Routing and Scheduling in different ways: Abridged Preliminary Report

J. Behrens[1], R. Kaminski[1,2], T. Schaub[1,2], T. C. Son[3], J. Švancara[4] and P. Wanko[2]

[1]*University of Potsdam, Germany*
[2]*Potassco Solutions, Germany*
[3]*New Mexico State University, USA*
[4]*Charles University, Prague, Czech Republic*

Abstract

We present alternative approaches to routing and scheduling in Answer Set Programming (ASP), and explore them in the context of Multi-agent Path Finding. The idea is to capture the flow of time in terms of partial orders rather than time steps attached to actions and fluents. This also abolishes the need for fixed upper bounds on the length of plans. The trade-off for this avoidance is that (parts of) temporal trajectories must be acyclic, since multiple occurrences of the same action or fluent cannot be distinguished anymore. While this approach provides an interesting alternative for modeling routing, it is without alternative for scheduling since fine-grained timings cannot be represented in ASP in a feasible way. This is different for partial orders that can be efficiently handled by external means such as acyclicity and difference constraints. We formally elaborate upon this idea and present several resulting ASP encodings. Finally, we demonstrate their effectiveness via an empirical analysis.

Keywords

Answer Set Programming, Routing, Scheduling, Multi-agent Path Finding

1. Introduction

The ease of Answer Set Programming (ASP [1]) to express reachability has made it a prime candidate for addressing various routing problems, like multi-agent path finding [2], phylogenetic inference [3], wire routing [4], etc. This lightness vanishes, however, once routing is combined with scheduling for expressing deadlines and durations since fine-grained timings cannot be feasibly represented in ASP. This is because ASP [5], just like CP [6] and SAT [7], usually account for time by indexing action and fluent variables with time steps, a technique tracing back to situation calculus [8] and temporal logic [9]. Each time step results in a copy of the problem description. Hence, the finer the granularity of time, the more copies are produced. Although this is usually a linear increase, it eventually leads to a decrease in performance.

We rather capture flows of time by means of partial orders on actions and/or fluents, similar to partial-order planning [10]. In fact, the avoidance of time steps also eliminates the need for upper bounds on temporal trajectories, that is, horizons or makespans. The trade-off for this is that (parts of) temporal trajectories must be acyclic, since multiple occurrences of the same

ASPOCP'23: 16th Workshop on Answer Set Programming and Other Computing Paradigms, London, UK, July, 2023
0000-0002-1361-6045 (R. Kaminski); 0000-0002-7456-041X (T. Schaub); 0000-0003-3689-8433 (T. C. Son); 0000-0002-6275-6773 (J. Švancara); 0000-0003-4986-4881 (P. Wanko)

action or fluent cannot be distinguished without indexing. Moreover, to cease the influence of the granularity of time on the solving process, the idea is to outsource the treatment of partial orders by using hybrid ASP, more precisely, acyclicity and difference constraints. Intuitively, these constraints are used for ordering actions in view of collision evasion. As a matter of fact, we have already applied this technique in several industrial-scale applications involving routing and scheduling, namely, train scheduling [11], system design [12], and warehouse robotics [13]. However, the intricacy of these applications obscured a clear view on the underlying encoding techniques and their formal foundations, which we present in what follows. To simplify this, we apply it to Multi-Agent Path Finding (MAPF [14]), a simple yet highly relevant AI problem.

2. Background

We begin with some preliminaries from graph theory [15]. We consider *graphs* (V, E) where V is a finite set of *vertices* and $E \subseteq V \times V$ is a set of *edges*. A *walk* π in a graph (V, E) is a sequence $(v_i)_{i=0}^n$ of vertices $v_i \in V$ for $0 \leq i \leq n$ such that $(v_i, v_{i+1}) \in E$ for all $0 \leq i < n$. We use $\pi(i) = v_i$ to refer to the vertex at index $0 \leq i \leq n$ of walk π and $|\pi| = n$ to refer to the *length* of the walk. A walk π in a graph (V, E) *leads* from $u \in V$ to $v \in V$ if $\pi(0) = u$ and $\pi(n) = v$. The *vertex set* of a walk π is $V(\pi) = \{\pi(i) \mid 0 \leq i \leq |\pi|\}$; we write $v \in \pi$ for $v \in V(\pi)$. The *index set* of a walk π in a graph (V, E) for a vertex $v \in V$ is $\iota_\pi(v) = \{i \mid v = \pi(i), 0 \leq i \leq |\pi|\}$. A *path* is a walk in which all vertices (and therefore also all edges) are distinct. A *cycle* is a walk in which only the first and last vertices are equal. Note that a cycle with just one vertex is also a path. A *stroll* in a graph (V, E) is a walk in the reflexively closed graph $(V, E \cup \{(v, v) \mid v \in V\})$. That is, a stroll is obtained from a walk in (V, E) by repeating some or none of its vertices (to mimic waiting). A stroll π is *path-like*, if $\iota_\pi(v) = [\min \iota_\pi(v), \max \iota_\pi(v)]$ for all $v \in \pi$. Informally, a stroll is path-like, if dropping all repeated vertices results in a path.

We use the above to define the MAPF problem. In what follows, we consider *simple* graphs (V, E) where $E \subseteq V \times V$ is irreflexive. A *MAPF problem* is a triple (V, E, A) where (V, E) is a simple graph and A is a finite set of agents. Each *agent* $a \in A$ has a *start* vertex $s_a \in V$ and a *goal* vertex $g_a \in V$. We stipulate that all start and all goal vertices are disjoint. That is, for all $a, b \in A$, we require that $a \neq b$ implies $s_a \neq s_b$ and $g_a \neq g_b$. The start and goal vertex of an agent may coincide, that is, we may have $s_a = g_a$ for $a \in A$. An agent can either wait at its current vertex or move to a neighboring one. Hence, we use strolls to capture the movement of agents. A *plan* of length n for a MAPF problem (V, E, A) is a family $\{\pi_a\}_{a \in A}$ of strolls π_a of length n in (V, E) leading from s_a to g_a for all $a \in A$. A plan for a MAPF problem is *path-based* if all its strolls are path-like. We use ι_a as a shortcut for ι_{π_a} whenever it is clear from context that agent a is associated with stroll π_a. A plan $\{\pi_a\}_{a \in A}$ of length n for a MAPF problem (V, E, A) is

1. *vertex conflict-free* if $\pi_a(i) \neq \pi_b(i)$ for all $a, b \in A$ such that $a \neq b$ and $0 \leq i \leq n$,
2. *swap conflict-free* if $\pi_a(i + 1) \neq \pi_b(i)$ or $\pi_a(i) \neq \pi_b(i + 1)$ for all $a, b \in A$ such that $a \neq b$ and $0 \leq i < n$, and
3. *follow conflict-free* if $\pi_a(i + 1) \neq \pi_b(i)$ for all $a, b \in A$ such that $a \neq b$ and $0 \leq i < n$.

A vertex conflict occurs if two agents occupy the same vertex at some point. A swap conflict occurs if two agents traverse the same edge in opposite directions at some point. A follow conflict occurs if an agent enters a vertex another agent just left. The absence of follow conflicts implies the absence of swap conflicts.

Finally, we rely on a basic acquaintance with ASP, and refer the interested reader for details to the literature [16]; the input language of *clingo* is described in the *Potassco User Guide* [17].

3. Routing

Consider the MAPF problem (V, E, A) in Figure 1, delineating in blue and red the movements

$$V = \{v_{i,j} \mid 0 \leq i \leq 1, 0 \leq j \leq 3\} \qquad s_a = v_{0,2}$$
$$E = \{(v_{i,j}, v_{k,l}) \in V_3^2 \mid |i-k| + |j-l| = 1\} \qquad g_a = v_{1,3}$$
$$A = \{a, b\} \qquad s_b = v_{1,0}$$
$$g_b = v_{0,0}$$

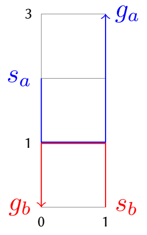

Figure 1: Example MAPF problem together with stroll candidates

of agents a and b all of which traverse vertices $v_{0,1}$ and $v_{1,1}$ on the following strolls:

$$\pi_a^4 = (v_{0,2}, v_{0,1}, v_{1,1}, v_{1,2}, v_{1,3}) \qquad \pi_b^4 = (v_{1,0}, v_{1,1}, v_{0,1}, v_{0,0}, v_{0,0}) \tag{1}$$
$$\pi_a^5 = (v_{0,2}, v_{0,1}, v_{1,1}, v_{1,2}, v_{1,3}, v_{1,3}) \qquad \pi_b^5 = (v_{1,0}, v_{1,0}, v_{1,0}, v_{1,1}, v_{0,1}, v_{0,0}) \tag{2}$$
$$\pi_a^6 = (v_{0,2}, v_{0,1}, v_{1,1}, v_{1,2}, v_{1,3}, v_{1,3}, v_{1,3}) \qquad \pi_b^6 = (v_{1,0}, v_{1,0}, v_{1,0}, v_{1,0}, v_{1,1}, v_{0,1}, v_{0,0}) \tag{3}$$

Consider the following plans: $\{\pi_a^4, \pi_b^4\}$ has length 4 and has a swap conflict, viz. $\pi_a(1) = \pi_b(2)$ and $\pi_a(2) = \pi_b(1)$, plan $\{\pi_a^5, \pi_b^5\}$ has length 5 and has no vertex- and swap-conflicts but a follow conflict, viz. $\pi_a(2) = \pi_b(3)$ (its moves are shown in Figure 2), and plan $\{\pi_a^6, \pi_b^6\}$ has length 6 and is conflict-free. (cf. Figure 3).

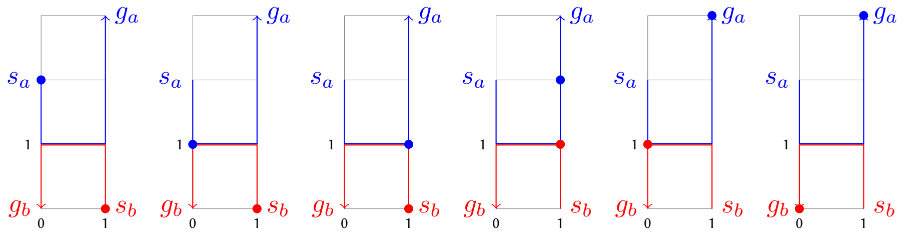

Figure 2: The plan $\{\pi_a^5, \pi_b^5\}$ from (2) avoids vertex and swap conflicts but has a follow-conflict

3.1. Event orderings

Next, we characterize plans for MAPF problems in terms of orders on agent positions.

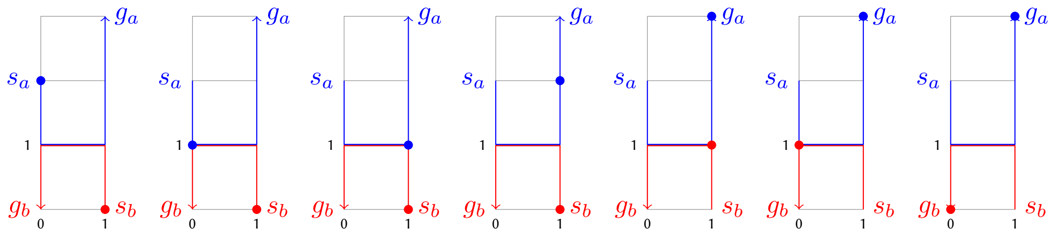

Figure 3: The plan $\{\pi_a^6, \pi_b^6\}$ from (3) avoids vertex, swap, and follow conflicts

An *event set* $\mathcal{E}$ for a MAPF problem (V, E, A) is a set of *events* of form $a@v$ where $a \in A$ and $v \in V$. For illustration, consider the events corresponding to the positions of agents a and b in Figures 2 and 3 (see also the plans in (2) to (3)):

$$\mathcal{E}_a = \{a@v_{0,2}, a@v_{0,1}, a@v_{1,1}, a@v_{1,2}, a@v_{1,3}\} \tag{4}$$
$$\mathcal{E}_b = \{b@v_{1,0}, b@v_{1,1}, b@v_{0,1}, b@v_{0,0}\} \tag{5}$$

Clearly, the combination $\mathcal{E}_a \cup \mathcal{E}_b$ is also an event set.

An *ordered event set* $(\mathcal{E}, \prec)$ is a pair of events $\mathcal{E}$ for a MAPF problem (V, E, A) and a relation $\prec$ establishing a partial order among the events. We use $\prec$ to denote the *cover* of $\prec$. That is, $\prec$ is the smallest relation such that $\prec^* = \prec$ where $\cdot^*$ is the transitive closure. The *restriction* of an ordered event set $(\mathcal{E}, \prec)$ for a MAPF problem (V, E, A) to a single agent $a \in A$ is denoted $(\mathcal{E}_a, \prec_a)$ where $\mathcal{E}_a = \{a@v \in \mathcal{E}\}$ and $\prec_a = \prec \cap (\mathcal{E}_a \times \mathcal{E}_a)$.

Taking a total order whose cover corresponds to the moves of agents a and b in Figures 2 and 3 results in totally ordered event sets $(\mathcal{E}_a, \prec_a)$ and $(\mathcal{E}_b, \prec_b)$ for both agents:

$$a@v_{0,2} \prec_a a@v_{0,1} \prec_a a@v_{1,1} \prec_a a@v_{1,2} \prec_a a@v_{1,3} \tag{6}$$
$$b@v_{1,0} \prec_b b@v_{1,1} \prec_b b@v_{0,1} \prec_b b@v_{0,0} \tag{7}$$

As above, the combination $(\mathcal{E}_a \cup \mathcal{E}_b, \prec_a \cup \prec_b)$ is also an ordered event set.

Definition 1. *An ordered event set* $(\mathcal{E}, \prec)$ *for a MAPF problem* (V, E, A) *is* path-based *if*

a) $(\mathcal{E}_a, \prec_a)$ *is totally ordered with* $\min_{\prec_a} \mathcal{E}_a = a@s_a$ *and* $\max_{\prec_a} \mathcal{E}_a = a@g_a$*, and*

b) $(u, v) \in E$ *for all* $a@u, a@u \in \mathcal{E}$ *with* $a@u \prec_a a@v$*.*

For the MAPF problem (V, E, A) depicted in Figure 1, the ordered event set $(\mathcal{E}_a \cup \mathcal{E}_b, \prec_a \cup \prec_b)$ is path-based since it contains the totally ordered event sets $(\mathcal{E}_a, \prec_a)$ and $(\mathcal{E}_b, \prec_b)$ with least and greatest elements corresponding to the start and goal positions of agents a and b and their covers agree with the edges of the graph (V, E).

Definition 2. *An ordered event set* $(\mathcal{E}, \prec)$ *for a MAPF problem* (V, E, A) *is* conflict-free *if for all events* $a@u, b@u \in \mathcal{E}$ *with* $a \neq b$ *either*

a) there exists an event $a@v \in \mathcal{E}$ *such that* $a@u \prec_a a@v$ *and* $a@v \prec b@u$*, or* *b) there exists an event $b@v \in \mathcal{E}$ such that $b@u \prec_b b@v$ and $b@v \prec a@u$.*

In fact, the ordered event set $(\mathcal{E}_a \cup \mathcal{E}_b, \prec_a \cup \prec_b)$ comprises two conflicts because $a@v_{0,1}, b@v_{0,1}$ as well as $a@v_{1,1}, b@v_{1,1}$ belong to $\mathcal{E}_a \cup \mathcal{E}_b$ but are unrelated by $\prec_a \cup \prec_b$. The adjacency of $v_{0,1}$ and $v_{1,1}$ allows us to resolve this by adding $a@v_{1,2} \prec b@v_{1,1}$ (since this implies $a@v_{1,1} \prec b@v_{0,1}$). That is, the ordered event set

$$(\mathcal{E}_a \cup \mathcal{E}_b, (\prec_a \cup \prec_b \cup \{a@v_{1,2} \prec b@v_{1,1}\})^*) \tag{8}$$

for the MAPF problem from Figure 1 is (path-based and) conflict-free.

The next definition establishes a general relation between path-based ordered event sets.

Definition 3. *Two ordered event sets $(\mathcal{E}, \prec_1)$ and $(\mathcal{E}, \prec_2)$ for a MAPF program (V, E, A) are* compatible *if*

a) $a@u \prec_1 a@v$ iff $a@u \prec_2 a@v$ for all $a@u, a@v \in \mathcal{E}$, and

b) $a@u \prec_1 b@u$ iff $a@u \prec_2 b@u$ for all $a@u, b@u \in \mathcal{E}$ and $a, b \in A$ with $a \neq b$.

In other words, two ordered event sets are compatible, if they resolve all conflicts among agents in the same way.

We consider the ordered event set in (8) together with the following two:

$$(\mathcal{E}_a \cup \mathcal{E}_b, (\prec_a \cup \prec_b \cup \{a@v_{1,3} \prec b@v_{1,1}\})^*) \tag{9}$$

$$(\mathcal{E}_a \cup \mathcal{E}_b, (\prec_a \cup \prec_b \cup \{b@v_{0,0} \prec a@v_{0,1}\})^*) \tag{10}$$

All three ordered event sets are path-based and conflict-free. However, they differ regarding compatibility. The ordered event sets in (8) and (9) are compatible; agent b is waiting longer in (9) before starting to move. The ordered event sets in (8) and (9) are both incompatible with the one in (10); in the former two, agent a starts moving before agent b and in the latter, agent b starts moving before agent a.

An ordered event set $(\mathcal{E}, \prec_1)$ is *smaller* than $(\mathcal{E}, \prec_2)$ if $\prec_1 \subseteq \prec_2$. This allows us to distinguish minimally ordered event sets.

Proposition 1. *For each conflict-free path-based ordered event set for a MAPF problem, there is a unique minimally compatible conflict-free path-based ordered event set for the problem.*

We call such an event set a *minimal* conflict-free path-based ordered event set.

The ordered event set in (8) is smaller than the one in (9). In fact, the one in (8) is a minimal conflict-free path-based ordered event set.

Proposition 2. *We have an onto relationship between plans and ordered event sets for MAPF problems:*

a) for each conflict-free path-based plan, there exists exactly one minimal conflict-free path-based ordered event set, and *b) for each minimal conflict-free path-based ordered event set, there exists at least one conflict-free path-based plan.*

This onto relationship underlines the role of conflict-free path-based ordered event sets as an abstraction of conflict-free path-based plans; the essence lies in the order of events, no matter the continuance in a position. For example, the conflict-free path-based plan $\{\pi_a^6, \pi_b^6\}$ from (3) induces the minimal conflict-free path-based ordered event set in (8) and vice versa. The same applies to extensions of $\{\pi_a^6, \pi_b^6\}$ repeating vertices, as long as they preserve the order in (8).

Let us now detail the constructions underlying Proposition 2.a) and 2.b).

Given a conflict-free path-based plan $\{\pi_a\}_{a \in A}$ for a MAPF problem (V, E, A), the corresponding minimal conflict-free path-based ordered event set is the smallest ordered event set $(\{a@\pi_a(i) \mid a \in A, 0 \leq i \leq |\pi_a|\}, \prec)$ satisfying the following conditions:

1. $a@\pi_a(i) \prec a@\pi_a(i+1)$ for all $0 < i \leq n$ with $\pi_a(i) \neq \pi_a(i+1)$, and
2. $a@\pi_a(i+1) \prec b@\pi_b(j)$ for all $a, b \in A$ and $0 \leq i < j \leq n$ with $a \neq b$ and $\pi_a(i) = \pi_b(j) \neq \pi_a(i+1)$.

Observe that the ordered event set in (8) satisfies the above conditions for plan $\{\pi_a^6, \pi_b^6\}$ given in (3). In fact, it is the smallest ordered event set satisfying them.

Given a conflict-free path-based ordered event set $(\mathcal{E}, \prec)$ for a MAPF problem (V, E, A), a conflict-free path-based plan $\{\pi_a\}_{a \in A}$ is constructed as follows. First of all, we define a sequence of mappings from events to time steps: for each $\epsilon \in \mathcal{E}$ and $i > 0$, define

1. $\alpha_0(\epsilon) = 0$
2. $\alpha_i(\epsilon) = \max\{\alpha_{i-1}(\epsilon)\} \cup \{\alpha_{i-1}(\epsilon') + 1 \mid \epsilon' \in \mathcal{E}, \epsilon' \prec \epsilon\}$

Consider a fixed point $\alpha = \alpha_i$ of this construction satisfying $\alpha_i = \alpha_{i+1}$ for some $i \geq 0$. Intuitively, α gives the earliest arrival time of each agent at its traversed locations. With it, we define for each agent $a \in A$ its stroll $\pi_a = (u_i)_{i=0}^n$ as follows:

1. $n = \max\{\alpha(\epsilon) \mid \epsilon \in \mathcal{E}\}$
2. $u_j = v$ for all $a@v \prec a@w$ and all $\alpha(a@v) \leq j < \alpha(a@w)$
3. $u_j = v$ for $a@v = \max_{\prec_a} \mathcal{E}_a$ and all $\alpha(a@v) \leq j \leq n$

The resulting plan $\{\pi_a\}_{a \in A}$ is conflict-free and path-based. Note that there is no shorter plan corresponding to the order since agents move as early as possible.

Consider the construction of arrival time mappings in Table 1 for the ordered event set in (8). Each row corresponds to a mapping, where the first column is the running index i and the remaining ones give the arrival time of an agent at a vertex as indicated by the table header. We obtain that $\alpha = \alpha_6$ is a fixed point. Observe that we can use this mapping to construct the plan $\{\pi_a^6, \pi_b^6\}$ given in (3). To verify, note that $\min \iota_c(v) = \alpha(v@c)$ for all $c \in \{a, b\}$ and $v \in \pi_c^6$.

Table 1
Arrival time mapping construction for the ordered event set in (8)

α_i	agent a					agent b			
i	$v_{0,2}$	$v_{0,1}$	$v_{1,1}$	$v_{1,2}$	$v_{1,3}$	$v_{1,0}$	$v_{1,1}$	$v_{0,1}$	$v_{0,0}$
0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	0	1	1	1
2	0	1	2	2	2	0	2	2	2
3	0	1	2	3	3	0	3	3	3
4	0	1	2	3	4	0	4	4	4
5	0	1	2	3	4	0	4	5	5
6	0	1	2	3	4	0	4	5	6

3.2. Fact format

We represent a MAPF problem (V, E, A) as a set of facts $\Gamma(V, E, A)$ consisting of an atom `vertex`(v) for each vertex $v \in V$, an atom `edge`(u,v) for each edge $(u, v) \in E$, and atoms `agent`(a), `start`(a,s_a), and `goal`(a,g_a) for each agent $a \in A$. We assume that a suitable syntactic representation is chosen for values of italic variables; this representation is set in typewriter in the source code.

We represent plans by predicates `move/3` or `move/4`, depending on whether we use explicit time points or not. In both cases, the first argument identifies an agent, and the second and third an edge. A path or stroll $(v_i)_{i=0}^{n}$ of an agent a is then represented by atoms of form `move`(a,v_{i-1},v_i) or `move`(a,v_{i-1},v_i,i) with $v_{i-1} \neq v_i$ for $0 < i \leq n$, respectively. We use the variant with time points in the next section and drop them in Section 3.4.

3.3. Vanilla Encoding for MAPF

We first present a basic encoding for MAPF following the traditional approach in Answer Set Planning [5], which relies on time steps.

The encoding in Listing 1 computes conflict-free plans $\{\pi_a\}_{a\in A}$ of length n for MAPF problems (V, E, A). The parameter m allows for preventing follow conflicts when set to `fc`. We use a choice rule in Line 1 to generate move candidates. For each time point $1 \leq t \leq n$ and each agent $a \in A$, we choose at most one `move`(a,u,v,t) for an edge $(u, v) \in E$. The selected moves indicate agents a exiting vertex u at time point $t-1$ and arriving at vertex v at time point t. In Lines 3, 4, and 5, we generate candidates for strolls π_a from the start positions s_a of agents $a \in A$ and moves selected in Line 1. Line 3 ensures that $\pi_a(0) = s_a$ encoded by `at`$(a,s_0,$`0`$)$. In the following line, we derive agent positions `at`(a,v,t) from `move`(a,u,v,t) establishing $\pi_a(t) = v$. The last line in the block encodes that an agent stays at its position if it has not been moved. That is, if we have $\pi_a(t-1) = u$ and the agent is not moved at time point t, we obtain $\pi_a(t) = u$ captured by `at`(a,u,t). Note the use of _ in the negative literal. This can be seen as a shortcut for **not** `move(A,U,T)` together with the rule `move(A,U,T) :- move(A,U,V,T)` projecting out variable `V`; the _ must not be interpreted as an anonymous variable here. At this point, the vertices in the stroll candidates π_a are not necessarily connected by edges nor do

Listing 1
Encoding to find bounded length plans for MAPF

```
 1  { move(A,U,V,T): edge(U,V) } <= 1 :- agent(A), T=1..n.

 3  at(A,U,0) :- start(A,U).
 4  at(A,V,T) :- move(A,_,V,T), T=1..n.
 5  at(A,U,T) :- at(A,U,T-1), not move(A,U,_,T), T=1..n.

 7  :- move(A,U,_,T), not at(A,U,T-1).
 8  :- goal(A,U), not at(A,U,n).

10  :- { at(A,U,T) } > 1, vertex(U), T=0..n.
11  :- move(_,U,V,T), move(_,V,U,T).
12  :- at(A,U,T), at(B,U,T+1), A!=B, m=fc.

14  :- { at(A,U,T) } != 1, agent(A), T=1..n.
```

they lead to goal vertices g_a. The next two integrity constraints in Lines 7 and 8 ensure that the candidates π_a are indeed strolls. Connectedness is ensured by the first integrity constraint. We discard candidates whenever the agent is not at the source vertex of a move. For each true atom $\mathtt{move}(a,u,v,t)$, we require $\pi_a(t-1) = u$ by discarding solutions where **not** $\mathtt{at}(a,u,t-1)$ is true. At this point, the stroll candidates are indeed strolls. The only missing piece is to require that they lead to goal vertices. This is addressed by the second integrity constraint ensuring that $\pi_a(n) = g_a$. For now, we have a plan $\{\pi_a\}_{a \in A}$ that is not necessarily conflict-free. In Lines 10, 11, and 12, we ensure that the plan is free of vertex, swap, and follow conflicts, respectively. The first integrity constraint discards plans in which two agents occupy the same vertex at the same time. The next integrity constraint ensures that there are no swap conflicts. Here, we deviate slightly from the definition of swap conflicts. Observe that a swap conflict can only occur if two agents travel between two vertices in opposing directions at the same time point. Thus, we discard solutions with opposing moves. We treat follow conflicts in Line 12 following the definition. Finally, we add a redundant check in Line 14 asserting that any agent must be at exactly one position. This check is meant to improve solving performance.

Proposition 3 (Soundness). *Given a MAPF problem (V, E, A), let P be the union of $\Gamma(V, E, A)$ and the program in Listing 1 for some $n \in \mathbb{N}_0$ (and $m = \mathtt{fc}$).*

If X is an answer set of P, then $\{(v_i \mid \mathtt{at}(a,v_i,i) \in X)_{i=0}^{n}\}_{a \in A}$ is a vertex, swap, (and follow) conflict-free plan for (V, E, A).

Proposition 4 (Completeness). *Given a MAPF problem (V, E, A), let P be the union of $\Gamma(V, E, A)$ and the program in Listing 1 for some $n \in \mathbb{N}_0$ (and $m = \mathtt{fc}$).*

If $\{\pi_a\}_{a \in A}$ is a vertex, swap (and follow) conflict-free plan for (V, E, A), then $\Gamma(V, E, A) \cup \{\mathtt{at}(a,\pi_a(i),i) \mid a \in A, 0 \leq i \leq n\} \cup \{\mathtt{move}(a,\pi_a(i-1),\pi_a(i),i) \mid 0 < i \leq n, \pi_a(i-1) \neq \pi_a(i)\}$ is an answer set of P.

3.4. Ordering Encoding for MAPF

We now present a solution to MAPF reflecting event orderings; it consists of Listing 2, 3, and 4.

Listing 2
Encoding to find candidate paths for MAPF problems

```
1  { move(A,U,V): edge(U,V) } <= 1 :- agent(A), vertex(V).
2  { move(A,U,V): edge(U,V) } <= 1 :- agent(A), vertex(U).
3  :-  move(A,U,_), not start(A,U), not move(A,_,U).
4  :-  move(A,_,U), not  goal(A,U), not move(A,U,_).

6  :- start(A,U), move(A,_,U).
7  :-  goal(A,U), move(A,U,_).
8  :- start(A,U), not  goal(A,U), not move(A,U,_).
9  :-  goal(A,U), not start(A,U), not move(A,_,U).
```

The first encoding in Listing 2 selects atoms $\texttt{move}(a,u,v)$ for agents $a \in A$ and edges $(u, v) \in E$ for a MAPF problem (V, E, A). Given a stable model X of Listing 2, the selected moves for each agent a form a subgraph (V_a, E_a) of (V, E) such that

1. $E_a = \{(u, v) \mid \texttt{move}(a,u,v) \in X\}$,
2. $V_a = \{s_a, g_a\} \cup \{u, v \mid (u, v) \in E_a\}$,
3. $\deg^-(v) = \deg^+(u) = 1$ for all $\texttt{move}(a,u,v) \in X$,
4. $\deg^-(u) = 1$ if $u \neq s_a$ and $\deg^+(v) = 1$ if $v \neq g_a$ for all $\texttt{move}(a,u,v) \in X$,
5. $\deg^-(s_a) = \deg^+(g_a) = 0$, and
6. $\deg^-(g_a) = \deg^+(s_a) = 1$ if $s_a \neq g_a$.

Any vertex on a path different from the start vertex must have an in degree ($\deg^-$) of one in (V_a, E_a); any vertex on a path different from the goal vertex must have an out degree ($\deg^+$) of one in (V_a, E_a); the start and goal vertices must be the start and end of the path. Hence, the subgraph for an agent a consists of exactly one path leading from s_a to g_a and zero or more separate cycles with a length greater than or equal to two.

To see this, let us go over the rules and see how they affect the in and out degrees of vertices. The rules in Lines 1 and 2 generate atoms $\texttt{move}(a,u,v)$ for agents $a \in A$ such that $(u, v) \in E$. Furthermore, they ensure that $\deg^-(v) \leq 1$ and $\deg^+(u) \leq 1$ for all $\texttt{move}(a,u,v)$. We also get $\deg^-(v) \geq 1$ and $\deg^+(u) \geq 1$ for all $\texttt{move}(a,u,v)$. Thus, at this point, we have $\deg^-(v) = \deg^+(u) = 1$ for all $\texttt{move}(a,u,v)$. Lines 3 and 4 ensure that $\deg^-(u) = 1$ if $u \neq s_a$ and $\deg^+(v) = 1$ if $v \neq g_a$ for all $\texttt{move}(a,u,v)$. Note that this uses $\deg^-(u) \leq 1$ and $\deg^+(v) \leq 1$ ensured in Lines 1 and 2. Lines 6 and 7 ensure that $\deg^-(s_a) = \deg^+(g_a) = 0$. Lines 8 and 9 ensure that $\deg^-(g_a) = \deg^+(s_a) = 1$ if $s_a \neq g_a$. Note that this uses $\deg^-(g_a) \leq 1$ and $\deg^+(s_a) \leq 1$ ensured in Lines 1 and 2.

The second encoding in Listing 3 selects atoms `resolve(`a`,`b`,`u`)` in accord with Definition 2. When such an atom is derived, agent a has to depart from vertex u before agent b arrives at u. The rule in Line 4 chooses which of two agents moving to the same vertex has to move first.

Listing 3
Encoding to find candidate event orders for MAPF problems

```
1 resolve(A,B,U) :- start(A,U), move(B,_,U), A!=B.
2 resolve(A,B,U) :-  goal(B,U), move(A,_,U), A!=B.
3 { resolve(A,B,U);
4   resolve(B,A,U) } >= 1 :- move(A,_,U), move(B,_,U), A<B.

6 :- resolve(A,B,U), resolve(B,A,U).
```

The encoding assumes that $s_a \neq s_b$ and $g_a \neq g_b$ for all $a, b \in A$ with $a \neq b$. Thus, a conflict at a start and goal vertex can only arise if another agent moves to such a vertex. The rules in Lines 1 and 2 select the right resolution order at these vertices: agents at their start vertex as well as agents passing through the goal position of another agent have to move first. At this point, there is at least one `resolve` atom for each case in Definition 2. The integrity constraint in Line 6 ensures that there is exactly one. Note that there is exactly one true `move(`a`,`u`,`v`)` for each true `resolve(`a`,`b`,`u`)`. This is obvious for atoms derived by the rule in Line 4. Assume that `resolve(`a`,`b`,`u`)` is derived by the rule in Line 1. Then there are two cases. If $s_a = g_a$, a conflicting atom would be derived in Line 2. If $s_a \neq g_a$, there must be a move for agent a. The same argument can be made for `resolve` atoms derived by the rule in Line 2.

Finally, the encoding in Listing 4 ensures that the `move` and `resolve` atoms from Listings 2 and 3 form a conflict-free path-based ordered event set. The edge directive in Line 1 specifies

Listing 4
Encoding to find conflict-free ordered event sets for MAPF

```
1 #edge ((A,U),(A,V)) : move(A,U,V).
2 #edge ((A,V),(B,U)) : resolve(A,B,U), move(A,U,V).
```

a graph containing edges given by the `move` atoms. We can interpret the tuples (a, u) in the edge right after the **`#edge`** keyword as events $a@u$. Thus for each true `move(`a`,`u`,`v`)` an edge $(a@u, a@v)$ is added to the graph. The solver discards all solutions where this graph contains a cycle. Remember that the encoding in Listing 2 admits graphs with paths and cycles for agents. Thus, at this point, we have ensured that the moves form a path-based ordered event set. The edge directive in Line 2 further extends the above graph. In accord with Definition 2, we add edge $(a@v, b@u)$ for each atom `resolve(`a`,`b`,`u`)` where $a@u \prec_a a@v$ is captured by `move(`a`,`u`,`v`)`. We have seen above that there is exactly one such move. Thus, if the resulting graph is acyclic, then there is a corresponding conflict-free path-based ordered event set.

Next, we define how to obtain orders from answers sets of the above programs.

Definition 4. *Let (V, E, A) be a MAPF problem and X be an answer set of $\Gamma(V, E, A)$ and the programs in Listings 2, 3, and 4.*

The ordered event set $(\mathcal{E}, \prec)$ corresponding to X is the smallest ordered event set such that *a)* $\mathcal{E} = \{a@u, a@v \mid \texttt{move}(a,u,v) \in X\} \cup \{b@u \mid \texttt{resolve}(a,b,u) \in X\}$

b) $a@u \prec a@v$ *for all* $\texttt{move}(a,u,v) \in X$*, and*

c) $a@v \prec b@u$ *for all* $\texttt{resolve}(a,b,u), \texttt{move}(a,u,v) \in X$*.*

Proposition 5 (Soundness). *Let (V, E, A) be a MAPF problem and P be the union of $\Gamma(V, E, A)$ and the programs in Listings 2, 3, and 4.*

If X is an answer set of P, then the ordered event set $(\mathcal{E}, \prec)$ corresponding to X is a minimal conflict-free path-based ordered event set for (V, E, A).

Together with Proposition 2.b), this implies the existence of a corresponding path-based plan, which can be constructed as described at the end of Section 3.1.

Definition 5. *Let (V, E, A) be a MAPF problem and $(\mathcal{E}, \prec)$ be a minimal conflict-free path-based ordered event set for (V, E, A).*

We define the set X of atoms corresponding to $(\mathcal{E}, \prec)$ as the smallest set such that

a) $\Gamma(V, E, A) \subseteq X$,

b) $\texttt{move}(a,u,v) \in X$ *for all* $a@u \prec_a a@v$*, and*

c) $\texttt{resolve}(a,b,u) \in X$ *for all* $a@v \prec b@u$ *and* $a@u \prec_a a@v$ *with* $a \neq b$*.*

Proposition 6 (Completeness). *Let (V, E, A) be a MAPF problem and P be the union of $\Gamma(V, E, A)$ and the programs in Listings 2, 3, and 4.*

If $(\mathcal{E}, \prec)$ is a minimal conflict-free path-based ordered event set for (V, E, A), then the set of atoms corresponding to $(\mathcal{E}, \prec)$ is an answer set of P.

This and Proposition 2.a) implies that each path-based plan of a MAPF problem has a corresponding stable model of the MAPF encodings.

4. Experiments

To evaluate our encoding variants, we built an ASP-based benchmark generator for different types of grid-based MAPF problems,[1] viz. random and room configurations. The instances are created by choosing subgraphs of a square grid graph (V, E) with $V = \{v_{ij} \mid 1 \leq i, j \leq n\}$ and $E = \{(v_{ij}, v_{kl}) \in V \times V \mid |i - k| + |j - l| = 1\}$ for some $n > 0$. The generated graphs are undirected and connected. Agents' start and goal vertices are picked randomly. For room instances, the grid is evenly divided into rooms separated by walls of width one; neighboring rooms are connected by choosing up to one random vertex as a door. For random ones, a certain percentage of the vertices of the square grid graph is chosen.

Instances are built via the following parameters: $10 \leq n \leq 40$ for the size of the underlying square grid graph, between 5 and 30 agents, square rooms of size 1 to 5, and vertices are chosen with a 50% probability for the random instances. In total, we consider 105 instances

[1]`https://github.com/krr-up/mapf-instance-generator`

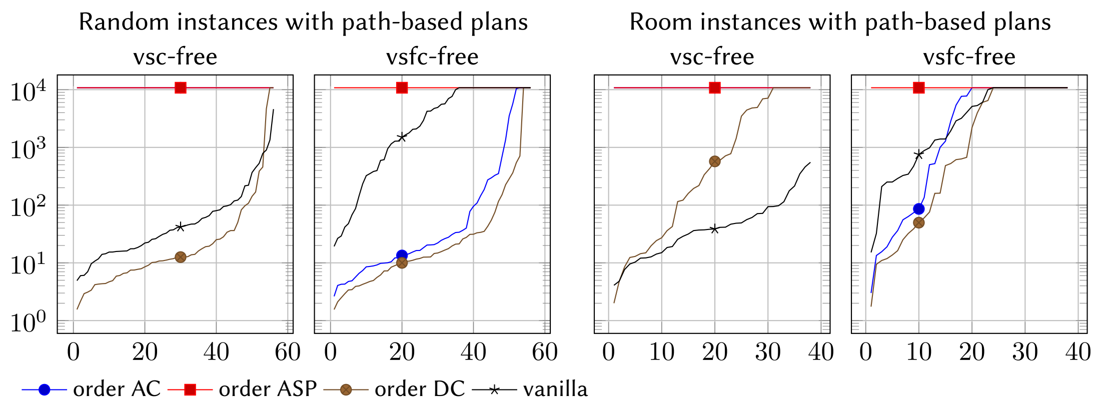

Figure 4: Evaluation of random and room instances with path-based plans

(56 random, 38 room) with at least one conflict-free path-based plan. We ran all instances on a compute cluster with Intel Xeon E5-2650v4@2.9GHz CPUs with 64GB of memory running Debian Linux 10.[2] We used a timeout of 3h and limited the memory to 16GB per instance.

We consider the vanilla encoding (*vanilla*) in Listing 1 and the event ordering encoding (*order AC*) described in Section 3.4. Furthermore, we use a variant of the latter replacing the acyclicity constraints in Listing 4 with difference constraints (*order DC*) and *clingo*[DL] as solver:

```
1  &diff{(A,U)+1}<=(A,V) :- move(A,U,V).
2  &diff{(A,V)+1}<=(B,U) :- resolve(A,B,U), move(A,U,V).
```

Finally, we use a plain ASP encoding (*order ASP*) to check if the generated order is acyclic. Without going into details, we mention that this check results in groundings of cubic size in the number of possible events.

We try to find both conflict-free (*vsfc-free*) and vertex and swap conflict-free plans (*vsc-free*). In the second setting, this means setting the mode m to `fc` for the vanilla encoding. For the DC encoding, this means changing the difference constraint in Line 1 above to use `+0` instead of `+1` and an additional integrity constraint to discard swap conflicts. Note that the integrity constraint does not increase the asymptotic size complexity of the grounding. For the ASP and AC encodings, the second setting involves a more complicated construction, which we do not detail here. Note that even for the AC encodings, this involves a lot more overhead comparable to that of the order ASP setting.

We formulate the following hypotheses for evaluating the generated benchmark on the configurations described above. The vanilla encoding should perform well for small plan lengths (*H1*). The order ASP configuration is not applicable in practice because of the too large grounding (*H2*). The order AC configuration works well for path-based plans in the vsfc-free setting even with large plan lengths (*H3*) but does not work well in the vsc-free setting (*H4*). The order DC configuration works well for path-based plans with large plan lengths independent of the conflict handling (*H5*).

The plots in Figure 4 present cactus plots showing the time (x-axis) to solve an instance (y-

[2] https://www.cs.uni-potsdam.de/bs/research/labs.html#hardware

axis); run times are sorted in ascending order for each curve matching one of the configurations discussed above. Note that we fix the plan length for the vanilla encoding in the vsc-free setting to the optimal length such that an instance is satisfiable. Since we did not have this information available for vsfc-free (and it is hard to compute), we added 20% to the plan length.

We begin with evaluating the vsc-free setting. In line with H1, we observe that the vanilla encoding performs well noting that agents can move to their goal vertices rather directly. Agreeing with H5, order DC performs well too. On room instances it is better than vanilla, which we attribute to the longer plan length necessary to avoid obstacles. As stipulated in H2 and H4, the order ASP and order AC configurations do not work at all.

In the vsfc-free setting, we evaluate the random and room instances separately. We begin with the random instances. Note that order DC performs almost the same as in the vsc-free setting. However, the vanilla configuration becomes much slower if plans have to be follow conflict-free. While this is partly associated with larger plan lengths, we cannot fully explain this behavior. Finally, we confirm H3, noting that order AC performs well here. However, despite the less expressive AC constraints, it performs does not perform as well as order DC. This is due to *clingo*[DL] implementing stronger propagation enabled using option `--propagate=full`. The room instances are much harder in the vsfc-free setting. Not just vanilla but also the event order based configurations are slowed down. We attribute this to short plan length due to H1 and large overhead due to many reachable vertices for the event order settings.

5. Discussion

Although our focus has been on routing, the addition of scheduling boils down to replacing the acyclicity constraints in Listing 4 by difference constraints taking durations into account. Assuming that the `edge` predicate has a third duration argument, we use the following rules:

```
&diff{(A,U)+D}<=(A,V) :- move(A,U,V), edge(U,V,D).
&diff{(A,V)+D}<=(B,U) :- resolve(A,B,U), move(A,U,V), edge(U,V,D).
```

Without entering details, the above should suffice to indicate that our approach generalizes to combined routing and scheduling in a seamless way (as detailed in the upcoming full version of our paper). As mentioned, we have already successfully applied this technique in industrial applications involving routing and scheduling [11, 12, 13]. This paper aims at providing an introduction to the underlying encoding techniques and their formal foundations.

Our approach builds on the characterization of plans in terms of partially ordered event sets. This is similar to partial order planning, where a partial order is maintained among actions [10]. In MAPF, a related idea was implemented using activity constraints from constraint-based scheduling [18]. Also, simple temporal networks were used to add schedules to pre-computed plans [19]. Similarly, in robotics, action dependency graphs were used to avoid online collisions by adding temporal dependencies to existing plans [20]. Finally, it is worth mentioning that it was recently shown that whenever each agent has a predefined path, the problem of deciding if there is a MAPF solution (without any bounds on any cost function) is NP-Hard [21].

Acknowledgments. This work was funded by DFG grant SCHA 550/15.

References

[1] V. Lifschitz, Answer Set Programming, Springer-Verlag, 2019. doi:10.1007/978-3-030-24658-7.

[2] E. Erdem, D. Kisa, U. Öztok, P. Schüller, A general formal framework for pathfinding problems with multiple agents, in: M. desJardins, M. Littman (Eds.), Proceedings of the Twenty-seventh National Conference on Artificial Intelligence (AAAI'13), AAAI Press, 2013, pp. 290–296.

[3] D. Brooks, E. Erdem, S. Erdogan, J. Minett, D. Ringe, Inferring phylogenetic trees using answer set programming, Journal of Automated Reasoning 39 (2007) 471–511.

[4] E. Erdem, V. Lifschitz, M. Wong, Wire routing and satisfiability planning, in: J. Lloyd, V. Dahl, U. Furbach, M. Kerber, K. Lau, C. Palamidessi, L. Pereira, Y. Sagiv, P. Stuckey (Eds.), Proceedings of the First International Conference on Computational Logic (CL'00), volume 1861 of *Lecture Notes in Computer Science*, Springer-Verlag, 2000, pp. 822–836.

[5] V. Lifschitz, Answer set programming and plan generation, Artificial Intelligence 138 (2002) 39–54.

[6] P. Baptiste, P. Laborie, C. Le Pape, W. Nuijten, Constraint-based scheduling and planning, in: F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Elsevier Science, 2006, pp. 761–799. doi:10.1016/S1574-6526(06)80026-X.

[7] J. Rintanen, Planning and SAT, in: A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, volume 185 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2009, pp. 483–504.

[8] J. McCarthy, P. Hayes, Some philosophical problems from the standpoint of artificial intelligence, in: B. Meltzer, D. Michie (Eds.), Machine Intelligence, volume 4, Edinburgh University Press, 1969, pp. 463–502.

[9] J. Kamp, Tense Logic and the Theory of Linear Order, Ph.D. thesis, University of California at Los Angeles, 1968.

[10] E. Sacerdoti, The nonlinear nature of plans, in: Proceedings of the Fourth International Joint Conference on Artificial Intelligence, 1975, pp. 206–214. URL: http://ijcai.org/Proceedings/75/Papers/028.pdf.

[11] D. Abels, J. Jordi, M. Ostrowski, T. Schaub, A. Toletti, P. Wanko, Train scheduling with hybrid ASP, Theory and Practice of Logic Programming 21 (2021) 317–347. doi:10.1017/S1471068420000046.

[12] C. Haubelt, L. Müller, K. Neubauer, T. Schaub, P. Wanko, Evolutionary system design with answer set programming, Algorithms 16 (2023). URL: https://www.mdpi.com/1999-4893/16/4/179. doi:10.3390/a16040179.

[13] D. Rajaratnam, T. Schaub, P. Wanko, K. Chen, S. Liu, T. Son, Solving an industrial-scale warehouse delivery problem with answer set programming modulo difference constraints, Algorithms 16 (2023). URL: https://www.mdpi.com/1999-4893/16/4/216. doi:10.3390/a16040216.

[14] R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. Kumar, R. Barták, E. Boyarski, Multi-agent pathfinding: Definitions, variants, and benchmarks, in: P. Surynek, W. Yeoh (Eds.), Proceedings of the Twelfth International Symposium on Combinatorial Search (SOCS'19), AAAI Press, 2019, pp. 151–159.

[15] E. Bender, S. Williamson, Lists, Decisions and Graphs, University of California, San Diego, 2010.

[16] V. Lifschitz, Answer set planning, in: D. de Schreye (Ed.), Proceedings of the International Conference on Logic Programming (ICLP'99), MIT Press, 1999, pp. 23–37.

[17] M. Gebser, R. Kaminski, B. Kaufmann, M. Lindauer, M. Ostrowski, J. Romero, T. Schaub, S. Thiele, Potassco User Guide, 2 ed., University of Potsdam, 2015. URL: http://potassco.org.

[18] R. Barták, J. Svancara, M. Vlk, A scheduling-based approach to multi-agent path finding with weighted and capacitated arcs, in: E. André, S. Koenig, M. Dastani, G. Sukthankar (Eds.), Proceedings of the Seventeenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS'18), IFAAMAS, 2018, pp. 748–756.

[19] W. Hönig, T. Kumar, L. Cohen, H. Ma, H. Xu, N. Ayanian, S. Koenig, Multi-agent path finding with kinematic constraints, in: A. Coles, A. Coles, S. Edelkamp, D. Magazzeni, S. Sanner (Eds.), Proceedings of the Twenty-sixth International Conference on Automated Planning and Scheduling (ICAPS'16), AAAI Press, 2016, pp. 477–485.

[20] A. Berndt, N. van Duijkeren, L. Palmieri, T. Keviczky, A feedback scheme to reorder a multi-agent execution schedule by persistently optimizing a switchable action dependency graph, CoRR abs/2010.05254 (2020). URL: https://arxiv.org/abs/2010.05254. doi:10.48550/arXiv.2010.05254.

[21] M. Abrahamsen, T. Geft, D. Halperin, B. Ugav, Coordination of multiple robots along given paths with bounded junction complexity, CoRR abs/2303.00745 (2023). URL: https://doi.org/10.48550/arXiv.2303.00745. doi:10.48550/arXiv.2303.00745.