Should Multi-Agent Path Finding Algorithms Coordinate Target Arrival Times?

Jonathan Morag, Noy Gabay, Daniel Koyfman, Roni Stern

Ben-Gurion University of the Negev
{moragj, noygab, koyfdan}@post.bgu.ac.il, roni.stern@gmail.com

Abstract

Multi-Agent Path Finding (MAPF) deals with finding conflict-free paths for a set of agents from an initial configuration to a given target configuration. The Lifelong MAPF (LMAPF) problem is a well-studied online version of MAPF in which an agent receives a new target when it reaches its current target. The common approach for solving LMAPF is to treat it as a sequence of MAPF problems, periodically replanning from the agents' current configurations to their current targets. A significant drawback of this approach is that in MAPF the agents must reach a configuration in which all agents are at their targets simultaneously. Coordinating the agents' paths such that they occupy their targets at the same time is needlessly restrictive for LMAPF. Techniques have been proposed to indirectly mitigate this drawback. In this position paper, we describe cases where these mitigation techniques fail. As an alternative, we propose to solve LMAPF problems by solving a sequence of modified MAPF problems, in which the objective is for each agent to eventually visit its target, but not necessarily for all agents to do so simultaneously. We refer to this MAPF variant as MAPF for Lifelong (MAPF4L) and propose how to solve it by modifying several existing MAPF algorithms. A limited experimental evaluation identifies some cases where using a MAPF4L algorithm can improve the system throughput significantly.

Code — github.com/J-morag/MAPF/releases/25.SoCS.MAPF4L

Introduction

The Multi-Agent Path Finding (MAPF) problem deals with finding conflict-free paths for a set of agents from an initial configuration to a given target configuration. MAPF problems manifest in real-world applications such as automated warehouses (Ma et al. 2017; Li et al. 2021b; Varambally, Li, and Koenig 2022; Morag, Stern, and Felner 2023), and a wide range of MAPF algorithms have been proposed (Wu et al. 2023). In many MAPF applications, it is often necessary to solve an online version of MAPF called *Lifelong MAPF (LMAPF)* (Li et al. 2021b; Švancara et al. 2019), where when an agent reaches its target, it is assigned a new target. LMAPF has attracted significant interest in both academia and industry, including a recent LMAPF competition sponsored by Amazon Robotics (www.LeagueOfRobotRunners.org). The common approach for solving LMAPF is to treat it as a sequence of MAPF problems, each dealing with finding paths from the agents' current configuration to their current targets.

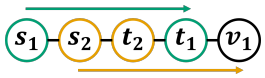

Figure 1: An LMAPF problem that cannot be solved as a sequence of MAPF problems.

A significant drawback of this approach stems from the fact that the agents in LMAPF are not required to reach a configuration where all agents are at their targets, since once an agent reaches its target it may receive a new one. For example, consider the LMAPF problem in Figure 1. Two agents a_1 and a_2 are positioned in vertices s_1 and s_2 and assigned targets t_1 and t_2, respectively. As an LMAPF problem, both targets can be reached: once a_2 reaches its target t_2, it receives a new target, and can choose to move to v_1 so that a_1 may also reach its target. Solving this LMAPF problem as a sequence of MAPF problems, however, will fail, because as a MAPF problem it has no solution, as the agents cannot swap positions in this graph.

Some prior works acknowledged this MAPF-LMAPF "mismatch" and proposed several techniques to mitigate it to some extent. These techniques include planning for a limited horizon (Li et al. 2021b), dynamic replanning whenever a target is reached (Varambally, Li, and Koenig 2022), defining "dummy paths" to follow after visiting a target (Liu et al. 2019), and assigning each agent with a sequence of targets (Grenouilleau, Van Hoeve, and Hooker 2019). We discuss these techniques and show they do not provide a complete solution to the MAPF-LMAPF mismatch.

As an alternative, we propose to solve LMAPF problems by solving a sequence of modified MAPF problems that we call *MAPF for Lifelong* (MAPF4L). In MAPF4L, the objective is to find conflict-free paths for the agents such that each agent visits its target at some point along its path. Importantly, there is no need to reach a configuration in which all agents are at their targets at the same time. We discuss the properties of the MAPF4L problem and adapt several popular MAPF algorithms to solve it. Finally, we conduct a small-scale experiment, showing the significant potential benefit of using MAPF4L algorithms in an LMAPF system.

Background

A classical *Multi-Agent Path Finding (MAPF)* problem is defined by a graph $G = (V, E)$; a set of agents A; and a source and target vertex for each agent $a_i \in A$, denoted s_i and t_i, respectively. A *solution* π to a MAPF problem is a mapping of each agent $a_i \in A$ to a path π_i in G such that π_i starts at s_i, ends at t_i, and does not *conflict* with any path that is mapped in π to a different agent (agents are allowed to wait in place). We consider as a conflict only *vertex* and *swapping* conflicts (Stern et al. 2019). A pair of paths π_i and π_j has a vertex conflict if there exists an index x such that $\pi_i[x] = \pi_j[x]$, where $\pi_i[x]$ is the x^{th} vertex in the path π_i. A swapping conflict occurs if there exists an index x such that $\pi_i[x] = \pi_j[x+1]$ and $\pi_i[x+1] = \pi_j[x]$. The length of a path is defined as the number of vertices (non-unique) in the path, minus one ($len(\pi_i) - 1$). *Sum Of Costs* (SOC) and *Makespan* are common MAPF solution cost function. SOC is the sum of the lengths of all paths in the solution and makespan is the length of the longest path in the solution. Some MAPF algorithms, such as Conflict Based Search (CBS) (Sharon et al. 2015) guarantee the solution they return is optimal w.r.t. SOC or makespan, while others, such as Prioritized Planning (PrP) (Bennewitz, Burgard, and Thrun 2001) do not.

Lifelong Multi-Agent Path Finding (LMAPF) (Li et al. 2021b; Morag, Stern, and Felner 2023) is a generalization of MAPF in which whenever an agent reaches its target, it may be assigned a new target to reach. Due to its online nature, an LMAPF problem is typically solved by repeatedly interleaving planning and execution. In every *planning period* a path for each agent is computed. Then, a prefix of the planned paths is executed, and another planning period starts. This process continues indefinitely until interrupted by the user. The performance of an LMAPF algorithm is commonly evaluated using *throughput*, which counts how many times agents reached their targets before a fixed number of time steps have passed. A common approach for solving LMAPF is by iteratively calling a MAPF solver every planning period to find paths for the agents from their current locations to their current targets. Different MAPF solvers have been used for this purpose, including optimal and suboptimal MAPF algorithms.

MAPF-LMAPF Mismatch and Mitigations

The role of an LMAPF algorithm is to output plans for the agents in every planning period. We say that an LMAPF algorithm is *complete* if it does so, and incomplete otherwise. The example in Figure 1 shows that the common approach of solving LMAPF as a sequence of MAPF problems yields incomplete LMAPF algorithms. Several techniques have been proposed to mitigate this incompleteness to some extent.

Dynamic replanning This means whenever a target is reached, all agents immediately replan (Varambally, Li, and Koenig 2022). This option is not feasible in some cases since it requires significant computational cost. Moreover, this approach does not solve the problem in Fig. 1 as the agents will fail to find any MAPF solution at time 0.

Pre-assigned sequence of targets This means each agent is given a sequence of targets to reach and the LMAPF algorithm returns plans to visit the next couple of targets (Grenouilleau, Van Hoeve, and Hooker 2019). In many cases, however, the path planner does not know which tasks will arrive in the future. Also, the planning itself is harder when planning to visit more than one node. Moreover, this technique is also incomplete in some cases.

Dummy paths This means that after an agent reaches its target, it will follow a pre-defined path so as to avoid causing congestion near its target (Liu et al. 2019). This mitigation strategy still suffers the MAPF limitation that a specific configuration of the agents must be reached. Indeed, identifying effective dummy paths is not trivial, and poorly located dummy paths may even create new incompleteness scenarios (similar to Fig. 1). This technique may also make planning more difficult, as it requires longer paths.

Well-formed instances This involves modifying the environment and tasks such that agents can always stay at their targets indefinitely without blocking any other agents (Čáp, Vokřínek, and Kleiner 2015). However, modifying the environment and tasks is often not possible, as those may be a consequence of various factors beyond our control (warehouse/city layouts, inventory management, etc.)

Limited planning horizon This means ignoring conflicts that are further than w time steps in the future when planning, where w is a parameter. This technique is embodied in the Rolling Horizon Collision Resolution (RHCR) framework (Li et al. 2021b), which is the state of the art in LMAPF. Common implementations of RHCR use LNS (Li et al. 2021a) or PrP for planning, but any MAPF algorithm can be easily adapted for this purpose. Using a limited planning horizon is common, yet it may lead to deadlocks and to inefficiencies stemming from myopic planning. Fig. 3 shows an example where this occurs. We discuss this example later.

Fail policies Morag et al. (2023) proposed an LMAPF framework designed to handle planning periods in which the underlying MAPF algorithm failed to return a plan. They proposed several *fail policies* for this purpose and techniques for utilizing partial solutions the MAPF algorithm may return. An LMAPF algorithm with a fail policy can be viewed as a complete LMAPF algorithm, yet the fail policies proposed so far are ad-hoc and may be inefficient.

In summary, while current techniques partially reduce the impact of the MAPF-LMAPF "mismatch", they do so indirectly and often in an incomplete manner. In this work, we propose a more direct approach to address this mismatch by modifying the type of MAPF problem being solved in every planning period. We call this modified MAPF problem *MAPF for Lifelong* (MAPF4L) and define it below.

MAPF for Lifelong (MAPF4L)

The input to a MAPF4L problem is the same as the input for MAPF. The solution to a MAPF4L problem is also a mapping of each agent to a path in G. The key difference between MAPF and MAPF4L is that in a solution to a MAPF

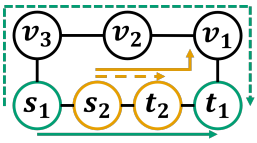

Figure 2: Example of solutions with different SOC, SST, Makespan, and MKST costs.

problem the path π_i mapped to each agent a_i must *end* at t_i, in a MAPF4L solution it suffices that π_i *includes* t_i. Based on existing complexity results for MAPF (Nebel 2020; Yu and LaValle 2013), we proved that finding optimal solutions to MAPF4L is also NP-Hard for common cost functions (as is MAPF). Proofs are in the supplementary materials.

Solution Cost Functions

Standard MAPF solution cost functions such as SOC and makespan can be used for MAPF4L. As we show below, there also is a mismatch between these cost functions and the LMAPF performance measure, throughput. We propose two cost functions more appropriate for MAPF4L: *Sum of Service Times (SST)* and *Makespan Service Time (MKST)*. Both functions are based on the notion of *Service Time* (Okumura et al. 2022), which is the time elapsed between when an agent received a target and when it first visited it. SST is similar to SOC, but instead of summing the length of each path, we sum the number of time steps it took each agent to visit its target for the first time in its path. Formally,

$$SST(\pi) := \sum(\min(x|\pi_i[x] = t_i)|\pi_i \in \pi)$$

MKST is similar to Makespan, but instead of finding the maximal length of any path, we find the maximal number of time steps it took any agent to reach its target in its path.

To demonstrate how these new cost functions are more appropriate in the context of LMAPF, consider Figure 2, which shows solutions for a MAPF problem and a MAPF4L problem that is given the same input. There are two agents, a_1 and a_2, where s_1 and s_2 are their sources and t_1 and t_2 are their targets, respectively. In the MAPF solution illustrated with dashed arrows, a_1 travels through the top vertices to reach its target, taking the path $[s_1, v_3, v_2, v_1, t_1]$ (in green). Agent a_2 travels directly to its target, taking the path $[s_2, t_2]$ (in orange). The SOC of this solution is $4 + 1 = 5$, the SST is $4 + 1 = 5$, the Makespan is $\max(4, 1) = 4$, and the MKST is $\max(4, 1) = 4$. This solution is optimal (minimal) in terms of its SOC and Makespan. In the MAPF4L solution, illustrated with solid arrows, a_1 travels directly to its target, taking the path $[s_1, s_2, t_2, t_1]$ (in green). Agent a_2 visits its target once at time step 1, but then continues advancing to make way for a_1, taking the path $[s_2, t_2, t_1, v_1]$ (in orange). The SOC of this solution is $3 + 3 = 6$, the SST is $3 + 1 = 4$, the Makespan is $\max(3, 3) = 3$, and the MKST is $\max(3, 1) = 3$. This solution is optimal in terms of its SST, MKST, and Makespan. Commonly, LMAPF algorithms prefer a solution with lower SOC. This solution has a higher SST, so agents will arrive at their targets unnecessarily late. Repeated over time, this could lower throughput.

Solving the MAPF4L Problem

One may view MAPF4L as a degenerated version of LMAPF, where when an agent reaches its target, it is not given a new target, and use LMAPF algorithms to solve it. However, most LMAPF algorithms are not designed to plan paths for agents without a target, while this is needed for solving MAPF4L problems (see Fig. 1).

The Priority Inheritance with Backtracking (PIBT) algorithm (Okumura et al. 2022) is a notable exception. PIBT solves the *Iterative MAPF* problem, which is a generalization of LMAPF in which each agent receives multiple targets to reach. In every iteration, PIBT chooses the configuration the agents should move to in the next time step by using a sophisticated version of PrP with a one-step lookahead. PIBT can easily be adapted to solve MAPF4L problems by simply not assigning agents with new targets after reaching their current target. Instead, it can associate such agents with a minimal priority, allowing other agents to move them as needed. Moreover, under certain conditions, PIBT guarantees that each agent will reach its target eventually (Okumura et al. 2022). Thus, PIBT solves MAPF4L problems, except that they were not defined as such as the emphasis was on solving the larger LMAPF problem. PIBT was also designed to solve MAPF problems. Once an agent reaches its target, PIBT continuously re-assigns it to the agent, hopefully reaching a configuration where all agents are at their targets simultaneously. Aiming to stay at the target after visiting it is not necessarily beneficial for solving MAPF4L.

PrP, LNS, and CBS for MAPF4L

The one-step lookahead planning done by PIBT often limits its effectiveness in complex scenarios with narrow corridors, which are common in automated warehouses. Other MAPF algorithms such as PrP, LNS, and CBS consider longer planning horizons by using A* (Hart, Nilsson, and Raphael 1968) to find optimal or close to optimal paths for individual agents. A state in the search space of this A* is a pair of location and time ($\langle v, x \rangle$).

To use these MAPF algorithms for MAPF4L, we extend the state definition in A* to include a Boolean flag indicating whether or not any of the state's ancestors visited the agent's target ($\langle v, x, b \rangle$). This flag is initially false. It is set to true in a state if it contains the target or if the flag was true in the parent of that state. These flags are used in two places. First, two states that have the same time and vertex, but differ in having an ancestor that visited the target, must be considered distinct, and neither must be discarded as a duplicate of the other. Otherwise, the search may miss valid paths. Second, the definition of a goal state is modified to require that the state had an ancestor that visited the target ($\langle v, x, true \rangle$), instead of requiring that the vertex is equal to the target.

Case Studies

We conducted a limited experimental evaluation of our MAPF4L algorithms on hand-crafted LMAPF scenarios. In each experiment, we compare the performance of solving LMAPF problems using either a MAPF or a MAPF4L algorithm in every planning period. Specifically, we used PIBT,

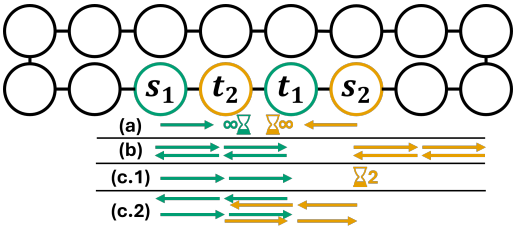

Figure 3: An LMAPF problem in which pathological behaviors occur under different conditions and solvers.

PrP, and CBS and the MAPF4L versions of PrP and CBS we developed, denoted as PrP$_L$ and CBS$_L$. All experiments were run on Linux virtual machines in a cluster of AMD EPYC 7763 CPUs, with 16GB of RAM each.

Case Study 1 The first LMAPF problem we consider is illustrated in Fig. 1. Agent a_1 starts at s_1 and is assigned targets going back and forth between s_1 and t_1, indefinitely. The same is done for agent a_2 with vertices s_2 and t_2. For example, a_1 will be assigned targets $[t_1, s_1, t_1, s_1 ...]$. As discussed above, limiting the planning horizon using a MAPF solver results in planning failure, as the agents must swap their positions, which is impossible. Limiting the planning horizon by using RHCR helps mitigate this problem, which raises the question — is this a substantial problem in LMAPF? We solved this LMAPF problem with a planning horizon of 10, running for 500 time steps, and using PrP, PrP$_L$, and PIBT. We found that PrP and PrP$_L$ both achieved a throughput of 500, whereas PIBT achieved a throughput of 320. In comparing A* expansions, PrP required a total of 13,509 expansions, whereas PrP$_L$ required only 11,965 expansions. The total runtime was 56ms for PrP, 50ms for PrP$_L$, and 163ms for PIBT. This experiment demonstrates how, even when using RHCR, the new MAPF4L solver, PrP$_L$, had advantages in runtime over the MAPF solver PrP, and in throughput over the existing MAPF4L solver PIBT.

Case Study 2 The second LMAPF problem we consider is illustrated in Figure 3. Here there are two agents, a_1 and a_2, tasked with moving repeatedly between s_1 and t_1, or s_2 and t_2, respectively. All agents are allowed to re-plan at every time step. Consider a planning horizon of 1 is used. Both the MAPF and MAPF4L solvers will behave as seen in Figure 3(a), moving towards each other, and then waiting in place indefinitely, always assuming that once the planning horizon passes they would be able to swap their positions and reach their targets. This results in a throughput of 0 for both solvers. The same would happen with a horizon of 2 for the MAPF4L solver, and any horizon ≤ 5 for the MAPF solver. Second, assume an infinite planning horizon. The behavior of the MAPF solver is illustrated in Figure 3(b), where it will have a_1 move counter-clockwise towards its target, while a_2 attempts to reach its target by moving counter-clockwise. After two time steps, agent a_1 reaches its target and receives its next target, s_1. This causes a_2 to start moving clockwise, to avoid disrupting agent a_1. This repeats indefinitely, resulting in starvation of agent a_2, and a throughput of $timesteps \div 2$. This will be true for any horizon larger than 6. Conversely, the MAPF4L solver will have a_2 wait in place for a_1 to reach t_1, as the solver is aware that a_1 will not be blocking t_1 indefinitely after reaching it. This is illustrated in Figure 3(c.1). After agent a_1 reaches t_1 and receives its new target of s_1, both agents will start moving clockwise in unison, and reach their new targets after two time steps. They will then move counter-clockwise, and repeat this cycle indefinitely as seen in Figure 3(c.2), resulting in a throughput of $timesteps - 1$. The same will hold true for any horizon larger than 2. We also studied this behavior experimentally using MAPF and MAPF4L versions of CBS and PrP, as well as PIBT. We ran the LMAPF problem for 500 time steps with an infinite planning horizon, and found that the total runtime was 29,198ms for CBS, 33ms for CBS$_L$, 137ms for PrP, 68ms for PrP$_L$, and 80ms for PIBT. Throughput was 250 for CBS, 499 for CBS$_L$, 250 for PrP, 499 for PrP$_L$, and 499 for PIBT. This experiment demonstrates how MAPF4L algorithms are useful even while employing existing mitigation for the MAPF-LAMPF mismatch, achieving a higher throughput while requiring less runtime.

Case Study 3. Finally, we consider LMAPF problems on grids from the standard MAPF benchmark (Stern et al. 2019), under the frameworks described by Li et al. and Morag, Stern, and Felner. The number of time steps between planning iterations was set to 5, and the planning horizon (w) was set to 10, a configuration that was shown to be generally effective. The amount of (real) time for planning at each planning iteration was 5 seconds, and we randomly sampled from a restricted set of 10, 20, 30, or 40 targets. This simulates cases where there are specific locations of interest, such as picking or charging stations. Finally, we measured the throughput after 1,000 time steps. The results show that PrP$_L$ generally yields a relatively higher throughput when the number of possible targets is limited. On warehouse-20-40-10-2-1 with 500 agents and 20 targets, the throughput of PrP$_L$ was 549, whereas that of PrP was only 491. These results highlight the potential of using MAPF4L algorithms in such hard and dense LMAPF problems.

Conclusion and Future Work

In this work, we proposed solving LMAPF problems by solving a sequence of MAPF variants that we call MAPF for Lifelong (MAPF4L), in which agents do not have to stay at their targets after visiting them. We described that such a variant is needed to address a mismatch between the requirements of LMAPF and MAPF, and showed how to adapt existing MAPF algorithms to solve it. We evaluated using our MAPF4L algorithms within RHCR to solve LMAPF. Our results demonstrated that in pathological cases and when the targets are limited to a fixed small number of locations, using the MAPF4L algorithms is important, often achieving higher system throughput than their MAPF counterparts. Future work may generalize MAPF4L to applications where agents must spend some amount of time at their target to complete some task.

Acknowledgments

This research was partly supported by the Helmsley Charitable Trust through the Agricultural, Biological and Cognitive Robotics Initiative and by the Israel Science Foundation (ISF) grant #1238/23 to Roni Stern.

References

Bennewitz, M.; Burgard, W.; and Thrun, S. 2001. Optimizing schedules for prioritized path planning of multi-robot systems. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, 271–276.

Čáp, M.; Vokřínek, J.; and Kleiner, A. 2015. Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures. In *Proceedings of the international conference on automated planning and scheduling*, volume 25, 324–332.

Grenouilleau, F.; Van Hoeve, W.-J.; and Hooker, J. N. 2019. A multi-label A* algorithm for multi-agent pathfinding. In *International conference on automated planning and scheduling*, 181–185.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P.; and Koenig, S. 2021a. Anytime multi-agent path finding via large neighborhood search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. S.; and Koenig, S. 2021b. Lifelong multi-agent path finding in large-scale warehouses. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 11272–11281.

Liu, M.; Ma, H.; Li, J.; and Koenig, S. 2019. Task and path planning for multi-agent pickup and delivery. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Ma, H.; Li, J.; Kumar, T.; and Koenig, S. 2017. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *International Conference on Autonomous Agents and Multiagent Systems*.

Morag, J.; Stern, R.; and Felner, A. 2023. Adapting to Planning Failures in Lifelong Multi-Agent Path Finding. In *Proceedings of the International Symposium on Combinatorial Search*, volume 16, 47–55.

Nebel, B. 2020. On the computational complexity of multi-agent pathfinding on directed graphs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 212–216.

Okumura, K.; Machida, M.; Défago, X.; and Tamura, Y. 2022. Priority inheritance with backtracking for iterative multi-agent path finding. *Artificial Intelligence*, 310: 103752.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial intelligence*, 219: 40–66.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. S.; et al. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Symposium on Combinatorial Search (SoCS)*.

Švancara, J.; Vlk, M.; Stern, R.; Atzmon, D.; and Barták, R. 2019. Online multi-agent pathfinding. In *AAAI conference on artificial intelligence*, volume 33, 7732–7739.

Varambally, S.; Li, J.; and Koenig, S. 2022. Which MAPF model works best for automated warehousing? In *Proceedings of the international symposium on combinatorial search*, volume 15, 190–198.

Wu, M.; Yan, W.; Hasan, H.; and Jamaluddin, R. A. 2023. A Review of Multi-Agent Path Finding Algorithms. In *2023 11th International Conference on Information Systems and Computing Technology (ISCTech)*, 69–73. IEEE.

Yu, J.; and LaValle, S. 2013. Structure and intractability of optimal multi-robot path planning on graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 27, 1443–1449.