TPLP **20** (6): 974–989, 2020. © The Author(s), 2020. Published by Cambridge University Press
doi:10.1017/S1471068420000320

Explanation Generation for Multi-Modal Multi-Agent Path Finding with Optimal Resource Utilization using Answer Set Programming∗

AYSU BOGATARKAN and ESRA ERDEM

Sabanci University, Faculty of Engineering and Natural Sciences, Istanbul, Turkey
(*e-mail:* {aysubogatarkan,esraerdem}@sabanciuniv.edu)

submitted 8 August 2020; accepted 10 August 2020; accepted 11 August 2020

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, and subject to some constraints on the lengths of paths. We consider a general version of MAPF, called mMAPF, that involves multi-modal transportation modes (e.g., due to velocity constraints) and consumption of different types of resources (e.g., batteries). The real-world applications of mMAPF require flexibility (e.g., solving variations of mMAPF) as well as explainability. Our earlier studies on mMAPF have focused on the former challenge of flexibility. In this study, we focus on the latter challenge of explainability, and introduce a method for generating explanations for queries regarding the feasibility and optimality of solutions, the nonexistence of solutions, and the observations about solutions. Our method is based on answer set programming.

1 Introduction

Artificial Intelligence (AI) applications are used widely by people with different background and interests. For the success of these applications, two of the important features (and challenges) necessitated by AI methods are flexibility and explainability. A flexible AI method developed to solve a problem can accommodate variations of the problem, and thus can be used to investigate different options by people for a better understanding. An explainable AI method can provide answers to queries about the (in)feasibility and the optimality of solutions. One of the well-studied problems in AI that necessitates solutions for these two challenges is the multi-agent path finding (MAPF) problem.

MAPF problem aims to find plans for multiple agents in an environment without colliding with each other or obstacles. Optimal solutions can be found by optimizing the total plan length of agents or the makespan of the whole plan. These optimization functions can be extended according to the needs of an application. While single-agent shortest pathfinding can be solved in polynomial time (Dijkstra 1959), MAPF with constraints on plan lengths is intractable (Ratner and Warmuth 1986).

Our earlier studies (Bogatarkan et al. 2019; Bogatarkan et al. 2020; Erdem et al. 2013) have addressed the challenge of flexibility for MAPF and its variants, using Answer Set Programming (ASP) (Marek and Truszczyński 1999; Niemelä 1999; Lifschitz 2002)—a logic programming

∗ This work is supported by Tubitak Grant 118E931.

https://doi.org/10.1017/S1471068420000320 Published online by Cambridge University Press

paradigm based on answer sets (Gelfond and Lifschitz 1991; Gelfond and Lifschitz 1988). In this study, we investigate the challenge of explainability for a more general variant of MAPF problem (i.e., mMAPF (Bogatarkan et al. 2020)) applied in a robotics domain (i.e., autonomous warehouses), also utilizing ASP.

In warehouses, the robots' battery levels change as they travel around, and, in some parts of the warehouses, due to human occupancy or tight passages, the robots may need to move slowly to ensure safety. mMAPF (Bogatarkan et al. 2020) is motivated by these realistic conditions on optimal resource use and multi-modal navigation. In mMAPF, the agents have batteries, and their battery levels change while they are moving. In the environment, there are charging stations where the agents can refill their batteries. Some parts of the environment require the agents to slow down, so agents may have different velocities depending on where they are. For example, in some tight areas, the agents may need to move slower, so it takes longer to move from one place to another. Note that these conditions regarding multi-modality make the collision constraints more complicated since agents need to traverse one edge in more than one time step. The restrictions on resource consumption and multi-modality put constraints on the routes of the agents, and thus the overall goal of completing tasks as soon as possible (i.e., minimizing the maximum plan length) or by consuming minimum energy (i.e., minimizing the total plan lengths). Both types of goals can be addressed in ASP (Erdem et al. 2013).

We investigate the challenge of explainability for mMAPF problems, in particular, considering queries about the (in)feasibility and the optimality of solutions, as well as queries about the observations about these solutions. For instance, suppose that a mMAPF solution is being executed in a warehouse. Suppose also that an engineer in this warehouse would like to check whether some modifications of this mMAPF solution would still be feasible or not.

- *Explaining infeasibility or nonoptimality.* Suppose that the modified solution is found infeasible, e.g., using the ASP methods introduced by Bogatarkan et al. (2020). Then, an explanation regarding the infeasibility of the modified solution could be "due to collisions with obstacles or other robots", or "due to low battery-level." An explanation regarding nonoptimality of the modified solution could be "because some more time is needed to complete tasks" or "because some more charging is required".
- *Confirming feasibility and suggesting alternatives.* Suppose that the modified solution is found feasible. Furthermore, a better solution is computed (e.g., where the tasks are completed earlier). Then, in addition to confirming the feasibility of the plan, it would be useful to provide the alternative solutions to the engineer.

In an alternative scenario, suppose that the engineer would like to better understand the mMAPF solution being executed in the warehouse, and asks various queries about it. For such queries, it will be useful to generate explanations using counterfactuals.

- *Explaining why an agent is waiting too long at a location.* Suppose that the engineer observes that the agent is waiting for a while at some location but does not move, and she wants to know why. An explanation could be that "if the agent does not wait at that location for a while, it will collide with another robot." Alternatively, an explanation could be "actually, there is no need for the agent to wait there so long, but it needs to follow a different itinerary such as ... to complete tasks on time" or "actually, there is no need for the agent to wait there so long, but it needs to follow a different itinerary such as ... and will be late a bit."

https://doi.org/10.1017/S1471068420000320 Published online by Cambridge University Press

- *Explaining why an agent is taking a longer path.* Suppose that the engineer observes that the agent is following a path that seems rather long, and she wants to know why. An explanation could be that 'if the agent does not follow that itinerary then it will collide with other robots." Alternatively, an explanation could be "actually, there is no need for the agent to take a long path, but it needs to follow an alternative itinerary such as ...".
- *Explaining why an agent is charging at a distant station.* Suppose that the engineer observes that the agent is charging at a particular station that seems rather distant to its destination, and she wants to know why. An explanation could be that "if the agent does not charge at that station, it will have to wait for other robots and thus will not be able to reach the destination on time." Alternatively, an explanation could be "actually, there is no need for the agent to charge there, but it needs to follow an alternative itinerary such as ...".
- *Explaining why an agent is charging many times.* Suppose that the engineer observes that the agent is charging too many times, and she wants to know why. An explanation could be that "the agent cannot charge less, otherwise it will not be able to reach the destination on time." Alternatively, an explanation could be "actually, the agent can charge less, but it needs to follow an alternative itinerary such as ...".

Such queries and explanations would help the engineer to better understand the strengths and weaknesses of the solution being executed, as well as the limitations of the infrastructure.

With these motivating real life scenarios, we introduce a method to generate explanations for such a variety of queries about mMAPF solutions, using the expressive formalism and efficient solvers of ASP.

2 Preliminaries

mMAPF is a generalization of MAPF to enable multiple transportation modes and to take resource consumptions of the robots into account. We have earlier defined it as a graph problem, and introduced a flexible method to declaratively solve mMAPF and its variants using ASP mMAPF (Bogatarkan et al. 2020). Let us briefly go over the definition, and highlight the parts of the mMAPF ASP program.

mMAPF Problem Definition The input of mMAPF are

- a graph G characterizing the warehouse where agents move around,
- a set C describing where charging stations are located in the warehouse,
- a set S describing where agents can be located initially and in the end,
- a set O denoting the parts of the environment covered by the static obstacles,
- a set M denoting transportation modes (*slow* and *normal*) of edges,
- a function $mode : E \rightarrow M$ denoting the parts of the corridors where the agents should travel slowly or where they are allowed to go faster,
- a positive integer n denoting the number of agents,
- a set A of n agents,
- functions *init* and *goal* describing the initial locations and the goal locations of agents,
- a set B describing battery levels,
- a function $init_battery : A \rightarrow B$ describing the initial battery levels of agents,
- a set $W_{a_i} \subseteq V$ describing the set of waypoints for each agent a_i, and
- a positive integer τ to denote an upper bound on plan lengths.

https://doi.org/10.1017/S1471068420000320 Published online by Cambridge University Press

Given these input, mMAPF asks for a plan: 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. A traversal f of a path $P = \langle w_1, w_2, \ldots, w_n \rangle$ in G is understood as 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 $f(x+2) = w_{l+1}$.

mMAPF ensures

- about P_i that all the waypoints W_{a_i} are visited by the agent a_i without colliding with any static obstacles O,
- about f_i that the agents do not collide with each other while traversing their paths, and
- 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.

For further explanations of the problem definition, we refer the reader to our earlier paper (Bogatarkan et al. 2020, Section 4).

Solving mMAPF using ASP Bogatarkan et al. (2020) solve mMAPF using ASP by (i) representing it as a program in ASP-Core-2 language (Calimeri et al. 2020), (ii) using the ASP solver CLINGO) to find the answer sets for the program, and (iii) extracting the solutions from the answer sets, if there is an answer set.

According to the representation of mMAPF by Bogatarkan et al. (2020), first plans of agents are generated recursively. Every agent `A` starts his plan at time step `0` at his initial location `X`. It can either wait at its current location `X` (if `X` denotes a vertex but not `intransit`) until the next time step `T+1`, or move to the adjacent vertex `Y` via a normal edge or a slow edge. For instance, the traversal of an edge with slow mode is described as follows:

```
{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.
```

The uniqueness and existence of paths are ensured by constraints.

Similarly, the battery level of an agent is defined recursively. At each step `T`, if the agent is not at a charging station, its battery level reduces by 1. If the agent is at a charging location, its battery level may quickly get to the maximum level b or the agent can move forward without charging its battery. This behaviour of an agent at a charging station is described as follows:

```
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).
```

Constraints are added to ensure that a minimum level of battery level.

After that, mMAPF constraints are formulated. For instance, the collision constraint "No two agents are at the same place at the same time, except when they are both in transit." are described as follows:

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

For details of the ASP formulation, we refer the reader to our earlier paper (Bogatarkan et al. 2020, Section 5).

https://doi.org/10.1017/S1471068420000320 Published online by Cambridge University Press

3 Explanation Generation for mMAPF using ASP

Our ASP-based method consists of two parts: the main algorithm (implemented using Python), and the main ASP program Π for mMAPF (represented in ASP-Core-2 language, as described by Bogatarkan et al. (2020)).

Algorithm 1: The main algorithm for generating explanations for mMAPF problems.

```
Input  : a mMAPF instance, a plan for this instance, and a query q of type QW1–QU
Output: An explanation
// Suppose that Π denotes the mMAPF program described by Bogatarkan et al. (2020),
   possibly augmented with some hard constraints due to previous queries
if query q is of type QW1–QP5 then
    Π_h ← Add the relevant hard constraint for q to the mMAPF program Π
    if Π_h has an answer set X then
        Display an explanation, presenting an alternative (better/worse) solution
    else
        Π_w ← Replace the mMAPF constraints in Π_h relevant for q, with the corresponding
          rules and weak constraints
        Y ← Compute an answer set for Π_w
        Display a counterfactual-guided explanation, based on which constraints are
          violated
else
    // query q is of type QU
    Π_w ← Replace the mMAPF constraints in Π with the corresponding rules and weak
      constraints
    Y ← Compute an answer set for Π_w
    Display a counterfactual-guided explanation, based on which constraints are violated
```

The inputs of the main algorithm (Algorithm 1) are a mMAPF instance, a plan for this instance, and a query. Currently, our algorithm supports 14 types of queries about the given plan. Queries QW1–QW4 are about waiting, QC1–QC4 are about charging, QP1–QP5 are about traversals, and QU is about the nonexistence of a solution.

QW1 Why does Agent `a` wait at location `x` (at any time)?
QW2 Why does Agent `a` wait at location `x` at time `s`?
QW3 Why does Agent `a` wait at location `x` at time `s` for `n` steps?
QW4 Why does not Agent `a` wait at location `x` at time `s` for less than `n` steps?
QC1 Why does Agent `a` charge at location `x` (at any time)?
QC2 Why does Agent `a` charge at time `s`?
QC3 Why does Agent `a` charge at location `x` at time `s`?
QC4 Why does not Agent `a` charge less than `m` times?
QP1 Why does not Agent `a` have a plan whose length is less than `l`?
QP2 Why does Agent `a` visit location `x` (at any time)?
QP3 Why does Agent `a` visit location `x` at time `s`?
QP4 Why does Agent `a` move from location `x` to location `y` (at any time)?
QP5 Why does Agent `a` move from location `x` to location `y` at time `s`?
QU Why does not the instance have a solution?

https://doi.org/10.1017/S1471068420000320 Published online by Cambridge University Press

3.1 Explanation Generation for Queries QW1–QP5 about Solutions

Queries QW1–QP5 are associated by the following hard constraints:

```
QW1 :- plan(a,T,x), plan(a,T+1,x), T<t.
QW2 :- plan(a,s,x), plan(a,s+1,x).
QW3 :- plan(a,s,x),..,plan(a,s+n,x).
QW4 :- C = #count{T: plan(a,T,x), plan(a,T+1,x), T<s+n, T>=s}, C >= n.
QC1 :- batteryLevel(a, T+1, b), plan(a, T, x), charging(x).
QC2 :- batteryLevel(a,s+1,b), plan(a,s,X), charging(X).
QC3 :- batteryLevel(a, s+1, b), plan(a, s, x), charging(x).
QC4 :- C = #count{T: batteryLevel(a,T,b), T>0}, C >= m.
QP1 :- planLength(a,L), L>=l.
QP2 :- plan(a,T,x).
QP3 :- plan(a,s,x).
QP4 :- plan(a,T,x), plan(a,T+1,y), edge(x,y), mode(x,y,'n'), T<t-1.
    :- plan(a,T,x), plan(a,T+1,intransit), plan(a,T+2,y),
       edge(x,y), mode(x,y,'s'), T<t.
QP5 :- plan(a,s,x), plan(a,s+1,y), edge(x,y), mode(x,y,'n').
    :- plan(a,s,x), plan(a,s+1,intransit), plan(a,s+2,y),
       edge(x,y), mode(x,y,'s').
```

If the given query is of type QW1–QP5, then the algorithm first checks whether the ASP program Π_h, obtained from Π by adding the relevant hard constraint, has an answer set or not. If the augmented program has an answer set, then alternative plans are extracted from the answer sets and presented to the user with some recommendations. Here as some sample explanations for some queries:

QW1 Actually, Agent `a` does not have to wait at location `x`. Here is an alternative plan: ...
QW3 Actually, Agent `a` does not have to wait at location `x` at time `s` for `n` steps. Here is an alternative plan: ...
QC2 Actually, Agent `a` does not have to charge at time step `s`. Here is an alternative plan: ...
QC4 Actually, Agent `a` can charge less than `m` times. Here is an alternative plan: ...
QP1 Actually, Agent `a` can follow a shorter path whose length is smaller than `l`. Here is an alternative plan: ...
QP5 Actually, Agent `a` does not have to move from location `x` to location `y` at time `s`. Here is an alternative plan: ...

If the given plans are optimal then the explanations can involve further information: "Here is an alternative plan that is shorter: ...."

If the augmented program does not have an answer set, then the given plan is not feasible or optimal. Then the algorithm finds an explanation for why not, i.e., a query of type QU.

3.2 Explanation Generation for Query QU about the Nonexistence of Solutions

If a given plan is found infeasible or nonoptimal, our algorithm tries to identify which constraints relevant to the given question are violated. For that reason, we obtain a new ASP program Π_w from the mMAPF program Π_h by replacing each relevant mMAPF constraint by a set of rules and a weak constraint as follows.

We replace the collision constraint (i.e., no two agents are at the same place at the same time, except when they are both in transit)

https://doi.org/10.1017/S1471068420000320 Published online by Cambridge University Press

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

by the following rules that describe the conditions violating this collision constraint:

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

and the following weak constraints:

```
:~ violate_collision(A1,A2,T,X). [1@7, A1,A2,T,X,vc]
```

We replace the swapping constraints (i.e., swapping is not allowed along a normal edge or a slow 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.
:- 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.
```

by the following rules that describe the conditions violating these swapping constraints:

```
violate_swap(A1,A2,T,X,Y) :-
   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.
violate_slow_collision1(A1,A2,T,X,Y):-
   slow(A1,T,X,Y), slow(A2,T-1,Y,X), T>0, T<t-1, A1!=A2.
violate_slow_collision2(A1,A2,T,X,Y):-
   slow(A1,T,X,Y), slow(A2,T,Y,X), T<t-1, A1<A2.
```

and the following weak constraints:

```
:~ violate_swap(A1,A2,T,X,Y). [1@7, A1,A2,T,X,Y,vs]
:~ violate_slow_collision1(A1,A2,T,X,Y). [1@7, A1,A2,T,X,Y,vsc1]
:~ violate_slow_collision2(A1,A2,T,X,Y). [1@7, A1,A2,T,X,Y,vsc2]
```

Similarly, we replace the goal constraint (i.e., the agent should reach its destination), the waypoint constraint (i.e., the agent should visit the waypoints in its way to its destination), the obstacle collision constraint (i.e., no agent collides with an obstacle), and the battery constraint (i.e., the agent should have a positive battery level) by the following rules:

```
violate_goal(A,X) :- goal(A,X), not visit(A,X).
violate_waypoint(A,X) :- waypoint(A,X), not visit(A,X).
violate_obstacle(A,T,X) :- plan(A,T,X), obstacle(X), agent(A), time(T).
violate_min_battery(A,T,X):- batteryLevel(A,T,0), plan(A,T,X),
  planLength(A,L), T<L.
```

and the following weak constraints:

```
:~ violate_goal(A,X). [1@7, A,X,vg]
:~ violate_waypoint(A,X). [1@7, A,X,vw]
:~ violate_obstacle(A,T,X). [1@7, A,T,X,vo]
:~ violate_min_battery(A,T,X). [1@7, A,T,vb]
```

The idea is to identify from an answer set for Π_w, which constraints are violated, and then present an explanation to the user accordingly.

https://doi.org/10.1017/S1471068420000320 Published online by Cambridge University Press

3.3 Discussions

We briefly discuss some useful extensions and capabilities of our algorithm.

Relevancy of constraints to questions Instead of considering all constraints, we identify the constraints relevant to the given question to generate more meaningful explanations. For QW type queries, the constraints about collisions, obstacle, and waypoints are relevant. For QC type queries, the constraints about battery level, goal, obstacle, and waypoints are relevant. For QP and QU queries, all constraints (about battery level, collisions, goal, obstacle, waypoints) are relevant.

Weights and priorities In the ASP formulation presented above, the priority of the weak constraints is specified as a number larger than the priority of weak constraints used for mMAPF optimizations (i.e., minimizing the total number of times of charging, or minimizing the total plan length) because mMAPF constraints are more important than optimizations. Meanwhile, all mMAPF constraints are considered equally important, with the same priority and the same weight. In real-world applications, the users can change these priorities.

Π_h vs. Π_w Our algorithm first tries to generate an explanation for the observations of the user over the given plan, by means of queries QW1–QP5, using the program Π_h. If the given plan is not feasible or optimal, it generates further explanations by utilizing weighted weak constraints, i.e., using the program Π_w. The scrupulous reader might notice that we could have used Π_w from the very beginning, to answer queries QW1–QP5. However, it would not be computationally efficient. For instance, for a query of type QP1 over an instance with 4 agents on a grid of size 22×22, finding an answer using the program Π_h takes 31 seconds, while it takes 2249 seconds using the program Π_w.

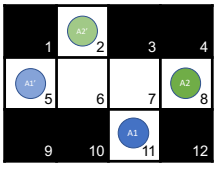
Additional explanations for QW and QC queries If the augmented program Π_h does not have an answer set, then the given plan is not feasible or optimal. At this point, for QW and QC queries about waiting and charging, the algorithm can generate further explanations by answering the following question: "What would happen if the agent did not wait/charge in the given plan?" For that, the algorithm revises the plan to exclude waiting/charging, obtains a program Π_c from program Π_w by adding the revised plan as a hard constraint, and checks the answer sets for Π_c. Scenario 2 in the next section provides a good example.

Possibility of infrastructure change For queries QU, our algorithm can generate further explanations based on the possibility of infrastructure change (e.g., removing some shelves) in the warehouse. For that purpose, the algorithm obtains two programs Π_{w1} and Π_{w2} from Π_h. If an infrastructure change is not possible, Π_{w1} does not include the weak constraint for obstacles; instead, it includes the hard constraints for obstacles and other relevant weak constraints. If an infrastructure change in the warehouse is possible, Π_{w2} includes the weak constraint for obstacles only. The algorithm computes answer sets for each program and generates more comprehensive explanations. Scenario 6 in the next section provides a good example.

4 Examples

Scenario 1 Consider the mMAPF instance shown in Figure 1(a) in a small warehouse, where Robot 1 is initially located at Cell 11 and aims to reach Cell 5, and Robot 2 is initially located at Cell 8 and aims to reach Cell 2. Cell 7 is a waypoint for both robots, and the upper bound on makespan of a plan is 4. For simplicity, suppose that the batteries of the robots are fully charged

https://doi.org/10.1017/S1471068420000320 Published online by Cambridge University Press



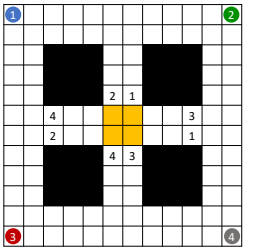
(a)

Time	A1 Location	A2 Location
0	11	8
1	7	8
2	6	7
3	5	6
4	-	2

(b)

Time	A1 Location	A2 Location
0	11	8
1	11	7
2	7	6
3	6	2
4	5	-

(c)



(d)

Fig. 1: (a) Scenarios 1 and 2: A1 and A2 denote the initial positions of Robots 1 and 2; A1′ and A2′ denote goal locations. Cell 7 is a waypoint for both robots, (b) and (c) are two optimal plans for this instance. (d) Scenario 3: Robots 1–4 are initially located at different corners of a 12*x*12 grid; each robot aims to reach the diagonally-opposite corner. Other cells labeled by a number *i* are waypoints for Robot *i*. Yellow cells are charging stations and black cells are obstacles.

initially, and sufficient for execution of any given plan, and that all edges have the normal mode of transportation.

About Plan 1 described in Figure 1(b), suppose that an engineer asks the following query of type QW1:

"Why does Robot 2 wait at Cell 8 (at any time)?"

Our algorithm first obtains Π_h^1 by adding the following constraint to the mMAPF program Π:

```
:- plan(2,T,8), plan(2,T+1,8), T<t.
```

checks whether there is an optimal plan where Robot 2 does not have to wait initially (i.e., Π_h has an answer set).

Once an alternative solution, Plan 2 described in Figure 1(c), is found, our algorithm presents the following explanation:

"Actually, Robot 2 does not have to wait at Cell 8 from time step 0 to 2. Here is an alternative optimal plan: Plan 2..."

Scenario 2 Continuing Scenario 1, suppose that the engineer then asks the following query of type QW1:

"Why does Robot 1 wait at Cell 11 (at any time)?"

Our algorithm first obtains Π_h^2 by adding the following constraint to Π_h^1:

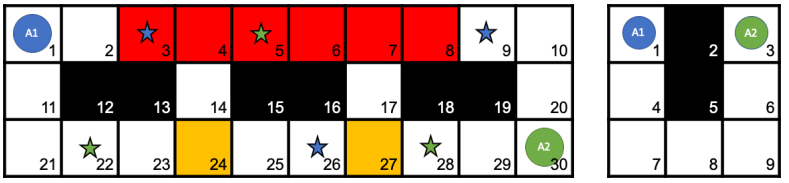
```
:- plan(1,T,11), plan(1,T+1,11), T<t.
```

and checks whether there is an optimal plan where Robot 1 does not have to wait initially (i.e., Π_h^2 has an answer set). However, Π_h^2 does not have an answer set: there are no other solutions.

Then, our algorithm tries to generate explanations by answering two questions: "What would happen if Robot 1 does not wait at Cell 11 in the current plan?" "What will happen if Robot 1 does not wait at Cell 11?" To answer the first question, our algorithm obtains a program Π_w^2 from Π_h^2, by replacing the collision constraints with the relevant rules and weak constraints as described above. It revises the current plan so that Robot 1 does not wait at Cell 11 (and the plans of other agents do not change), and obtains a program Π_c^2 by adding the revised plan as a hard constraint to Π_w^2. After that, our algorithm computes an answer set for Π_c^2, identifies the atoms `violate_collision(1,2,1,7)` and `violate_collision(1,2,2,6)` in it, generates the following explanation for the first question:

"Robot 1 has to wait at Cell 11 in the current plan; otherwise, Robot 1 and Robot 2 would collide with each other at Cell 7 at time step 1 and at Cell 6 at time step 2."

https://doi.org/10.1017/S1471068420000320 Published online by Cambridge University Press



(a) (b)

Fig. 2: (a) Instance M1. Robots 1 and 2 are initially located at two corners of the grid and each robot aims to reach the diagonally-opposite corner. The stars show the waypoints of the robot with the same color. Yellow cells are charging stations, red cells are the slow zone and black cells are obstacles. (b) Robots 1 and 2 are initially located at the given positions on the grid; the goal of each robot is the initial location of the other. Black cells are obstacles.

To answer the second question, our algorithm finds an answer set for Π_w^2. This answer set contains a different plan with a makespan of 4 and the atom `violate_collision(1,2,1,7)`, and thus our algorithm generates the following explanation for the second question:

"Robot 1 has to wait at Cell 11; otherwise, Robot 1 and Robot 2 will collide with each other at Cell 7 with another plan."

Scenario 3 Consider the mMAPF instance shown in Figure 1(d). Robots 1–4 are initially located at different corners of the warehouse. Each robot aims to reach the diagonally-opposite corner. Other cells labeled by a number i are waypoints for Robot i. Yellow cells are charging stations and black cells are obstacles.

Suppose that a plan is given to an engineer as an optimal solution, where each Robot executes a plan of length 22, and she asks a query of type QP1:

"Why does not Robot 1 follow a shorter plan whose length is smaller than 22?"

Our algorithm first obtains the program Π_h from the mMAPF program Π by adding the constraint

```
:- planLength(1,L), L>=22.
```

and checks whether there is another optimal solution where Robot 1 follows a shorter plan.

Since the program Π_h does not have an answer set, our algorithm tries to identify which mMAPF constraint is violated. For that, it obtains the program Π_w from Π_h by replacing all mMAPF constraints with relevant rules and weak constraints, as described in the previous section.

According to an answer set computed for Π_w, which contains `violate_waypoint(1,55)`, `violate_waypoint(1,82)` and `violate_goal(1,144)`, the algorithm generates the following explanation:

"Robot 1 cannot follow a shorter plan; otherwise, it would not be possible to visit all the waypoints and to reach its goal."

Scenario 4 Let us consider the mMAPF instance shown in Figure 2(a), Robots 1–2 are initially located at 1 and 30 an their goal locations are 30 and 1, respectively. Yellow cells are charging stations, red cells are the slow zone and black cells are obstacles. The stars show the waypoints of the agent of the same color. Maximum battery level is 10 and Robot 1 is fully charged initially but initial battery level of Robot 2 is 8.

Suppose that a plan is given to an engineer as an optimal solution, where each Robot is charged 2 times in the plan. The engineer wants to charge less, so she asks the following query of type QC4:

https://doi.org/10.1017/S1471068420000320 Published online by Cambridge University Press

" Why does not Robot 2 charge less than 2 times?"

To answer this query, our algorithm adds the following constraint to the mMAPF program Π and obtains Π_h:

```
:- C = #count{T: batteryLevel(2,T,b), T>0} , C >= 2.
```

Then, it tries to find a plan with less number of charging (i.e., an answer set for Π_h). There is no answer set of Π_h. The algorithm obtains the program Π_w from Π_h by replacing the relevant hard constraints with the weak constraints, and then obtains the program Π_c from Π_w by adding the current traversals of the agents as hard constraints. The answer set for Π_c contains the atom `violate_min_battery(2,8,intransit)`.

Then, our algorithm further tries to find an answer set for Π_w to find out which constraints would be violated regardless of the current plan. The answer set contains a different plan and the atom `violate_waypoint(2,5)`.

In the end, the algorithm generates the following explanation:

"Robot 2 cannot charge less than 2 times; otherwise, its battery will run out at time step 8 if it uses the current plan or it will not be able to visit its waypoint at Cell 5 with another plan."

Scenario 5 Again, let's consider the scenario in Figure 2(a). Robot 1 has the plan

$$P_1 = \langle 1,2,3,intransit,4,14,24,25,26,27,17,7,intransit,8,9,10,20,30\rangle .$$

The engineer wants to know if there is a plan without visiting the edge $\langle 4,14\rangle$ and asks the following query of type QP4:

"Why does Robot 1 move from Cell 4 to Cell 14 (at any time)?"

The algorithm adds the following constraint to Π and obtains Π_h:

```
:- plan(1,T,4), plan(1,T+1,14), edge(4,14), mode(4,14,'n'), T<t.
```

There exists an answer set for Π_h, with a longer plan for Robot 1. Therefore, the algorithm generates the following explanation:

"Actually, Robot 1 does not have to move from Cell 4 to Cell 14. Here is an alternative plan which is longer: ...

Scenario 6 Now consider the instance in Figure 2(b). Robots 1–2 are initially located at 1 and 3; their goals are at 3 and 1, respectively. Black cells denote obstacles. For simplicity, we assume that their batteries are initially fully charged and enough to traverse all of their paths, all edges have normal mode of transportation, and the only waypoint of each robot is located at its initial position.

There is no solution for this mMAPF instance. Suppose that an engineer wants to find out the reason and asks query QU:

"Why does not the instance have a solution?"

The algorithm obtains two programs Π_{w1} and Π_{w2} from Π_h, as explained in the previous section, and tries to find an answer set for each program. The answer set for Π_{w1} contains `violate_collision(1,2,3,8)`, and the algorithm generates the following explanation:

"There is no solution because Robot 1 and Robot 2 collide at Cell 8 at time step 3."

The answer set for Π_{w2} contains `violate_obstacle(2,1,2)`, and the algorithm further generates the following explanation:

"There is no solution because Robot 2 collides with the obstacle at Cell 2 at time step 1; this suggests removing this obstacle."

https://doi.org/10.1017/S1471068420000320 Published online by Cambridge University Press

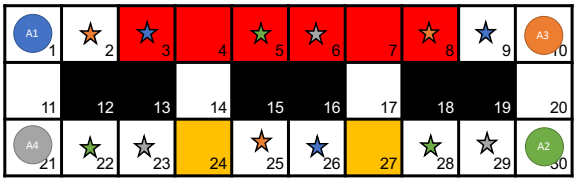


Fig. 3: Instance M2 with 4 agents initially located at corners and their goals are located at the diagonally-opposite corner.

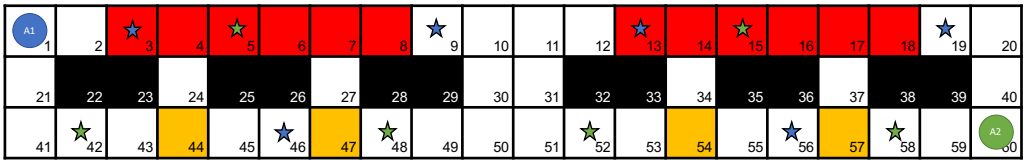


Fig. 4: Instance M3 with 2 agents an a larger grid. Agents are located at two corners of the grid and their goals are at the diagonally-opposite corner.

5 Experimental Evaluations

We have evaluated our ASP-based method for generating explanations, considering all 13 types of queries (except query QU) over two mMAPF instances with multi-modality, resource, waypoint, and plan length constraints within a tight space to move around. The first instance M1 is shown in Figure 2(a), with two agents and where the upper bound on makespan is 18. The second mMAPF instance M2 considers the same environment as in the first instance, with two more agents initially placed in the empty corners (Cells 10 and 21) with the goal of swapping their locations. The three waypoints for each new agent are placed in a similar way as for the existing agents. The upper bound on makespan is 23.

For each query type, we have generated all possible query instances over all agents, waypoints, time steps, etc. For instance, query QC1 asks why an Agent `a` charge at location `x`. According to the solution for the first mMAPF instance, Agent A1 charges once and Agent A2 charges twice in their plans so we consider all 3 query QC1 instances. Query QP1 asks why Agent `a` does not have a plan whose length is less than `l`, so we consider 2 query QP1 instances for each agent with length `l`.

For each query instance, we have run our algorithm to generate an explanation. For each query, we report the average CPU time in seconds, the total number of calls to CLINGO and the number of answer sets computed by CLINGO within these calls. The results of these experiments are shown in Tables 1(left) and 1(right). Note that the queries QW1–QW4 are not applicable for the first mMAPF instance since no agent waits. For example, in Table 1(left), for QP2 over the mMAPF instance M1, explanations are generated for 30 query instances. In total, 30 QP2 query instances call CLINGO 57 times (30 calls with hard constraints, and 27 calls with relevant weak constraints) where 867 answer sets are computed during optimizations; the average computation time for explanation generation per query instance is 0.394 seconds. For the mMAPF instance M2 (Table 1(right)), 59 QP2 instances call CLINGO 95 times (59 calls with hard constraints, and 36 calls with relevant weak constraints) where 5843 answer sets are computed during optimizations; the average CPU time for explanation generation per instance is 102.225 seconds.

We can observe from these results that doubling the number of agents increases the computation times. The explanations for most of the query instances over the mMAPF instance M1 include recommendations for alternative plans. For the mMAPF instance M2, only some of query

https://doi.org/10.1017/S1471068420000320 Published online by Cambridge University Press

Table 1: *Experimental results for (left-table) the mMAPF instance M1 shown in Figure 2 with 2 agents where the upper bound on makespan is 18, and (right-table) the revised mMAPF instance, M2, with 2 more agents located at the empty corners with the goals of swapping their locations and where the upper bound on makespan is 23. The total number of* CLINGO *calls and the answer sets computed in these calls, and the average CPU time (per query instance) in seconds are reported.*

Query	#Instances	#Calls [#Models]	Time (sec)
QC1	3	5 [48]	0.402
QC2	3	5 [63]	0.319
QC3	3	5 [63]	0.333
QC4	2	4 [91]	0.478
QP1	2	4 [42]	0.321
QP2	30	57 [867]	0.394
QP3	30	45 [545]	0.280
QP4	30	53 [846]	0.410
QP5	30	42 [452]	0.241

Query	#Instances	#Calls [#Models]	Time (sec)
QW1	4	4 [67]	3.695
QW2	4	4 [52]	3.457
QW3	4	4 [52]	3.605
QW4	4	4 [52]	3.571
QC1	6	8 [321]	45.762
QC2	7	7 [87]	3.381
QC3	7	7 [74]	3.387
QC4	4	7 [541]	100.269
QP1	4	6 [306]	13.806
QP2	59	95 [5843]	27.295
QP3	70	70 [898]	3.425
QP4	66	87 [4463]	18.743
QP5	66	66 [852]	4.127

Table 2: *Experimental results for the mMAPF instance M3 with 2 agents, shown in Figure 2, (left-table) with optimization and (right-table) with anytime search with a time limit of 100 seconds and upper bound for makespan is 45. The total number of* CLINGO *calls and the answer sets computed in these calls, and the average CPU time (per query instance) in seconds are reported.*

Query	#Instances	#Calls [#Models]	Time (sec)
QW1	2	2 [13]	57.652
QW2	2	2 [17]	57.319
QW3	2	2 [17]	57.462
QW4	2	2 [17]	57.674
QC1	6	11 [363]	2437.347
QC2	8	12 [266]	428.983
QC3	8	12 [328]	370.401
QC4	2	4 [159]	6576.363
QP1	2	4 [166]	579.143
QP2	63	116 [4277]	1083.826
QP3	74	113 [2893]	510.654
QP4	71	119 [4053]	985.894
QP5	72	109 [2824]	516.365

Query	#Instances	#Calls [#Models]	Time (sec)
QW1	2	2 [13]	53.282
QW2	2	2 [17]	48.105
QW3	2	2 [17]	47.203
QW4	2	2 [17]	47.265
QC1	6	11 [254]	125.401
QC2	8	12 [209]	94.569
QC3	8	12 [288]	93.489
QC4	2	4 [89]	181.45
QP1	2	4 [142]	129.776
QP2	63	116 [3109]	102.225
QP3	74	113 [2222]	98.412
QP4	71	119 [2951]	99.575
QP5	72	109 [1966]	95.451

QC1, QC4, QP1, QP2, QP4 instances include recommendations for alternative plans; that is why the CPU times are larger for these queries.

We have also experimented with a mMAPF instance, M3, obtained from the first instance M1 by replicating the environment twice, towards the right side, as illustrated in Figure 4. The two agents are initially placed at the far opposite corners in the same way as in the first instance, with the goal of swapping their locations. The results are shown in Table 2(left). We have observed from these results that doubling the size of the environment (and thus the plan) increases the computation times. For the mMAPF instance M3, 63 QP2 instances call CLINGO 116 times (63 calls with hard constraints, and 53 calls with relevant weak constraints) where 4277 answer sets are computed during optimizations; the average CPU time for explanation generation per instance

https://doi.org/10.1017/S1471068420000320 Published online by Cambridge University Press

is around 18 minutes. This is not surprising since the increase in grid size has a significant effect in the computation time for MAPF problems, as also observed in our earlier studies (Erdem et al. 2013).

For the mMAPF instance M3, we have also experimented with CLINGO by utilizing its anytime search feature with a time threshold of 100 seconds: CLINGO reports the best solution it computes within 100 seconds. The results are shown in Table 2(right). Then, the average computation times reduce significantly. For instance, for QP2 instances, the average CPU time for generating an explanation reduces to 102.225 seconds.

6 Discussion and Conclusion

We present a novel method for generating a variety of explanations for mMAPF problems, motivated by real life applications in autonomous warehouses, using answer set programming. These explanations are requested by different types of queries posed by the users interactively. In that sense, it is useful for the users to better understand the strengths and weaknesses of the plans being executed by robots at their warehouses, and the limitations of the warehouse infrastructure.

This contribution of query-based explanation generation is important also from the perspective of studies on MAPF. MAPF has been investigated in AI using search algorithms (Silver 2005; Luna and Bekris 2011; Dresner and Stone 2008; Wang and Botea 2008; Jansen and Sturtevant 2008; Chouhan and Niyogi 2015; Sharon et al. 2015; Stern et al. 2019) or declarative methods (Yu and LaValle 2013; Surynek 2012; Erdem et al. 2013). However, explainability for MAPF problems, as described above, has not been investigated in the literature. The only relevant work that studies explainability for MAPF problems is very recently published: explainability is understood as verification of whether a given plan involves collisions (Almagor and Lahijanian 2020), and the authors introduce a decomposition-based search method for such explanation schemes. In that sense, our study is useful for MAPF studies by providing a novel query-based declarative method for generating a variety of knowledge-rich explanations for a general variant of MAPF.

Explainability of plans has been emphasized by Smith (2012) for planning as an iterative process. As planning domains approximate the real-world and optimization functions may not reflect the desired conditions, the computed plans may not be as good as expected. Smith suggests, in such cases, that planning should be an iterative process where the users inspect the plans and provide feedback to the planner for further improvement. In this process, Smith points out that providing explanations to following questions plays an important role: Why is a given action included in the plan? Why is this action done before that one? Why does the plan not satisfy this property? Why does the plan not achieve this goal? Note that all these questions can be handled by QP, QW, QC and QU queries in the context of mMAPF. Recently, Eifler et al. (2020) has presented a method to generate "contrastive" explanations to questions of the last two forms: "Why not p?" where p is a propositional formula describing an plan property. The underlying idea is to generate an explanation based on the properties q (selected from a given set of properties) entailed by p: "because that would necessitate $\neg q$." The authors, in particular, focus on properties p (called "action-set" properties) that are about actions involved in the plan, like "vertex x is visited by agent y" or "edge from x to y is used by agent y." For instance, one of the questions they study in the IPC NoMystery domain—a transportation domain with trucks delivering packages to destinations—is "Why does truck $T0$ not avoid the road from location $L0$ to location $L5$?" Note that this question is very similar to the question QP4 stated in Scenario 5. In that sense,

https://doi.org/10.1017/S1471068420000320 Published online by Cambridge University Press

our approach of utilizing hard constraints and weak constraints is general enough to generate explanations to questions investigated for plan explainability.

Our method is based on an algorithm that utilizes weighted weak constraints of answer set programming for generating explanations by means of counterfactuals, and that allows a sequence of interactive query answering by means of hypothetical reasoning. This contribution of counterfactual-based explanation generation using weighted weak constraints is important also from the perspective of studies on explanation generation in ASP. Explanation generation has been investigated in answer set programming, based on justifications, debugging and/or argumentations (Pontelli et al. 2009; Schulz and Toni 2013; Schulz and Toni 2016; Cabalar et al. 2014; Cabalar and Fandinno 2016; Damásio et al. 2013; Brain et al. 2007; Gebser et al. 2008; Oetsch et al. 2010; Erdem and Öztok 2015), as summarized in the surveys (Fandinno and Schulz 2019; Dodaro et al. 2019). For instance, in our earlier studies (Erdem and Öztok 2015), we generate explanations for complex biomedical queries for drug discovery (expressed in a controlled natural language), based on the idea of finding justifications. Our method extends this list by the use of weighted weak constraints.

Acknowledgments We have benefited from useful discussions with Alexander Kleiner (Robert Bosch GmbH, Corporate Research) and Volkan Patoglu (Sabanci University, Mechatronics Engineering) about real world robotic applications of mMAPF in warehouses.

References

ALMAGOR, S. AND LAHIJANIAN, M. 2020. Explainable multi agent path finding. In *Proc. of AAMAS*. 34–42.

BOGATARKAN, A., ERDEM, E., KLEINER, A., AND PATOGLU, V. 2020. Multi-modal multi-agent path finding with optimal resource utilization. In *Proceedings of 5th International Conference on the Industry 4.0 Model for Advanced Manufacturing*. 313–324.

BOGATARKAN, A., PATOGLU, V., AND ERDEM, E. 2019. A declarative method for dynamic multi-agent path finding. In *Proceedings of the 5th Global Conference on Artificial Intelligence*. 54–67.

BRAIN, M., GEBSER, M., PÜHRER, J., SCHAUB, T., TOMPITS, H., AND WOLTRAN, S. 2007. "that is illogical captain!"–the debugging support tool spock for answer-set programs: System description.

CABALAR, P. AND FANDINNO, J. 2016. Justifications for programs with disjunctive and causal-choice rules. *Theory Pract. Log. Program. 16,* 5-6, 587–603.

CABALAR, P., FANDINNO, J., AND FINK, M. 2014. Causal graph justifications of logic programs. *Theory Pract. Log. Program. 14,* 4-5, 603–618.

CALIMERI, F., FABER, W., GEBSER, M., IANNI, G., KAMINSKI, R., KRENNWALLNER, T., LEONE, N., RICCA, F., AND SCHAUB, T. 2020. ASP-Core-2 Input Language Format. *Theory Pract. Log. Program. 20,* 2, 294–309.

CHOUHAN, S. S. AND NIYOGI, R. 2015. DMAPP: A distributed multi-agent path planning algorithm. In *Proc. of AI*. 123–135.

DAMÁSIO, C. V., ANALYTI, A., AND ANTONIOU, G. 2013. Justifications for logic programming. In *Proc. of LPNMR*. 530–542.

DIJKSTRA, E. W. 1959. A note on two problems in connexion with graphs. *Numer. Math. 1,* 1, 269–271.

DODARO, C., GASTEIGER, P., REALE, K., RICCA, F., AND SCHEKOTIHIN, K. 2019. Debugging non-ground ASP programs: Technique and graphical tools. *Theory Pract. Log. Program. 19,* 2, 290–316.

DRESNER, K. M. AND STONE, P. 2008. A multiagent approach to autonomous intersection management. *J. Artif. Intell. Res. (JAIR) 31*, 591–695.

EIFLER, R., CASHMORE, M., HOFFMANN, J., MAGAZZENI, D., AND STEINMETZ, M. 2020. A new

https://doi.org/10.1017/S1471068420000320 Published online by Cambridge University Press

approach to plan-space explanation: Analyzing plan-property dependencies in oversubscription planning. In *Proc. of AAAI*. 9818–9826.

ERDEM, E., KISA, D. G., OZTOK, U., AND SCHUELLER, P. 2013. A general formal framework for pathfinding problems with multiple agents. In *Proc. of AAAI*.

ERDEM, E. AND ÖZTOK, U. 2015. Generating explanations for biomedical queries. *Theory Pract. Log. Program. 15,* 1, 35–78.

FANDINNO, J. AND SCHULZ, C. 2019. Answering the "why" in answer set programming - A survey of explanation approaches. *Theory Pract. Log. Program. 19,* 2, 114–203.

GEBSER, M., PÜHRER, J., SCHAUB, T., AND TOMPITS, H. 2008. A meta-programming technique for debugging answer-set programs. In *Proc. of AAAI*. 448–453.

GELFOND, M. AND LIFSCHITZ, V. 1988. The stable model semantics for logic programming. In *Proceedings of International Logic Programming Conference and Symposium*. 1070–1080.

GELFOND, M. AND LIFSCHITZ, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9*, 365–385.

JANSEN, R. AND STURTEVANT, N. 2008. A new approach to cooperative pathfinding. In *Proc. of AAMAS*. 1401–1404.

LIFSCHITZ, V. 2002. Answer set programming and plan generation. *Artificial Intelligence 138*, 39–54.

LUNA, R. AND BEKRIS, K. E. 2011. Efficient and complete centralized multi-robot path planning. In *Proc. of IROS*. 3268–3275.

MAREK, V. AND TRUSZCZYŃSKI, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*. Springer Verlag, 375–398.

NIEMELÄ, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence 25*, 241–273.

OETSCH, J., PÜHRER, J., AND TOMPITS, H. 2010. Catching the ouroboros: On debugging non-ground answer-set programs. *Theory Pract. Log. Program. 10,* 4-6, 513–529.

PONTELLI, E., SON, T. C., AND EL-KHATIB, O. 2009. Justifications for logic programs under answer set semantics. *Theory Pract. Log. Program. 9,* 1, 1–56.

RATNER, D. AND WARMUTH, M. K. 1986. Finding a shortest solution for the n $\times$ n extension of the 15-puzzle is intractable. In *Proc. of AAAI*. 168–172.

SCHULZ, C. AND TONI, F. 2013. Aba-based answer set justification. *Theory Pract. Log. Program. 13,* 4-5-Online-Supplement.

SCHULZ, C. AND TONI, F. 2016. Justifying answer sets using argumentation. *Theory Pract. Log. Program. 16,* 1, 59–110.

SHARON, G., STERN, R., FELNER, A., AND STURTEVANT, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artif. Intell. 219*, 40–66.

SILVER, D. 2005. Cooperative pathfinding. In *Proc. of AIIDE*. 117–122.

SMITH, D. E. 2012. Planning as an iterative process. In *Proc. of AAAI*, J. Hoffmann and B. Selman, Eds.

STERN, R., STURTEVANT, N. R., FELNER, A., KOENIG, S., MA, H., WALKER, T. T., LI, J., ATZMON, D., COHEN, L., KUMAR, T. K. S., BARTÁK, R., AND BOYARSKI, E. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proc. of SOCS*. 151–159.

SURYNEK, P. 2012. On propositional encodings of cooperative path-finding. In *Proc. of ICTAI*. 524–531.

WANG, K.-H. C. AND BOTEA, A. 2008. Fast and memory-efficient multi-agent pathfinding. In *Proc. of ICAPS*. 380–387.

YU, J. AND LAVALLE, S. M. 2013. Planning optimal paths for multiple robots on graphs. In *Proc. of ICRA*. 3612–3617.

https://doi.org/10.1017/S1471068420000320 Published online by Cambridge University Press