Flexible, Lifelong, Explainable, and Robust Solutions for Multi-Agent Path Finding Problems

Aysu Bogatarkan

Sabancı University, Faculty of Engineering and Natural Sciences, Istanbul, Turkiye

Abstract

The multi-agent path finding (MAPF) problem is a combinatorial search problem that aims at finding paths for multiple agents in an environment (e.g., robots in an autonomous warehouse) such that no two agents collide with each other, and subject to some constraints on the lengths of paths. The real-world applications of MAPF require flexible, lifelong, robust and explainable solutions. In this study, these challenges are being addressed.

Keywords

answer set programming, multi-agent path finding, autonomous warehouses, explanation generation

1. Introduction

For the success of Artificial Intelligence (AI) applications, two of the important features (and challenges) needed to be addressed by them 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 for a better understanding. An explainable AI method can provide answers to queries about the (in)feasibility and the optimality of solutions. In addition to being flexible and explainable, it is desired for AI methods to provide robust and lifelong solutions for the problems. A robust solution for an AI application can still be used even after unexpected errors occur, and a lifelong AI method can adapt to changes during operation. One of the well-studied problems in AI that necessitates solutions for these 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, subject to some constraints on the maximum or the total plan length. Optimal solutions for MAPF are usually found by optimizing the makespan or the total plan length. According to the needs of the application, other optimization functions can be used. MAPF with constraints on plan lengths is intractable [1]. MAPF has been studied in various domains, such as robotics [2], autonomous warehouse systems [3], traffic control [4], and video games [5].

This study focuses on introducing flexible, lifelong and robust methods for MAPF and its variants, and explainable frameworks for some MAPF variants. In all of our solutions, we utilize Answer Set Programming (ASP) [6, 7, 8]—a logic programming paradigm based on answer sets [9, 10] and implement our methods using the ASP solver Clingo [11].

For some real-world applications, being able to solve MAPF problem may not be enough to address all challenges or the realistic conditions of the application. For instance, in real-world automated warehouses, the robots' battery levels change as they travel and they may need to be charged to complete their tasks. Furthermore, some parts of the warehouses with human occupants or tight passages may require robots to move slowly to ensure safety. One aim of our study is to address these issues with flexible frameworks in the spirit of elaboration tolerance [12]. We have defined a general variation of MAPF (called mMAPF) and introduced a method addressing these challenges [13].

We have also investigated the challenge of explainability for this problem, in particular, considering queries about the (in)feasibility and the optimality of solutions, as well as queries about the observations about these solutions. Given a solution for mMAPF, our explainable framework is able to explain

ICLP DC 2025: 21st Doctoral Consortium on Logic Programming, September 2025, Rende, Italy.

aysubogatarkan@sabanciuniv.edu (A. Bogatarkan)
0000-0003-3363-1987 (A. Bogatarkan)

infeasibility or nonoptimality of this solution, confirm its feasibility and suggest alternatives for the solution, and provide explanations for some queries, utilizing counterfactual reasoning and identifying violations of constraints [14].

In a warehouse that is not completely autonomous, some changes may occur during the execution of a plan: existing agents may leave the environment, or new agents may be included in the team with new tasks, existing obstacles may be removed from the environment or moved to some other location in the environment. To be able to handle these changes, we have defined a general Dynamic Multi-Agent Path Finding (D-MAPF) problem and introduced multiple lifelong methods to solve this problem [15, 16].

In addition to our currently existing methods, we aim to address robustness for variations of MAPF. Furthermore, we aim to go beyond MAPF, considering possible real-world applications (e.g., in robotics, games).

2. Related Work

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

For instance, Silver [17] introduces an incremental method where the paths of agents are computed one by one with A* [18]. Search algorithms such as [4, 19], compute paths independently, and in case of a collision, 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.

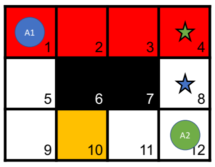
The declarative methods reduce MAPF to some formalisms such as ILP [21], SAT [22] and ASP [23] and use general problem solvers to find plans and optimize makespan or distance. None of these earlier works is applicable to multi-modal problems and consider resource utilization of the agents or changes in the environment.

In the only relevant work that studies explainability for MAPF problems, explainability is understood as verification of whether a given plan involves collisions [24], and the authors introduce a decomposition-based search method for such explanation schemes.

Target Assignment and Path Finding problems (TAPF and G-TAPF) [25, 26] are variants of MAPF. They partition the agents into teams, assign some goals to each team, and solve MAPF utilizing ASP, however, changes in the environment are not considered. Like TAPF and G-TAPF, Multi-Agent Pickup and Delivery (MAPD) [27] also involves assignment of tasks to agents but over time, requiring an online solution. While MAPD has a dynamic aspect of emerging new tasks, it does not allow the team or environment to change. Online MAPF [28] considers addition of new agents to the team while a plan is being executed, but no other changes in the team or environment is considered. Moreover, it is assumed that agents disappear when they reach their goal and that new agents may wait before entering their initial location in the environment. Lifelong MAPF [29, 30] considers assignment of new goal locations to the agents who have completed their plans, which can be viewed as adding a new agent. Atiq et al. [31] consider the D-MAPF problem and when a change occurs in the environment, a minimal subset of agents having conflicts are identified and replanning is applied to resolve conflicts.

3. Flexible Solutions for MAPF

In real-world automated warehouses, the robots' battery levels change as they move around, and in some parts of these warehouses, due to presence of humans or tight passages, the robots may need to move slowly to ensure safety. Hence, to be able to address more realistic scenarios, a mathematical model general enough to handle multi-modal transportation conditions and multi-objective optimizations 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 modelled a general version of MAPF (called mMAPF– multi-modal MAPF with resources) as a rich graph problem and introduced a flexible method to solve mMAPF declaratively, using ASP. Our method can handle the following variations of MAPF: *multi-objective optimization*, *waypoints*, *resource constraints* and *multi-modal transportation*.



(a)

Time	0	1	2	3	4	5	6	7	8
Robot 1 Location	1	5	9	10	11	12	8	12	-
Robot 2 Location	12	8	4	transit	3	transit	2	transit	1

(b)

Figure 1: (a) A1 and A2 denote the initial positions of Robots 1 and 2; each robot aims to reach the diagonally-opposite corner. Cell 8 is a waypoint for Robot 1 and Cell 4 is a waypoint for Robot 2. Yellow cell is the charging station, red cells are the slow zone, and black cells are obstacles. (b) is an optimal plan for this instance.

As an example, consider the instance described in Figure 1(a) in a small warehouse, viewed as a 3x4 grid. The warehouse contains a shelf unit that occupies two grid cells, denoted as obstacles shown by the black cells. As the robots move, their battery level decreases by 1 at each time step and they may need charging. For this purpose, the warehouse contains a charging station, denoted by the yellow cell located at Cell 10. If a robot is at a charging station, its battery level may quickly get to the maximum level or the robot can move forward without charging its battery. In the corridor denoted by red cells, the robots should move slowly, due to humans working nearby. Normally, it takes one time step for a robot to move from one grid cell to the other, but in this slow corridor it takes two time steps to move from one cell to the next. When the robot is located between two grid cells, we say that it is "*in transit*". There are two robots, $A1$ and $A2$, in this warehouse and they are initially located in two corners, denoted by the colored circles. The robots aim to swap their places at the end. The stars denote the waypoint of the same colored robot and the robot must visit its waypoint to pick up items on its way to its goal location. An optimal solution for this instance is described in Figure 1(b).

Details of the mathematical model and the ASP formulation can be found in our paper, *Multi-Modal Multi-Agent Path Finding with Optimal Resource Utilization* [13].

4. Explainable Solutions for MAPF

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

Explaining infeasibility. If the modified solution is found infeasible, e.g., using the ASP methods introduced by Bogatarkan et al. [13], then an explanation regarding its infeasibility could be "due to collisions with obstacles or other robots", or "due to low battery-level".

Explaining nonoptimality. If the modified solution is not optimal, then an explanation regarding its nonoptimality 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 (e.g., where the tasks are completed earlier) is computed. Then, in addition to confirming the feasibility of the plan, it would be useful to provide this alternative solution 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 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 this long path, but it needs to follow a different 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 have introduced a method for generating explanations for mMAPF. Our method considers different types of queries about mMAPF, and generates knowledge-rich explanations (including suggestions) for each type of queries. For queries with affirmative answers, it generates alternative solutions as suggestions. For queries with negative answers, it utilizes counter-factual reasoning and weighted weak constraints to generate causality-based explanations and further recommendations. Since our method is query-based, utilizing the elaboration tolerance of ASP, it allows a sequence of interactive query answering by means of hypothetical reasoning.

Details of the algorithm and the ASP formulations, together with more examples and experimental evaluations are presented in our paper, *Explanation Generation for Multi-Modal Multi-Agent Path Finding with Optimal Resource Utilization using Answer Set Programming* [14].

5. Lifelong Solutions for MAPF

We introduced lifelong solutions for MAPF in dynamic environments. When changes occur in a dynamic environment, such as obstacles being removed or moved, existing agents leaving the environment or new agents being added to the team, the aim is to find a new solution for the new team of agents in the modified environment. We call this problem Dynamic Multi-Agent Path Finding Problem (D-MAPF).

5.1. Revise-and-Augment for D-MAPF

One of the possible solutions for D-MAPF is replanning: consider a new MAPF instance defined by the current locations and goal locations of both the existing and the new agents, and the updated environment, and compute a solution for this instance. Although replanning finds a solution, if one exists, it does not re-use the plans of the existing agents and may not be computationally efficient.

With this motivation, we proposed a novel method to solve D-MAPF, using Answer Set Programming (ASP). The main idea (and novelty) of this method is, instead of replanning for all the agents right away, to *revise and augment* the existing MAPF solution: (*revise*) try to schedule the waiting times of existing agents as they traverse the rest of their paths, (*augment*) while computing paths for the new agents within a given makespan (i.e., the length of the plan). In this way, the paths for the existing agents can be re-used as part of the new plan. We have implemented this framework using Python and the ASP solver Clingo. In our experimental evaluations, we observed that the re-use of plans as proposed by our method improves the computational efficiency in timings significantly compared to replanning.

Figures 2–4 present an example (from our paper) to illustrate the overall idea of *Revise-and-Augment* method.

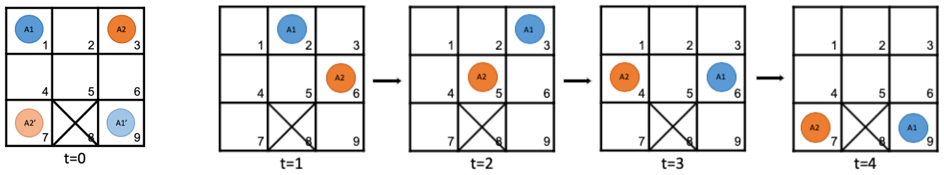


Figure 2: Initially ($t = 0$), two agents are at $A1$ and $A2$; their goals are at $A1'$ and $A2'$. A solution to this instance is shown: Plan of $A1$ is 1, 2, 3, 6, 9; and plan of $A2$ is 3, 6, 5, 4, 7.

The problem definition, ASP formulation, algorithm and experimental evaluations are described in detail in our paper *A Declarative Method for Dynamic Multi-Agent Path Finding* [15].

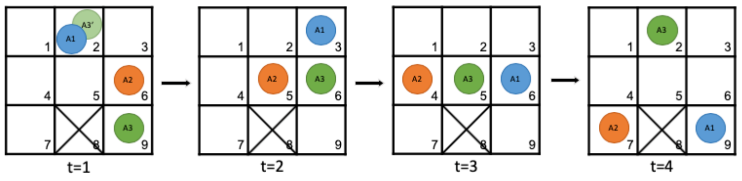


Figure 3: While executing the plan shown in Fig. 2, at time step $t = 1$, another agent $A3$ joins the team with goal $A3'$. A solution to this D-MAPF instance is computed, and a path for $A3$ is found, as shown above: $A1$ is at 2, moves to 3, 6, 9; $A2$ is at 6, moves to 5, 4, 7; $A3$ starts at 9, moves to 6, 5, 2.

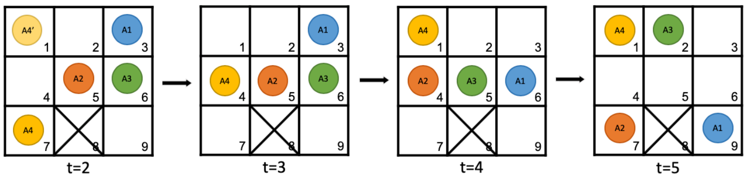


Figure 4: While executing the plan shown in Fig. 3, at time step $t = 2$, another agent $A4$ joins the team with goal $A4'$. A solution to this D-MAPF instance is computed, but no solution found with makespan $t = 4$, so the makespan is increased by 1. The new solution is found by scheduling the waiting times of $A1$, $A2$ and $A3$, and by computing a plan for $A4$, as shown above: $A1$ is at 3, waits at 3, then moves to 6,9; $A2$ is at 5, waits at 5, then moves to 4,7; $A3$ is at 6, waits at 6, then moves to 5,2; $A4$ starts at 7, moves to 4, 1, and waits at 1.

5.2. Revise-and-Augment-in-Tunnels for D-MAPF

In a more recent study, we investigated D-MAPF problem further. We introduced a rigorous definition for D-MAPF, that is general enough to cover 1) various changes in the environment and the team of agents over time, 2) different objective functions on plans, and 3) different assumptions on appearances/disappearances of agents, and that is not specifically oriented towards a particular method. Furthermore, we introduced a new framework to solve D-MAPF, that is general and flexible enough to allow different replanning and/or repairing methods. With the motivation of a modular architecture and efficient computations, our framework utilizes multi-shot computation [32] of ASP, unlike our previous work [15], where single-shot ASP was used. Multi-shot solving allows changes to the input ASP program in time, by introducing an external control to the ASP system. The external control allows operations, such as adding and grounding new programs, assigning truth values of some atoms, and solving the updated program, while the ASP system is running.

We designed and implemented the Replan-All (that replans for every agent after each change) and Revise-and-Augment methods using multi-shot ASP, and integrated them in the general D-MAPF framework. We empirically observed that multi-shot Replan-All is computationally more efficient but sometimes dramatic changes in the paths of the existing agents occur in the recomputed plans. Such changes are not desired from the perspective of real-world applications. For instance, in a warehouse where robots collaborate with human workers, changes in the routes of robots might be unexpected, distracting, unsafe, and inefficient for human workers.

With this motivation, we introduced a new method for D-MAPF, called *Revise-and-Augment-in-Tunnels*, that combines the advantages of these two methods. Unlike Revise-and-Augment, this method does not require that every existing agent follow their existing paths while revising their plans. Instead, 1) it creates a “tunnel” for each existing agent, that consists of the agent’s existing path and the neighboring locations within a specified “width”, and 2) it allows every existing agent to follow a path within their own tunnel while it revises their plans. While revising the plans of existing agents within their tunnels, 3) the Revise-and-Augment-in-Tunnels method computes plans for the new agents and augments these plans with the revised plans, respecting the collision constraints. As the tunnel width gets larger (resp. smaller), the Revise-and-Augment-in-Tunnels method gets closer to the Replan-All method (resp. the Revise-and-Augment method). Figure 5 shows sample tunnels with different width values for a sample agent. Note that a tunnel with a zero width contains only the path of the agent and the edges between them. Even though a zero width tunnel contains the path only, this method allows the agents to visit the vertices of their paths in a different order, unlike Revise-and-Augment. We implemented the Revise-and-Augment-in-Tunnels method using multi-shot ASP, and integrated it in our D-MAPF framework. We designed and performed experiments to better understand the strengths and the weaknesses of this new method, considering computational performance (in time) and quality of solutions (in terms of plan changes).

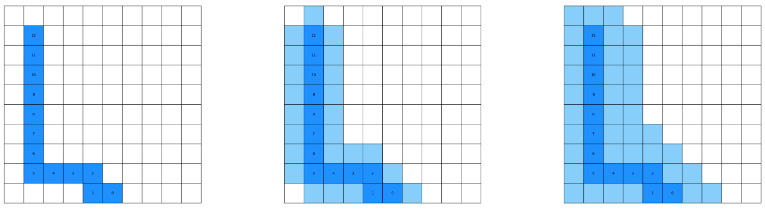


Figure 5: Tunnels with widths 0 (left), 1 (middle) and 2 (right).

Details of our general framework, the problem definition, ASP formulations, and experimental evaluations are can be found in in our paper *A General Framework for Dynamic MAPF using Multi-Shot ASP and Tunnels.* [16].

6. Conclusion and Future Work

We have introduced flexible, lifelong and explainable frameworks of MAPF problem, addressing different challenges. While mMAPF focuses on a flexible framework considering resource optimization and multi-modality, D-MAPF focuses on lifelong solutions considering dynamic changes in the environment using different methods. In addition to these, an explainable framework for mMAPF is also introduced, generating explanations about feasibility and optimality about given mMAPF solutions together with some observations or suggestions, via queries and counterfactual reasoning.

Currently, we are working on more extensive evaluations by considering additional benchmarks and evaluation metrics. We will conduct theoretical analysis for our methods by considering computational complexity and investigating correctness of our methods.

Besides investigating our existing methods further, we have progressed in a novel method for defining robustness of MAPF plans, to be able to address more challenges of real-life applications. We are evaluating our method with experiments.

In addition to new methods and problems, we aim to consider some real-life applications of MAPF and demonstrate our methods on some selected real-world applications. For this purpose, we are collaborating with a logistics company.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] D. Ratner, M. K. Warmuth, Finding a shortest solution for the n $\times$ n extension of the 15-puzzle is intractable, in: Proc. of AAAI, 1986, pp. 168–172.

[2] J. Lee, W. Yu, A coarse-to-fine approach for fast path finding for mobile robots, in: Proc. of IROS, 2009, pp. 5414–5419. doi:10.1109/IROS.2009.5354686.

[3] P. R. Wurman, R. D'Andrea, M. Mountz, Coordinating hundreds of cooperative, autonomous vehicles in warehouses, AI Magazine 29 (2008) 9–20. doi:10.1609/aimag.v29i1.2082.

[4] K. M. Dresner, P. Stone, A multiagent approach to autonomous intersection management, J. Artif. Intell. Res. (JAIR) 31 (2008) 591–695. doi:10.1613/jair.2502.

[5] T. S. Standley, R. E. Korf, Complete algorithms for cooperative pathfinding problems, in: Proc. of IJCAI, 2011, pp. 668–673. doi:10.5591/978-1-57735-516-8/IJCAI11-118.

[6] V. Marek, M. Truszczyński, Stable models and an alternative logic programming paradigm, in: The Logic Programming Paradigm: a 25-Year Perspective, Springer Verlag, 1999, pp. 375–398. doi:10.1007/978-3-642-60085-2_17.

[7] I. Niemelä, Logic programs with stable model semantics as a constraint programming paradigm, Annals of Mathematics and Artificial Intelligence 25 (1999) 241–273. doi:10.1023/A:1018930122475.

[8] V. Lifschitz, Answer set programming and plan generation, Artificial Intelligence 138 (2002) 39–54. doi:10.1016/S0004-3702(02)00186-8.

[9] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Generation Computing 9 (1991) 365–385. doi:10.1007/BF03037169.

[10] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: Proceedings of International Logic Programming Conference and Symposium, 1988, pp. 1070–1080.

[11] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, M. Schneider, Potassco: The potsdam answer set solving collection, AI Commun. 24 (2011) 107–124. doi:10.5555/1971622.1971623.

[12] J. McCarthy, Elaboration tolerance, in: Proc. of CommonSense, 1998.

[13] A. Bogatarkan, E. Erdem, A. Kleiner, V. Patoglu, 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, 2020, pp. 313–324. doi:10.1007/978-3-030-46212-3_24.

[14] A. Bogatarkan, E. Erdem, Explanation generation for multi-modal multi-agent path finding with optimal resource utilization using answer set programming, Theory Pract. Log. Program. 20 (2020) 974–989. URL: https://arxiv.org/abs/2008.03573. doi:10.1017/S1471068420000320.

[15] A. Bogatarkan, V. Patoglu, E. Erdem, A declarative method for dynamic multi-agent path finding, in: Proc. of the Global Conference on Artificial Intelligence, 2019, pp. 54–67. doi:10.29007/cnzw.

[16] A. Bogatarkan, E. Erdem, A general framework for dynamic mapf using multi-shot asp and tunnels, Theory and Practice of Logic Programming (2025). doi:10.48550/arXiv.2507.20703.

[17] D. Silver, Cooperative pathfinding, in: Proc. of AIIDE, 2005, pp. 117–122.

[18] P. E. Hart, N. J. Nilsson, B. Raphael, Correction to "a formal basis for the heuristic determination of minimum cost paths", SIGART Newsletter 37 (1972) 28–29.

[19] R. Jansen, N. Sturtevant, A new approach to cooperative pathfinding, in: Proc. of AAMAS, 2008, pp. 1401–1404.

[20] G. Sharon, R. Stern, A. Felner, N. R. Sturtevant, Conflict-based search for optimal multi-agent pathfinding, Artif. Intell. 219 (2015) 40–66.

[21] J. Yu, S. M. LaValle, Planning optimal paths for multiple robots on graphs, in: Proc. of ICRA, 2013, pp. 3612–3617.

[22] P. Surynek, A. Felner, R. Stern, E. Boyarski, Efficient SAT approach to multi-agent path finding under the sum of costs objective, in: Proc. of ECAI, 2016, pp. 810–818.

[23] E. Erdem, D. G. Kisa, U. Oztok, P. Schueller, A general formal framework for pathfinding problems with multiple agents, in: Proc. of AAAI, 2013.

[24] S. Almagor, M. Lahijanian, Explainable multi agent path finding, in: Proc. of AAMAS, 2020, pp. 34–42.

[25] H. Ma, S. Koenig, Optimal target assignment and path finding for teams of agents, in: Proc. of AAMAS, 2016, pp. 1144–1152.

[26] V. Nguyen, P. Obermeier, T. C. Son, T. Schaub, W. Yeoh, Generalized target assignment and path finding using answer set programming, in: Proc. of IJCAI, 2017, pp. 1216–1223.

[27] H. Ma, J. Li, T. K. S. Kumar, S. Koenig, Lifelong multi-agent path finding for online pickup and delivery tasks, in: Proc. of AAMAS, 2017, pp. 837–845.

[28] J. Svancara, M. Vlk, R. Stern, D. Atzmon, R. Bartak, Online multi-agent pathfinding, in: Proc. of AAAI, 2019.

[29] Q. Wan, C. Gu, S. Sun, M. Chen, H. Huang, X. Jia, Lifelong multi-agent path finding in a dynamic

environment, in: Proc. of ICARCV, 2018, pp. 875–882.

[30] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. K. S. Kumar, S. Koenig, Lifelong multi-agent path finding in large-scale warehouses, in: Proc. of AAAI, 2021, pp. 11272–11281.

[31] B. Atiq, V. Patoglu, E. Erdem, Dynamic multi-agent path finding based on conflict resolution using answer set programming, Electronic Proceedings in Theoretical Computer Science 325 (2020) 223–229. URL: http://dx.doi.org/10.4204/EPTCS.325.27. doi:10.4204/eptcs.325.27.

[32] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Multi-shot asp solving with clingo, Theory and Practice of Logic Programming 19 (2019) 27–82. doi:10.1017/S1471068418000054.