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

Aysu Bogatarkan

Sabanci University, Istanbul, Turkey
ORCID (Aysu Bogatarkan): https://orcid.org/0000-0003-3363-1987

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 is a combinatorial search problem that aims to find paths for multiple agents in an environment (e.g., robots in an autonomous warehouse) such that no two agents collide with each other or obstacles, and subject to some constraints on the plan length. MAPF with constraints on plan lengths is intractable [15]. 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 has been studied in various domains, such as robotics [10], autonomous warehouse systems [17], traffic control [5], and video games [16].

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) [12, 14, 11]—a logic programming paradigm based on answer sets [9, 8] and implement our methods using the ASP solver CLINGO [6].

Flexible Solutions for MAPF

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 with flexible frameworks in the spirit of elaboration tolerance [13].

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*.

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 [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. Given a solution for mMAPF, our explainable framework is able to explain 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. 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. [4], 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 counterfactual 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* [1].

Lifelong Solutions for MAPF

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.

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 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.

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* [3].

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. 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 [7] of ASP, unlike our previous work [3], 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, allowing 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, it creates a "tunnel" for each existing agent, that consists of the agent's existing path and the neighboring locations within a specified "width". It allows every existing agent to follow a path within their own tunnel while it revises their plans. At the same time, it 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). 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).

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.* [2].

Ongoing and Future Work

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.

References

[1] A. Bogatarkan and E. Erdem. Explanation generation for multi-modal multi-agent path finding with optimal resource utilization using answer set programming. *Theory Pract. Log. Program.*, 20(6):974–989, 2020. doi: 10.1017/S1471068420000320. URL https://arxiv.org/abs/2008.03573.

[2] A. Bogatarkan and E. Erdem. A general framework for dynamic mapf using multi-shot asp and tunnels. In *Proceedings of the 41st International Conference on Logic Programming (ICLP 2025)*, 2025.

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

[4] A. Bogatarkan, E. Erdem, A. Kleiner, and 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*, pages 313–324, 2020. doi: 10.1007/978-3-030-46212-3_24.

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

[6] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, and M. Schneider. Potassco: The potsdam answer set solving collection. *AI Commun.*, 24(2):107–124, Apr. 2011. ISSN 0921-7126. doi: 10.5555/1971622.1971623.

[7] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. Multi-shot asp solving with clingo. *Theory and Practice of Logic Programming*, 19(1):27–82, 2019. doi: 10.1017/S1471068418000054.

[8] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of International Logic Programming Conference and Symposium*, pages 1070–1080. 1988.

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

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

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

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

[13] J. McCarthy. Elaboration tolerance. In *Proc. of CommonSense*, 1998.

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

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

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

[17] P. R. Wurman, R. D'Andrea, and M. Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 29(1):9–20, 2008. doi: 10.1609/aimag.v29i1.2082.