Multi-modal Multi-agent Path Finding with Optimal Resource Utilization

Aysu Bogatarkan[1], Esra Erdem[1], Alexander Kleiner[2], and Volkan Patoglu[1(✉)]

[1] Faculty of Engineering and Natural Sciences, Sabancı University, İstanbul, Turkey
{aysubogatarkan, esraerdem, vpatoglu}@sabanciuniv.edu
[2] Robert Bosch GmbH, Corporate Research, Stuttgart, Germany
alexander.kleiner@de.bosch.com

Abstract. The multi-agent path finding (MAPF) problem is a combinatorial search problem that aims at finding paths for multiple agents (e.g., robots) in an environment (e.g., an autonomous warehouse) such that no two agents collide with each other. We study a general version of MAPF, called mMAPF, that involves further challenges, such as multi-modal transportation modes, a set of waypoints to visit for each agent, and consumption of different types of resources. We introduce a declarative method to solve mMAPF, using answer set programming that provides a flexible formal framework to address all these challenges while optimizing multiple objectives.

AQ1

Keywords: Multi-agent Path Finding · Answer Set Programming · Declarative problem solving · Autonomous warehouses

1 Introduction

Autonomous robot teams are increasingly deployed for industrial applications in warehouses and production. Commonly tasks are within intralogistics in which the goal of the team is to efficiently transport crates and pallets between stationary locations such as packing stations and conveyer entry points. For example, in Amazon's Kiva system or Ali Baba's smart warehouses, there exist multiple robots picking and delivering relevant shelves with products to human workers so that orders can be completed efficiently in time. While these systems heavily depend on engineered infrastructures, i.e., warehouses build from scratch with movable shelves, a substantially larger fraction of intralogistics problems are arising from scenarios with existing infrastructure in which the aforementioned solutions generally do not apply. Challenges for robots in conventional environments are robust methods for Simultaneous Localization and Mapping (SLAM) and Multi-agent Path Finding (**MAPF**). Whereas lidar-based approaches are successfully deployed for solving the SLAM problem today, only inflexible baseline approaches have been deployed for **MAPF** so far.

MAPF problem aims to find a plan for multiple agents to reach their destinations in a certain environment with static obstacles, subject to some constraints on the maximum or the total plan length. Every agent can be considered as a dynamic obstacle for

L. Wang et al. (Eds.): AMP 2020, LNME, pp. 1–12, 2020.
https://doi.org/10.1007/978-3-030-46212-3_24

other agents. Therefore, obstacles and agents are leading to some constraints for the executability of the plan: agents cannot pass through obstacles, and agents cannot collide with each other. While single-agent shortest pathfinding can be solved in polynomial time [7], **MAPF** (with constraints on the plan length) is an intractable problem [19] due to the latter constraint that no two agents can be in the same location at the same time.

MAPF has been investigated in Artificial Intelligence by utilizing various heuristic search algorithms. Some of these studies address time efficiency and minimize the maximum plan length of an agent. Some of them address energy efficiency and minimize the total sum of distance traveled by the agents. However, many realistic conditions observed in warehouses have not been considered in these studies. For instance, the robots' battery levels change as they travel around, and it may be necessary for them to be charged to complete their tasks. Furthermore, some parts of the warehouses, for instance with human occupants or tight passages, may necessitate robots to move slowly to ensure safety.

Along these lines, to handle more realistic autonomous warehouse scenarios, a mathematical model general enough to handle multi-objective optimizations and multi-modal transportation conditions is needed. Furthermore, the computational framework is required to be flexible such that a large set of variations of **MAPF** problems can be addressed.

Motivated by these challenges, we mathematically model a general version of **MAPF** (called **mMAPF** – multi-modal **MAPF** with resources) as a rich graph problem and introduce a flexible method to solve **mMAPF** declaratively.

Our method relies on the declarative programming paradigm Answer Set Programming (ASP) [2, 3, 15, 17, 18]. By utilizing an expressive formal language and efficient solver of ASP, our method can handle the following variations of **MAPF**:

- **Multi-objective optimization:** Our method can find priority-based optimal solutions with respect to time optimization (the minimum plan length), energy optimization (the sum of distances) and their combinations.
- **Waypoints:** Our method can handle scenarios where robots can visit several locations (to pick and deliver different items) on their way to the goal. Note that the determination of the order of items to be collected requires further decision making while computing optimal solutions.
- **Resource constraints:** In addition to time constraints and total energy consumption, our method also considers individual resource such as battery consumption of each robot. As robots travel, their battery levels decrease. There exist charging stations scattered around the warehouses such that robots can charge their batteries. Therefore, while finding optimal solutions, our method also considers battery consumption and the possibility of including charging stations in their itineraries.
- **Multi-modal transportation:** Our method considers different transportation modes, for instance, for regions where robots should move slow or where they are allowed to move fast, while computing optimal solutions.

In the following, once we define our mathematical model for **mMAPF** as a graph problem, we describe how to solve **mMAPF** using ASP. We illustrate an application of our method with an interesting scenario, emphasizing the advantages listed above.

2 Related Work

There are mainly two kinds of **MAPF** solvers: some of them use search-based problem solving (mostly based on a variant of A* search), and some of them use declarative problem solving.

For instance, Silver [21] introduces an incremental method where the paths of agents are computed one by one with A* [13]; once a path is found for an agent, it is considered as an obstacle for other agents. Luna and Bekris [16] propose to compute the paths of agents independently, and then resolve the conflicts (i.e., when two agents collide with each other) with respect to some push-and-swap rules (e.g., there should be at least two free vertices in the graph). Chouhan and Niyogi [5, 6] propose a similar solution where the paths are computed independently; but the conflicts are resolved differently by assigning priorities to agents. Other search-based algorithms, like [8, 14, 24], also compute paths independently; in case a collision occurs, it is resolved by replanning one of the conflicting agents' route. Sharon et al. [20] propose a different method that performs a search on a tree based on the conflicts between agents.

Declarative methods reduce **MAPF** to formal frameworks (e.g., ILP, SAT, ASP) and use general problem solvers to find plans. Yu and Lavalle [25] model **MAPF** as a network flow problem and use an ILP solver to optimize the makespan (the time when the last robot reaches its goal) or the total distance traveled by all robots. Surynek et al. [23, 24] reduce **MAPF** to SAT and use a SAT solver to optimize the makespan or the sum of costs. Erdem et al. [9] model **MAPF** as a logic program and use an ASP solver to optimize the makespan or the distance.

None of the earlier works is applicable to multi-modal transportation and considers utilization of different resources. Our flexible method for **mMAPF** generalizes Erdem et al.'s ASP-based solution for **MAPF** [9] by including different transportation modes that allow priority-based optimization of multiple resource utilizations, and deciding the order of waypoints/charging stations visited by the agents on their ways to the goals.

3 mMAPF: Multi-modal MAPF with Optimal Resource Utilization

mMAPF can be viewed as a generalization of **MAPF** to enable multiple transportation modes and to take resource consumptions of the robots into account.

Let us first introduce some concepts and notation before we define **mMAPF**.

A traversal *f of a path* $P = \langle w_1, w_2, \ldots, w_n \rangle$ in a graph G, where every $w_l \in V$ and every $\langle w_l, w_{l+1} \rangle \in E$ within some time $t \in \mathbb{Z}^+$, is an onto function that maps every nonnegative integer less than or equal to t to a vertex in P or to *intransit*, such that, for every w_l and w_{l+1} in P and for every $x < t$.

- if $mode(\langle w_l, w_{l+1}\rangle) = normal$ and $f(x) = w_l$, then $f(x+1) = w_l$ or $f(x+1) = w_{l+1}$.
- if $mode(\langle w_l, w_{l+1}\rangle) = slow$ and $f(x) = w_l$, then $f(x+1) = w_l$, or $f(x+1) = intransit$ and $f(x+2) = w_{l+1}$.

We denote by $f(P)$ a traversal f of a path P (within time t).

Let f_i and f_j be traversals of two different paths P_i and P_j by agents a_i and a_j respectively, in a graph G within some time t. We say that the traversals f_i and f_j *do not collide with each other* within time t if the following three cases hold:

Case 1. For every time $x, x' \leq t$ such that $f_i(x) \neq intransit$, $f_j(x') \neq intransit$ the following holds: if $f_i(x) = f_j(x')$ then $x \neq x'$. That is, if the same vertex is visited by agents a_i and a_j then it should be visited at different times. Intuitively, no two agents can be at the same location at the same time. The type of collisions eliminated in this case are illustrated in Fig. 1.

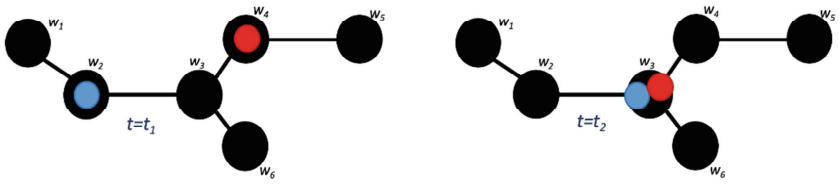

Fig. 1. Case 1. A collision occurs when two agents are at the same location at the same time. In this figure, either $mode(\langle w_2, w_3\rangle) = normal$ and $mode(\langle w_3, w_4\rangle) = normal$, or $mode(\langle w_2, w_3\rangle) = slow$ and $mode(\langle w_3, w_4\rangle) = slow$. In the former case, the collision occurs at time $t_2 = t_1 + 1$, after the agents start moving towards w_3 at time t_1. In the latter case, the collision occurs at time $t_2 = t_1 + 2$, after the agents start moving towards w_3 at time t_1.

Case 2. For every time $x < t$ such that $mode(\langle f_i(x), f_i(x+1)\rangle) = normal$, the following holds: if $f_i(x) = f_j(x+1)$ then $f_i(x+1) \neq f_j(x)$. That is, a normal edge cannot be visited by agents a_i and a_j in reverse directions at the same time. Intuitively, no two agents can swap their locations along a normal edge at the same time. The type of collisions eliminated in this case are illustrated in Fig. 2.

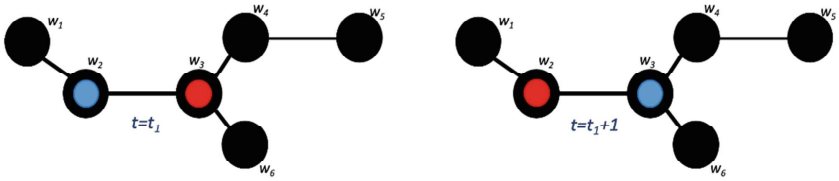

Fig. 2. Case 2. A collision occurs when two agents in the left figure move along the normal edge $\langle w_2, w_3\rangle$ at time t_1 towards each other, as they try to swap their places as in the right.

Case 3. For every time $0 < x \leq t-2$ and for every vertex $u, v \in V$, such that $mode(u,v) = slow$ and $f_i(x) = u, f_i(x+2) = v$,

- if $f_j(x-1) = v$ and $f_j(x) = intransit$, then $f_j(x+1) \neq u$; and
- if $f_j(x) = v$, then $f_j(x+2) \neq u$.

The first condition ensures that no two agents can swap their places along a slow edge if one of them is already in transit. The second condition ensures that no two agents can swap their places along a slow edge, if they are already located at the endpoints of the slow edge. The collisions eliminated in these cases are illustrated in Figs. 3 and 4.

Fig. 3. Case3(a). A collision occurs when two agents try to swap their places along a slow edge $\langle w_1, w_2 \rangle$ and one of them is a bit ahead in transit.

Fig. 4. Case3(b). A collision occurs when two agents located at the endpoints of a slow edge $\langle w_1, w_2 \rangle$ try to swap their places.

Now we can define **mMAPF** as a computational problem, in terms of its input and output, as shown in Fig. 5. Intuitively, graph G characterizes the warehouse where the agents move around, set C describes where charging stations are located in the warehouse, set S describe where agents can be located initially and in the end, set O denotes the parts of the environment covered by the static obstacles, set M denotes transportation modes, function *mode* denotes the parts of the corridors where the agents should travel slowly or where they are allowed to go faster, positive integer n denotes the number of agents, set A denotes the set of n agents, functions *init* and *goal* describe initial locations and goal locations of agents, set B describes the battery levels, function *init_battery* describes the initial battery levels of agents, set W_{ai} describes the set of waypoints for each agent a_i, and positive integer τ is an upper bound on plan lengths.

Given these input, **mMAPF** asks for, for each agent a_i, a path P_i in G from $init(a_i)$ to $goal(a_i)$, a traversal f_i of this path within time $u \leq \tau$, and a battery level function b_i showing how the agent's battery level changes during the traversal. **mMAPF** ensures about P_i that all the waypoints W_{ai} are visited by the agent a_i without colliding any static obstacles O. **mMAPF** ensures about f_i that the agents do not collide with each other while traversing their paths. **mMAPF** ensures about b_i that the agents' batteries have sufficient amount of energy (by charging at stations C, when needed) so that the agents can complete their plans.

mMAPF is an intractable problem: unless P $\neq$ NP, there does not exist a polynomial time algorithm to solve this problem.

Input:
- a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges,
- a set $C \subset V$ of charging nodes,
- a set $S \subset V$ of station nodes,
- a set $O \subseteq V$ of obstacles,
- a set $M = \{slow, normal\}$ of edge modes,
- a function $mode : E \rightarrow M$,
- a set $A = \{a_1, \ldots, a_n\}$ of agents, $(n > 0)$,
- a function $init : A \rightarrow S$,
- a function $goal : A \rightarrow S$,
- a set $B = \{1, .., maxBatteryLevel\}$ of battery levels,
- a function $init\_battery : A \rightarrow B$,
- a set $W_{a_i} \subseteq V$ of waypoints for each agent $a_i \in A$,
- a positive integer τ for maximum plan length.

Output: For every agent $a_i \in A$, for some positive integer $u \leq \tau$,
- a path $P_i = \langle w_{i,1}, \ldots, w_{i,n_i} \rangle$ of length n_i $(n_i \leq u)$,
  - that the agent a_i will follow to reach its goal location from its initial location (i.e., $w_{i,1}$=$init(a_i)$ and w_{i,n_i}=$goal(a_i)$),
  - every waypoint $w_i \in W_{a_i}$ is included in the path P_i,
  - without colliding with any obstacles (i.e., $w_{i,j} \in V \setminus O$)
- a traversal f_i of the path P_i within time u, such that
  - for every other agent $a_j \in A$ with a path P_j and its traversal f_j within u, $f_i(P_i)$ and $f_j(P_j)$ do not collide with each other.
- a battery level function $b_i : \{1, 2, \ldots, u\} \rightarrow B$ where, for every $x < u$,
  - if $f_i(x) \in C$, then $b_i(x + 1) = max(B)$,
  - if $f_i(x) \notin C$, then $b_i(x + 1) = b_i(x) - 1$.

Fig. 5. Problem definition

4 Solving MMAPF Using ASP

Answer Set Programming (ASP) [2, 3, 15, 17, 18] is a knowledge representation and reasoning paradigm that is oriented towards combinatorial search problems as well as knowledge-intensive applications. The idea of ASP is to represent a problem and relevant knowledge as a "program" and to reason about the program by computing its models (called "answer sets" [11, 12]). These models characterize solutions of the problem, and can be computed by "ASP solvers" such as CLINGO [10].

We solve **mMAPF** using ASP by (i) representing it as a program in an ASP language (in this case, the input language of the ASP solver CLINGO), (ii) using an ASP solver (in this case, CLINGO) to find the answer sets for the program, and (iii) extracting the solutions from the answer sets, if there is an answer set.

Let us describe how we represent **mMAPF** in the input language of CLINGO, so that the interested readers can use it directly to experiment with it.

Describing the input and the output. In the following, suppose that `t` is the maximum possible length for a plan (i.e., τ) and `b` is the maximum battery level (i.e., *maxBatteryLevel*). We represent the vertices of G by atoms of the form `vertex(X)` and edges by atoms of the form `edge(X,Y)`. The agents can be located at vertices but also may be in transit, so we define locations explicitly:

```
location(Y) :- vertex(Y).
location(intransit).
```

The transportation modes are described by atoms of the forms `mode(X,Y,n)` or `mode(X,Y,s)` for normal and slow modes of edges respectively. The vertices covered by static obstacles are described by atoms of the form `obstacle(X)`. Agents are described by atoms of the form `agent(A)`. The initial and goal vertices for each agent are defined by atoms of the form `init(A,X)` and `goal(A,Y)` respectively. The initial battery levels are described by atoms of the form `init_battery(A,B)`.

We describe the output by atoms of the forms `plan(A,T,X)` (agent `A` is at location `X` at time `T`) and `batteryLevel(A,T,B)` (agent `A` has battery level `B` at time `T`).

Generating the paths and their traversals. We generate plans of agents recursively. Every agent A starts its plan at time step `0` at its initial location `X`.

```
plan(A,0,X) :- init(A,X), agent(A).
```

Consider any agent `A` who visits location `X` at time `T`. Agent `A` can wait at its current location `X` (if `X` denotes a vertex but not `intransit`) until the next time step `T + 1`:

```
{plan(A,T + 1,X)}1 :- plan(A,T,X), vertex(X), time(T), T < t.
```

Alternatively, the agent can move to the adjacent vertex `Y`. Then agent `A` will be at `Y` at the next time step, if `X` and `Y` have a normal edge between them.

```
{plan(A,T + 1,Y)}1 :- plan(A,T,X), edge(X,Y), mode(X,Y,n),
time(T), T < t.
```

If there is a slow edge between `X` and `Y`, the agent moves to `Y` in two time steps: in the first step, it becomes `intransit` state; in the second step, it becomes at `Y`.

```
{plan(A,T + 1,intransit)}1 :- plan(A,T,X), edge(X,Y),
mode(X,Y,s), time(T), T < t-1.
1{plan(A,T + 2,Y): edge(X,Y), mode(X,Y,s)}1 :-
plan(A,T + 1,intransit), plan(A,T,X), time(T), T < t-1.
```

Validity of paths and their traversals. The paths generated recursively above should satisfy the existence and uniqueness constraints: every agent should be at some location at each time step; every agent cannot be at two different locations at the same time.

```
:- {plan(A,T,Y): location(Y)}0, agent(A), time(T).
:- 2{plan(A,T,Y): location(Y)}, agent(A), time(T).
```

Note that these constraints ensure that there are not forks in a path and the path is connected without any gaps.

Every agent should visit its goal as well as the waypoints.

```
:- goal(A,X), not visit(A,X).
:- waypoint(A,X), not visit(A,X).
```

Here `visit(A,X)` describes which vertices are visited by each agent.

```
visit(A,X) :- plan(A,T,X).
```

If there is an obstacle on vertex `X`, no agent visits it.

```
:- plan(A,T,X), obstacle(X), agent(A), time(T).
```

Collision constraints. No two agents are at the same place at the same time, except when they are both in transit.

```
:- plan(A1,T,X), plan(A2,T,X), agent(A1;A2), A1 < A2,
X! = intransit.
```

This constraint eliminates the types of collisions described in Case 1 (Fig. 1).

Swapping is not allowed along a normal edge.

```
:- plan(A1,T,X), plan(A1,T + 1,Y), plan(A2,T,Y), A1 < A2,
plan(A2,T + 1,X), agent(A1;A2), mode(X,Y,n), T < t.
```

This constraint eliminates the types of collisions described in Case 2 (Fig. 2).

Swapping is not allowed along a slow edge, either. For that, we first define the transition of an agent along a slow edge `(X,Y)` starting at time step `T`:

```
slow(A,T,X,Y) :- plan(A,T,X), plan(A,T + 1,intransit),
plan(A,T + 2,Y), mode(X,Y,s), T < t-1.
```

Then, we ensure that swapping is not allowed on a slow edge.

```
:- slow(A1,T,X,Y), slow(A2,T-1,Y,X), T > 0, T < t-1, A1! = A2.
:- slow(A1,T,X,Y), slow(A2,T,Y,X), T < t-1, A1 < A2.
```

These constraints eliminate the types of collisions described in Case 3 (Figs. 3–4).

Battery levels should remain positive. We define the battery level of an agent recursively starting from its initial battery level.

```
batteryLevel(A,0,B) :- init_battery(A,B), agent(A).
```

At each step `T`, if the agent is not at a charging station, its battery level reduces by 1.

```
batteryLevel(A,T + 1,B1-1) :- batteryLevel(A,T,B1),
plan(A,T,X), not charging(X), agent(A), time(T), T < L,
planLength(A,L).
```

If the agent is at a charging location, its battery level may quickly get to the maximum level or the agent can move forward without charging its battery.

```
1{batteryLevel(A,T + 1,b); batteryLevel(A,T + 1,B1-1)}1 :-
plan(A,T,X), batteryLevel(A,T,B1), charging(X),
agent(A), time(T), T < L, planLength(A,L).
```

Then we ensure that the battery level cannot be less than the minimum level 1.

```
:- batteryLevel(A,T,B), B < 1.
```

Optimizing the plan length. We identify, for each agent, when it reaches the goal (i.e., the time step `L`) ensuring that it visits all of its waypoints on the way.

```
planLength(A,L) :- #max{T: plan(A,T,X), goal(A,X)} = L,
agent(A).
:- plan(A,T,X), waypoint(A,X), planLength(A,L), L < T.
```

Note that `L` denotes the plan length for each agent.

Then we identify the maximum of the plan lengths for all agents.

```
maxPlanLength(M) :- #max{T: planLength(A,T)} = M.
```

and ask CLINGO to minimize it by the following weak constraint:

```
: ∼ maxPlanLength(M). [M@1]
```

Further optimizations. We can ask CLINGO to minimize the total plan lengths to reduce the total energy consumption:

```
: ∼ planLength(A,L). [L@1,A]
```

or we can also ask CLINGO to minimize the total number of charging the batteries:

```
: ∼ batteryLevel(A,T,b). [1@1,A,T]
```

For multi-objective optimization, suppose that we want to minimize the maximum plan length first since we want the tasks to be completed by a given time. Next, we want to minimize the total energy consumption and ensure that robots do not wander around redundantly. Finally, we want to ensure that robots charge their batteries only when needed, by minimizing the number of times they charge. We can express these multiple optimizations by setting the priorities of weak constraints accordingly.

Solving a problem instance using CLINGO. Once the input is described by a set of facts, we can compute an answer set for the program above using the ASP solver CLINGO. After that, we can extract the atoms of the forms `plan(A,T,X)` and `batteryLevel(A,T,B)`, to identify the traversals of agents and how their battery levels change along the way. An example scenario is illustrated in the next section.

5 An Example MMAPF Scenario

Let us illustrate how our method can be applied to solve an example **mMAPF** scenario, illustrated in Fig. 6. In this example, the warehouse consists of three shelf units denoted as obstacles (black cells). The charging stations are located at cells 24 and 27 highlighted by yellow, and the corridor where the agents should go slowly, covers cells 3–8 highlighted by red. There are two robots located at opposite corners of the warehouse. Robot *A1* start with an initial battery level of 10, while robot *A2* has an initial battery level of 8. The maximum battery level is set to 10. Each robot wants to collect some items on its way (denoted by stars that match the color of the relevant robot) and to deliver these items to the opposite corner of the warehouse. Therefore, eventually, they want to swap their places.

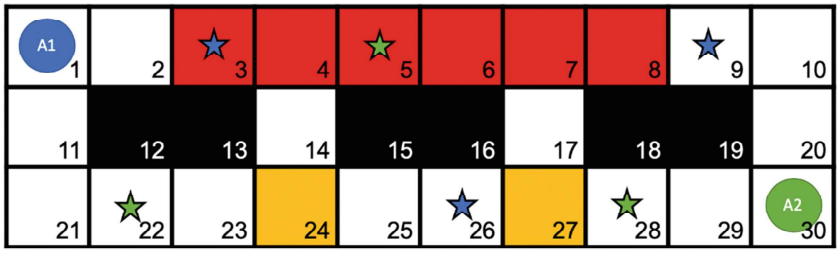

Fig. 6. A **mMAPF** instance. Initially, robot *A1* is located at cell 1 and robot *A2* is located at cell 30. The goal of *A1* is to deliver some items to cell 30, while the goal of *A2* is to deliver some items to cell 1. Each robot's waypoints are shown by stars with the same color as that robot.

A solution for this **mMAPF** instance is computed by CLINGO. Colored paths in Fig. 7 denote paths followed by the robots. Note that each robot visits its waypoints on the way to the goal. The traversals of these paths are shown in Table 1. The traversal of each slow edge (e.g., *A2* moving from cell 6 to 5) takes two time steps. The battery levels of the robots are also shown in this table. The battery level decreases at each time step, unless a robot is at a charging station. The battery level gets to its maximum when robots decide to charge while at a charging station. For instance, *A2*'s battery level increases to 10 when the charging station at cell 27 is visited.

Table 1. The table presents (i) the traversals of the blue and green paths shown in Fig. 7, by the blue robot *A1* and the green robot *A2*, respectively, from time step 0 to 18, and (ii) how the battery levels of these two robots change during these traversals.

Time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A1 location	1	2	3	t	4	14	24	25	26	27	17	7	t	8	9	10	20	30	–
A1 battery	10	9	8	7	6	5	4	3	2	1	10	9	8	7	6	5	4	3	–
A2 location	30	29	28	27	17	7	t	6	t	5	t	4	14	24	23	22	21	11	1
A2 battery	8	7	6	5	10	9	8	7	6	5	4	3	2	1	10	9	8	7	6

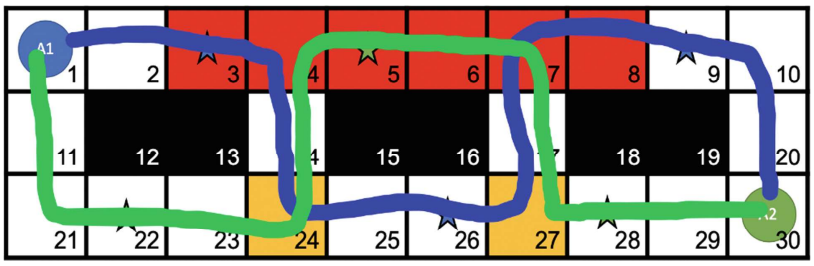

Fig. 7. A solution for the **mMAPF** instance shown in Fig. 6. *A1* follows the blue path while *A2* follows the green path to reach their goals, ensuring they collect items at their waypoints.

6 Conclusion

We introduced a general mathematical model for **mMAPF** problems as a rich graph problem that allows different transportation modes for parts of the environment, priority-based optimization of multiple resource utilizations, and agents to visit waypoints to pick/deliver some items on the way to the goal. Based on this model, we introduced a method to declaratively solve **mMAPF** using the expressive formalism and efficient solvers of ASP. The generality of our model and its declarativeness provide a formal yet flexible framework to investigate alternative solutions in autonomous warehouses. Particularly the flexibility of our framework allows to tailor solutions for highly diverse scenarios comprising various constraints, as they can arise in industrial domains.

Based on the theoretical setup provided in this paper, we plan to extend our studies with a comprehensive experimental evaluation to better understand the scalability of our method, and by considering changes in the environment as investigated in our earlier studies [1].

References

1. Bogatarkan, A., Patoglu, V., Erdem, E.: A declarative method for dynamic multiagent path finding. In: Proceedings of the 5th Global Conference on Artificial Intelligence, pp. 54–67 (2019)
2. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. ACM Commun. **54**(12), 92–103 (2011)
3. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming: an introduction to the special issue. AI Mag. **37**(3), 5–6 (2016)
4. Buccafurri, F., Leone, N., Rullo, P.: Enhancing disjunctive Datalog by constraints. IEEE Trans. Knowl. Data Eng. **12**(5), 845–860 (2000)
5. Chouhan, S.S., Niyogi, R.: DMAPP: a distributed multi-agent path planning algorithm. In: Proceedings of AI, pp. 123–135 (2015)
6. Chouhan, S.S., Niyogi, R.: DiMPP: a complete distributed algorithm for multi-agent path planning. J. Exp. Theor. Artif. Intell. **29**(6), 1129–1148 (2017)

AQ2

7. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numer. Math. **1**(1), 269–271 (1959)
8. Dresner, K.M., Stone, P.: A multiagent approach to autonomous intersection management. J. Artif. Intell. Res. (JAIR) **31**, 591–695 (2008)
9. Erdem, E., Kisa, D.G., Oztok, U., Schueller, P.: A general formal framework for pathfinding problems with multiple agents. In Proceedings of AAAI (2013)
10. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Clingo = ASP + control: Preliminary report. In Proceedings of ICLP (Technical Communications) (2014)
11. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Proceedings of International Logic Programming Conference and Symposium, pp. 1070–1080 (1988)
12. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generat. Comput. **9**, 365–385 (1991)
13. Hart, P.E., Nilsson, N.J., Raphael, B.: Correction to "a formal basis for the heuristic determination of minimum cost paths". SIGART Newslett. **37**, 28–29 (1972)
14. Jansen, R., Sturtevant, N.: A new approach to cooperative pathfinding. In: Proceedings of AAMAS, pp. 1401–1404 (2008)
15. Lifschitz, V.: Answer set programming and plan generation. Artif. Intell. **138**, 39–54 (2002)
16. Luna, R., Bekris, K.E.: Efficient and complete centralized multi-robot path planning. In Proceedings of IROS, pp. 3268–3275 (2011)
17. Marek, V., Truszczynski, M.: Stable models and an alternative logic programming paradigm. In: The Logic Programming Paradigm: A 25-Year Perspective, pp. 375–398. Springer, Heidelberg (1999)
18. Niemela, I.: Logic programs with stable model semantics as a constraint programming paradigm. Ann. Math. Artif. Intell. **25**, 241–273 (1999)
19. Ratner, D., Warmuth, M.K.: Finding a shortest solution for the n $\times$ n extension of the 15-puzzle is intractable. In: Proceedings of AAAI, pp. 168–172 (1986)
20. Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Conflict-based search for optimal multi-agent pathfinding. Artif. Intell. **219**, 40–66 (2015)
21. Silver, D.: Cooperative pathfinding. In: Proceedings of AIIDE, pp. 117–122 (2005)
22. Simons, P., Niemelae, I., Soininen, T.: Extending and implementing the stable model semantics. Artif. Intell. **138**(1), 181–234 (2002)
23. Surynek, P.: On propositional encodings of cooperative path-finding. In: Proceedings of ICTAI, pp. 524–531 (2012)
24. Surynek, P., Felner, A., Stern, R., Boyarski, E.: Efficient SAT approach to multiagent path finding under the sum of costs objective. In: Proceedings of ECAI, pp. 810–818 (2016)
25. Wang, K.-H.C., Botea, A.: Fast and memory-efficient multi-agent pathfinding. In: Proceedings of ICAPS, pp. 380–387 (2008)
26. Yu, J., LaValle, S.M.: Planning optimal paths for multiple robots on graphs. In: Proceedings of ICRA, pp. 3612–3617 (2013)