC score[2] for each instance. Solver gets a score of 0 if it did not manage to solve the instance in given time limit. Otherwise, the score is computed as

$$\frac{\textit{min. time}}{\textit{solver time}},$$

where *min. time* is the time it took the fastest solver and *solver time* is the time it took the solver in question. This produce a score in range from 0 to 1, where the bigger the number the better. The scores of all instances were summed and the result is presented in Table 1 as IPC score. We can see that (if we disregard CBS again) the Model 2+ is indeed the most successful closely followed by Model 2. Model 1 and Model 1+ reached similar results.

If we take into consideration CBS as well, we can see a trend similar to the one seen in the graph in Figure 5. If CBS is able to solve the instance in the given time limit, it can solve it faster than the reduction based solvers. This is apparent from the similar number of solved instances by CBS and instances on which CBS is fastest.

On the other hand, the reduction based solvers were able to solve more (or the same) number of instances than CBS. In this sense, Model 2 and Model 2+ are the most successful.

Interesting thing to note is that the enhancement "+" added to Model 1 and Model 2 seems to be useful only for Model 2, while applied on Model 1 it gives no advantage.

[2]The evaluation score was introduced in recent International Planing Competitions, hence the name.

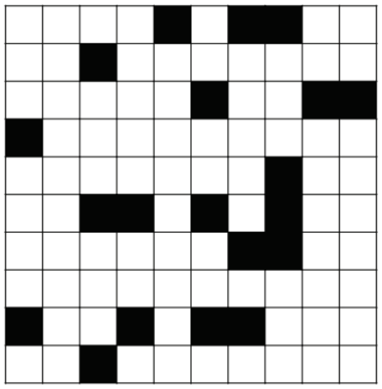

Figure 4: Example of a 10 × 10 grid graph with obstacles used in the experiments.

	M. 1	M. 2	M. 1+	M. 2+	CBS
# of solved	97	137	95	**139**	97
# of fastest	0	4	3	46	**88**
# of fastest (without CBS)	9	8	6	**118**	–
IPC score	16.11	44.56	16.76	57.07	**92.50**
IPC score (without CBS)	57.19	110.54	53.93	**134.29**	–

Table 1: Number of solved instances, number of times the specified solver was fastest, and IPC scores (both in total and when comparing only reduction based solvers). "Model" is abbreviated to "M."

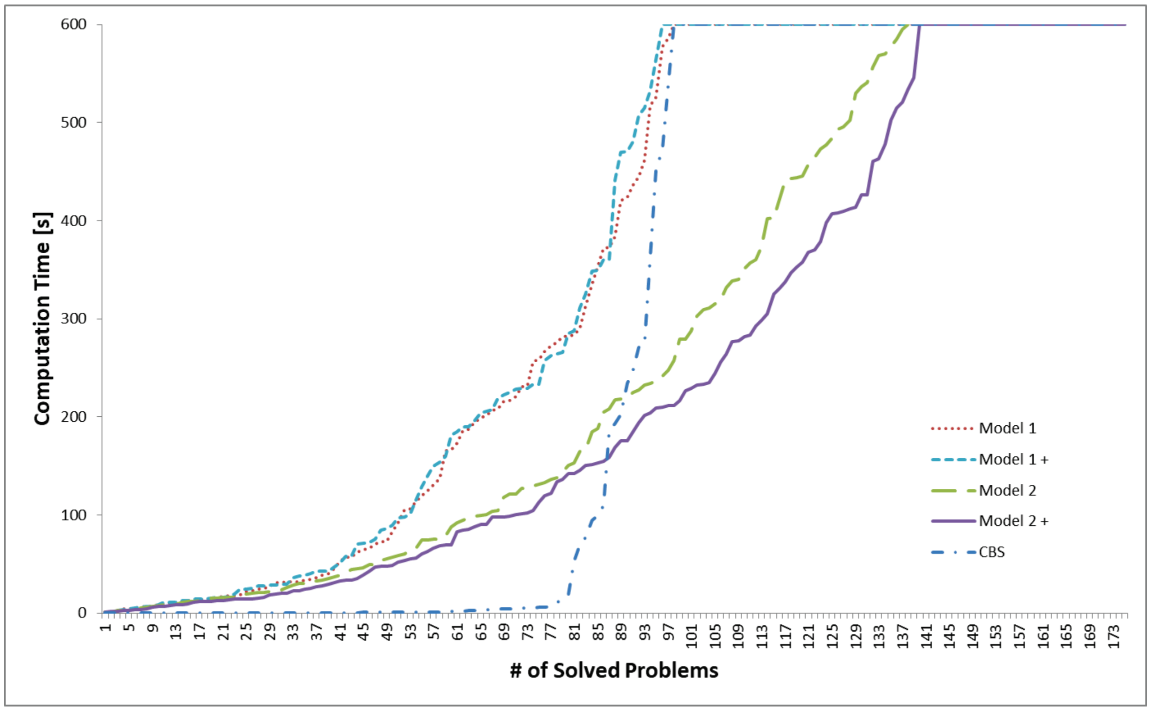

Figure 5: Measured results of the experiments. The y-axis is measured runtime. The x-axis describes the number of solved problems within a given time

Conclusion

In this paper, we studied the Multi-Agent Path Finding problem (MAPF) under the Sum of Costs objective. Translation of MAPF to some other formalism, namely translating to the satisfiability (SAT) problem, is a quite popular approach. While the translation to find the Makespan optimal solution is quite straightforward and natural using the time-expanded graphs, the translation for Sum of Costs is more complicated because we need to limit both Makespan (the number of layers in the time-expanded graph) and Sum of Costs.

We described two different models, one previously known (Surynek et al. 2016a) and one novel one, that provide such encoding. We further present an enhancement for both of the models that limits the number of variables entering the SAT solver. The models were tested against each other and against a search-based state-of-the-art Sum of Costs optimal solver CBS. The results show that while each of the models is fastest on some instances, the most prominent one (in terms of solved instances and computation time) is Model 2+, the novel model with the enhancement.

Another interesting result is that CBS tends to be the fastest solver if it solves an instance, but solves the fewest instances overall in the given time limit

A surprising result is that the enhancement reducing the number of variables entering the SAT solver improved Model 2, while it did not affect Model 1 much. Note that we do not actually remove the variables from the model but we constrain them to get the value zero. The different effect on Models 1 and 2 still requires further investigation. Also, a deeper study of the relationship between parameters of MAPF instances and performance of algorithms is needed to understand when each of these approaches surpasses the others. One may easily see that Model 1 generates (many) unsatisfiable SAT instances and the first satisfiable instance gives the SoC-optimal solution. Somehow complementary, Model 2 generates unsatisfiable instances to find the Makespan-optimal solution (similarly to Model 1 but without bounding SoC), but then it generates (mostly) satisfiable instances to minimize the Sum of Costs.

Acknowledgements

Research is supported by the Czech Science Foundation under the project P103-19-02183S and by the Charles University Grant Agency under the project 90119.

References

Barták, R.; Zhou, N.; Stern, R.; Boyarski, E.; and Surynek, P. 2017. Modeling and solving the multi-agent pathfinding problem in picat. In *29th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2017, Boston, MA, USA, November 6-8, 2017*, 959–966. IEEE Computer Society.

de Wilde, B.; ter Mors, A.; and Witteveen, C. 2014. Push and rotate: a complete multi-agent pathfinding algorithm. *J. Artif. Intell. Res.* 51:443–492.

Kautz, H. A., and Selman, B. 1992. Planning as satisfiability. In *ECAI*, 359–363.

Kim, D.; Hirayama, K.; and Park, G. 2014. Collision avoidance in multiple-ship situations by distributed local search. *JACIII* 18(5):839–848.

Kornhauser, D.; Miller, G. L.; and Spirakis, P. G. 1984a. 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*, 241–250. IEEE Computer Society.

Kornhauser, D.; Miller, G. L.; and Spirakis, P. G. 1984b. 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*, 241–250. IEEE Computer Society.

Michael, N.; Fink, J.; and Kumar, V. 2011. Cooperative manipulation and transportation with aerial robots. *Auton. Robots* 30(1):73–86.

Ratner, D., and Warmuth, M. K. 1990. Nxn puzzle and related relocation problem. *J. Symb. Comput.* 10(2):111–138.

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2011. The increasing cost tree search for optimal multi-agent pathfinding. In Walsh, T., ed., *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, 662–667. IJCAI/AAAI.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2012. Conflict-based search for optimal multi-agent path finding. In Hoffmann, J., and Selman, B., eds., *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.* AAAI Press.

Silver, D. 2005. Cooperative pathfinding. In Young, R. M., and Laird, J. E., eds., *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference, June 1-5, 2005, Marina del Rey, California, USA*, 117–122. AAAI Press.

Surynek, P.; Felner, A.; Stern, R.; and Boyarski, E. 2016a. Efficient SAT approach to multi-agent path finding under the sum of costs objective. In Kaminka, G. A.; Fox, M.; Bouquet, P.; Hüllermeier, E.; Dignum, V.; Dignum, F.; and van Harmelen, F., eds., *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, 810–818. IOS Press.

Surynek, P.; Felner, A.; Stern, R.; and Boyarski, E. 2016b. An empirical comparison of the hardness of multi-agent path finding under the makespan and the sum of costs objectives. In Baier, J. A., and Botea, A., eds., *Proceedings of the Ninth Annual Symposium on Combinatorial Search, SOCS 2016, Tarrytown, NY, USA, July 6-8, 2016.*, 145–147. AAAI Press.

Surynek, P. 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*, 3613–3619. IEEE.

Surynek, P. 2014. Compact representations of cooperative path-finding as SAT based on matchings in bipartite graphs. In *26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014, Limassol, Cyprus, November 10-12, 2014*, 875–882. IEEE Computer Society.

van den Berg, J.; Snoeyink, J.; Lin, M. C.; and Manocha, D. 2009. Centralized path planning for multiple robots: Optimal decoupling into sequential plans. In Trinkle, J.; Matsuoka, Y.; and Castellanos, J. A., eds., *Robotics: Science and Systems V, University of Washington, Seattle, USA, June 28 - July 1, 2009.* The MIT Press.

Wang, K. C., and Botea, A. 2008. Fast and memory-efficient multi-agent pathfinding. In *In ICAPS*, 380–387.

Yu, J., and LaValle, S. M. 2013. Structure and intractability of optimal multi-robot path planning on graphs. In desJardins, M., and Littman, M. L., eds., *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA.* AAAI Press.