On Path Selection for Reduction-Based Solving of Multi-Agent Pathfinding Using Graph Pruning

Matej Husár, Jiří Švancara, Roman Barták
Charles University, Czech Republic
husarmatej@gmail.com, svancara@ktiml.mff.cuni.cz, bartak@ktiml.mff.cuni.cz

Abstract

Multi-agent pathfinding is the task of navigating a set of mobile agents in a shared environment such that they avoid collisions. Finding an optimal solution in terms of the length of the plan is known to be a computationally hard problem (NP-Hard). In general, there are two schools of optimal algorithms: search-based and reduction-based. While search-based algorithms excel in solving large maps where few conflicts can be expected, reduction-based algorithms excel in smaller instances even when agents interact often. However, the reduction-based approaches lag behind in large instances, even with few agents. To mitigate this, a subgraph pruning method was introduced to prune unnecessary vertices to decrease the size of the instance. The pruning is based on the shortest paths for each agent. In the original study, the authors randomly selected the shortest routes. In this study, we replicate the overall approach while selecting the initial shortest path with more care. We provide several approaches for selecting one of the possible shortest paths and experimentally compare them. We note that when the makespan optimal plan is needed, not all agents are required to use the shortest path, as only the longest path dictates the makespan. Using this observation, we also introduce an approach that selects longer paths for some agents if it helps to reduce the total number of interactions between agents. We provide an experimental comparison of all proposed approaches and show that the latter performs significantly better, in most cases outperforming any approach that strictly selects only the shortest path.

Introduction

Multi-agent pathfinding (MAPF) is the task of navigating a set of mobile agents in a shared environment from their starting location to desired destination locations while avoiding collisions (Stern et al. 2019). This abstract problem has numerous practical applications in warehousing (Wurman, D'Andrea, and Mountz 2008), airplane taxiing (Morris et al. 2016), video game control (Silver 2005), and traffic junctions (Dresner and Stone 2008).

The problem of finding an optimal MAPF solution has been shown to be computationally hard for a wide range of cost objectives. Although there exist domain-specific optimal solvers, an alternative popular approach in theoretical computer science is to translate hard problems into a well-established formalism and to use an existing, highly optimized solver for that formalism, for example, the Boolean Satisfiability Decision Problem (SAT) (Biere et al. 2009).

Related Work

In this paper, the focus is on optimal solvers. In general, there are two main categories that optimal solvers fall into.

Search-based solvers make use of search algorithms. The most notable algorithm is Conflict-Based Search (CBS) (Sharon et al. 2015) and many of its improvements (Boyarski et al. 2015; Gange, Harabor, and Stuckey 2019). CBS plans each agent individually (using single-agent A*) and examines the solution. If any pair of agents is conflicting, it can be resolved in two ways – either the first agent is not allowed in that position at that time, or the other agent is not allowed in that position at that time. This creates a binary constraint tree that CBS searches over.

Reduction-based solvers translate the MAPF instance into a selected established formalism. In the literature, translation to SAT (Barták and Švancara 2019) or ASP (Achá et al. 2021) is prominent, but other formalisms also provide specific advantages described in a survey paper (Surynek 2022). It is known that both search-based and reduction-based solvers excel in different types of instances (Švancara et al. 2024). While search-based approaches excel in large sparse environments, reduction-based approaches excel in small densely occupied environments. Since reduction-based approaches have trouble scaling up in large environments, improvements have been proposed to mitigate large maps and remove unneeded vertices, creating a subgraph (Husár et al. 2022). In this paper, we build on top of the subgraph method using reduction to SAT. Both of these concepts will be explained in more detail in later sections.

Contributions

In this paper, we explore the subgraph method designed to improve the scalability of reduction-based MAPF solvers under the makespan objective. To this end, our contributions are as follows.

We develop in total four different schemes to find the shortest paths that serve as the starting point for the subgraph method. Additionally, since the optimized cost function is makespan, we develop a scheme that finds the shortest path for only the critical agent that dictates the makespan and allows the others to find a longer path, provided that it minimizes the number of potential conflicts. Lastly, the proposed approaches are experimentally compared. Based on the measured data, we conclude that a correct path selection makes a large portion of the instances trivial to solve optimally without invoking the solver, as the improved path selection produces a conflict-free plan.

Definitions

A *MAPF instance* $\mathcal{M}$ is a pair (G, A), where G is a graph $G = (V, E)$ representing the shared environment and A is a set of mobile agents. An agent $a_i \in A$ is a pair $a_i = (s_i, g_i)$, where $s_i \in V$ is the starting location and $g_i \in V$ is the destination location of agent a_i.

Our task is to find a *valid plan* π_i for each agent $a_i \in A$ being a valid path from s_i to g_i. We use $\pi_i(t) = v$ to denote that agent a_i is located at the vertex v at the time step t. Time is discrete and at each timestep every agent can either wait at its current location or move to a neighboring location. Furthermore, we require that each pair of plans π_i and π_j, $i \neq j$ is collision-free. Specifically, we forbid *vertex* and *swapping* conflicts. Vertex conflict occurs when two agents are located in the same vertex at the same time (i.e. $\pi_i(t) = \pi_j(t)$). Swapping conflict occurs when two agents are moving over the same edge in the opposite direction at the same time (i.e. $\pi_i(t) = \pi_j(t+1) \wedge \pi_i(t+1) = \pi_j(t)$).

We are interested in *makespan* optimal solutions. Makespan refers to the length of a plan. Once an agent arrives at its destination location, it does not disappear. An agent may move out of the destination location again; however, a plan ends once all agents are at the destination location at the same time. This means that the length of the plan $|\pi_i|$ is the same for all agents. Another common cost function is *sum of costs* (Sharon et al. 2011). Note that finding an optimal solution for either of the cost functions is an NP-hard problem (Yu and LaValle 2013; Surynek 2015).

Solving MAPF via Reduction to SAT

As mentioned in the Introduction, this paper builds on top of existing SAT-based makespan optimal solver (Barták and Švancara 2019) and subgraph method (Husár et al. 2022). In this section, we describe both in more detail.

SAT Encoding

For each agent a_i in timestep t, Boolean variables $At(a_i, v, t)$ and $Pass(a_i, u, v, t)$ are created to represent the location at the vertex v and the movement over edge (u, v), respectively. Given a bound on the makespan T, the variables are created for each timestep $\{0, \ldots, T\}$. A formula restricting the movement of the agents is created and forwarded to an underlying SAT solver. If the SAT solver finds a solution, there is a solution to the MAPF instance. If the formula is unsatisfiable, the T is increased by one, and a new formula is constructed. The initial value of T is the lower bound on the makespan – the longest of the distances between s_i and g_i, formally, $LB_{mks} = \max_{a_i \in A} dist(s_i, g_i)$. Iteratively increasing T guarantees finding a makespan-optimal solution.

The following formulas describe the constraints in CNF.

$$At(a_i, s_i, 0) \tag{1}$$

$$At(a_i, g_i, T) \tag{2}$$

$$\forall v, u \in V, u \neq v : \neg At(a_i, v, t) \vee \neg At(a_i, u, t) \tag{3}$$

$$At(a_i, v, t) \implies \bigvee_{(v,u)\in E} Pass(a_i, v, u, t) \tag{4}$$

$$Pass(a_i, v, u, t) \implies At(a_i, u, t+1) \tag{5}$$

$$\forall a_i, a_j \in A, a_i \neq a_j : \neg At(a_i, v, t) \vee \neg At(a_j, v, t) \tag{6}$$

$$\begin{aligned} &\forall a_i, a_j \in A, \forall (v, u) \in E : \\ &\qquad \neg Pass(a_i, v, u, t) \vee \neg Pass(a_j, u, v, t) \end{aligned} \tag{7}$$

The unit clauses 1 and 2 ensure that each agent starts at its start location and ends at the destination location, respectively. Clauses 3 ensure that each agent is located on the map just once and is not duplicated. Clauses 4 and 5 ensure that the movement of the agent follows a path; specifically, if an agent is present in a vertex, it leaves through one of the edges, and if an agent is moving over an edge, it arrives at the corresponding vertex in the next timestep. Lastly, 6 and 7 prohibit vertex and swapping conflicts, respectively.

The Subgraph Method

The subgraph method (Husár et al. 2022) is a framework designed to improve the scalability of reduction-based solvers by removing vertices from the graph. Given a MAPF instance $\mathcal{M} = (G, A)$ and a set of ground vertices, it creates a relaxed instance $\mathcal{M}_{k,m}$, where k is the distance from the ground vertices and m is the allowed cost to be added to the LB_{mks}. In the original paper, the ground vertices are selected as vertices that are on a randomly selected shortest path from s_i to g_i for each agent, then k dictates how many vertices are added to the subgraph, while m dictates the allowed cost T. See Figure 1 for an example with one agent. Note that the increased k draws contours around the selected ground vertices which fall into $k = 0$.

In the original paper, four strategies to iteratively explore different relaxed instances were described. In this paper, we will focus only on the ones that have been proven to produce makespan-optimal solutions. The baseline strategy $\mathbf{B}$ always takes the whole original graph G (using some sufficiently large k_{max}) and only iteratively increases m. This strategy is equivalent to the vanilla SAT-based makespan-optimal solvers. The strategy $\mathbf{P}$ starts with the smallest possible instance $\mathcal{M}_{0,0}$. In case of unSAT, k is iteratively increased until k_{max} is reached. Only after that, m increases by one and k is restarted again at 0. It is proven that such a

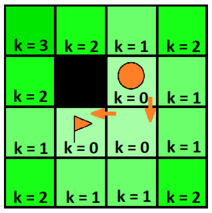

Figure 1: An instance with a single agent. Each vertex is labeled into which k-restricted graph it belongs. The black square is an impassable obstacle. Figure taken from (Husár et al. 2022).

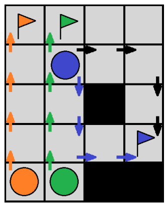

Figure 2: An example instance where the blue agent has two choices of the shortest path. If the blue path is chosen, the proposed strategies perform worse. Figure taken from (Husár et al. 2022).

strategy guarantees finding an optimal solution and has been experimentally shown to outperform the baseline approach.

A follow-up study (Svancara et al. 2023) focused on selecting more than a single shortest path for each agent to the set of ground vertices. However, the results show that increasing the number of vertices prolonged the computation time. Therefore, we will make use of the initial strategy of selecting just a single path for each agent.

Path Selection

The original study on the subgraph method chooses the initial ground vertices as any of the shortest paths for each agent. Given that these vertices are never changed in the algorithm, a more careful selection should be made. In this paper, we describe several schemes for selecting the set of vertices for each agent. Note that each scheme is based on some path from s_i to g_i, since the vertices need to ensure that each agent still can reach its goal.

Biased Random The baseline scheme comes from the original paper, where any shortest path may be selected. However, exploring the code shows that the shortest path algorithm explores neighboring vertices in fixed order, left, down, right, and up [1], thus preferring paths that move in a straight line as much as possible, and all agents tend to navigate around obstacles in the same direction. This phenomenon is most noticeable in the empty parts of the map. We refer to this scheme as *Biased random* or *Bia* for short.

Random To mitigate the biased factor of the previous scheme, we implemented an algorithm that selects truly random shortest paths for each agent. We refer to this scheme as *Random* or *Ran* for short.

Minimizing Crossing A better path selection may influence the complexity of the solving, as is demonstrated in Figure 2. Here, the blue agent has two possible shortest paths. Choosing the black one, the relaxed instance $\mathcal{M}_{0,0}$ is solvable. Selecting the blue path makes $\mathcal{M}_{0,0}$ unsolvable, and only after two iterations of the **P** strategy, $\mathcal{M}_{2,0}$ is solvable.

Therefore, we propose a scheme that tries to find for each agent the shortest path while crossing already selected paths the fewer number of times. This is achieved by the single-agent A* algorithm with a tie-breaking rule. The first minimized cost is the true distance, while the second minimized cost (i.e. tie-breaking) is the number of vertices selected by other agents. In the example in Figure 2, selecting the path for the blue agent, after the green and orange agents have selected their paths, it is guaranteed that the black path is selected.

Note that this algorithm is greedy in the sense that the agents select their paths in a fixed order. The algorithm does not minimize the global number of crossings, as that is a computationally hard problem, as it is related to the crossing number (Garey and Johnson 1983). We refer to this scheme as *Without crossing* or *WCr* for short.

Minimizing Conflicts The previous scheme may search for paths that do not cross unnecessarily. Since the MAPF problem has a time dimension, it is only required that the agents not meet at the same time. Given a shortest path, each vertex is associated with a time step when the agent is required to be there. Therefore, we upgrade the previous algorithm to first minimize the true distance, and second minimize the number of vertices that would cause a collision, taking the time into account. We refer to this scheme as *Without conflicts* or *WCo* for short.

Extended Minimizing Conflicts Given that the optimized cost function is makespan, it is not strictly necessary for all agents to select the *shortest* path. Recall that makespan is dictated by the longest plan. Using this information, we design a path selection scheme that first finds the lower bound on the makespan LB_{mks}. Again, we find paths for each agent in a fixed order using A*, however, the optimization cost is changed. First, minimize the number of positions that would cause a conflict, taking the time into account; second, minimize the true distance, while not allowing any agent to find a path longer than LB_{mks}. Note that since LB_{mks} is already determined, we are guaranteed to find a path for each agent, preferring the paths that cross other agents in time the least number of times. The order of the agents is taken in the decreasing order of $dist(s_i, g_i)$. The search-space of the A* consists both of space and time, as exploring a vertex in different time may lead to fewer conflicts, thus lower cost. We refer to this scheme as *Extended without conflicts* or *ExWCo* for short. Comparing *ExWCo* with the subgraph method to Prioritized Planning (Silver 2005), there are two

[1]The experiments are performed on a 4-connected grid graphs

Size	Type	B	P				
			Bia	Ran	WCr	WCo	ExWCo
32	empty	0.63	0.785	0.795	0.7	0.79	**1**
	maze	0.714	0.714	0.77	0.745	0.783	**0.925**
	random	0.985	0.99	0.985	0.985	0.985	**1**
	room	0.875	**0.89**	0.85	0.835	0.865	0.87
64	empty	0.1	0.41	0.38	0.355	0.375	**1**
	maze	0.134	0.263	0.279	0.268	0.274	**0.983**
	random	0.077	0.332	0.316	0.281	0.332	**1**
	room	0.2	0.455	0.43	0.38	0.435	**1**
128	empty	0	0.175	0.19	0.2	0.145	**1**
	maze	0	0.026	0.037	0.026	0.021	**1**
	random	0.02	0.175	0.16	0.15	0.14	**1**
	room	0.025	0.17	0.19	0.175	0.175	**1**

Table 1: Success rate of schemes using strategies $\mathbf{B}$ and $\mathbf{P}$.

Size	Type	P				
		Bia	Ran	WCr	WCo	ExWCo
32	empty	0.05	0.065	0.075	0.07	**1**
	maze	0.006	0.012	0.025	0.025	**0.845**
	random	0.04	0.045	0.065	0.035	**0.6**
	room	0.025	0.03	0.035	0.035	**0.55**
64	empty	0.095	0.105	0.135	0.1	**1**
	maze	0.028	0.034	0.028	0.034	**0.972**
	random	0.031	0.036	0.082	0.046	**1**
	room	0.035	0.06	0.04	0.035	**1**
128	empty	0.14	0.15	0.18	0.115	**1**
	maze	0.021	0.032	0.021	0.016	**1**
	random	0.095	0.085	0.075	0.05	**1**
	room	0.055	0.095	0.065	0.05	**1**

Table 2: Ratio of instances solved in the preprocess phase.

significant differences. First, the length of the allowed path to be found is bounded by LB_{mks}; second, *ExWCo* allows us to find conflicting paths with the promise from the subgraph method that these conflicts will be resolved, producing a complete (and optimal) algorithm.

Empirical Evaluation

To test and compare the proposed path selection schemes, we reimplement the subgraph method and reduction-based MAPF solver. The used underlying SAT solver is Kissat (Biere et al. 2020). All of the code is implemented in C++ and run on a PC with AMD Ryzen™ 9 5900X CPU and a limit of 56 GB of RAM. The code and more detailed results are provided in online repository[2].

Instances

The test instances are inspired by the well-known MAPF benchmark set (Stern et al. 2019). We selected map types *empty*, *maze*, *random*, and *room* with increasing sizes of 32×32, 64×64, and 128×128. We start with 5 randomly placed agents to create an instance; if the instance is solved in a given timeout of 30s, additional 5 agents are introduced to produce the next instance. Each setup was performed 10 times with a new random seed, i.e. placement of agents. All of the proposed schemes were tested on the same instances.

Results

The measured success rate is shown in Table 1. Instances that were not solved by any strategy or path selection scheme were not included. From the table, we can see that the *ExWCo* is the most successful scheme in most cases, being able to solve 100 agents in all map sizes. For the maps of size 32×32, the difference between the schemes is the lowest. The easiest map type was *random*, while the *maze* is the most difficult one. *Room* size 32×32 is the only map where the *Bia* scheme outperforms the *ExWCo*, on average the most agents solved being 95 and 93, respectively. All of the schemes are heuristic and depend on the order of agent selection. If the path selection does not produce a conflict-free solution, the SAT-based MAPF solver needs to resolve the remaining conflicts. In some cases, the random algorithm managed to find better initial paths that were either conflict-free or were easier for the MAPF solver to handle. Determining which conflicts are easier for the MAPF algorithm to handle is an open question. However, as can be seen, loosening the restriction on the path length in *ExWCo* helps tremendously on larger maps. As the map size grows, the *ExWCo* shows a superior success rate to all other schemes, while all of the others perform comparatively with a success rate only up to 20%.

Together, *ExWCo* solved 2284 instances, while the second most successful scheme *Bia* solved 1048. The improved performance of $\mathbf{P}$ over $\mathbf{B}$ is expected and in accordance with the original subgraph paper.

If the selected paths are conflict-free, it is guaranteed to be valid and optimal solution and the underlying solver is not required to be invoked. Table 2 shows the ratio of all instances that were solved without invoking the SAT solver. From the table, it can be seen that the strength of *ExWCo* lies in being able to solve the instance without the need of the solver. If the solver is needed to find a solution, the schemes have a similar success rate. However, as *ExWCo* produces longer and non-conflicting paths, it tends to keep more vertices in the subgraph. This may be sometimes harder for the solver to deal with, as can be seen in Table 1, size 32×32, *room*. On the other hand, as the size increases, *ExWCo* is able to solve all of the instances in the preprocess phase, while the other schemes are unable to do so. If the preprocess phase does not produce a solution, the underlying solver is also unable to solve it in the given timeout as those instances are simply too big (around hundred agents).

Conclusion

We improved the subgraph method for reduction-based MAPF solvers by selecting better initial paths for each agent. We showed that when optimizing makespan, many instances are solvable optimally in the preprocessing phase. In future work, we aim to explore the subgraph method for the sum of costs objective function.

[2]https://github.com/husarma/diplomka_kissat

Acknowledgments

This research was supported by the Czech Science Foundation Grant No. 23-05104S, by the Czech-Israeli Cooperative Scientific Research Project LUAIZ24104, by SVV project number 260 821, and by Charles University project UNCE 24/SCI/008.

References

Achá, R. A.; López, R.; Hagedorn, S.; and Baier, J. 2021. A New Boolean Encoding for MAPF and its Performance with ASP and MaxSAT Solvers. In Ma, H.; and Serina, I., eds., *Proceedings of the Fourteenth International Symposium on Combinatorial Search (SOCS'21)*, 11–19. AAAI Press.

Barták, R.; and Švancara, J. 2019. On SAT-Based Approaches for Multi-Agent Path Finding with the Sum-of-Costs Objective. In (Surynek and Yeoh 2019), 10–17.

Biere, A.; Fazekas, K.; Fleury, M.; and Heisinger, M. 2020. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling Entering the SAT Competition 2020. In Balyo, T.; Froleyks, N.; Heule, M.; Iser, M.; Järvisalo, M.; and Suda, M., eds., *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, 51–53. University of Helsinki.

Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds. 2009. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Tolpin, D.; Betzalel, O.; and Shimony, S. 2015. ICBS: Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding. In Yang, Q.; and Wooldridge, M., eds., *Proceedings of the Twenty-fourth International Joint Conference on Artificial Intelligence (IJCAI'15)*, 740–746. AAAI Press.

Dresner, K.; and Stone, P. 2008. A Multiagent Approach to Autonomous Intersection Management. *Journal of Artificial Intelligence Research*, 31: 591–656.

Gange, G.; Harabor, D.; and Stuckey, P. 2019. Lazy CBS: Implicit Conflict-Based Search Using Lazy Clause Generation. In Benton, J.; Lipovetzky, N.; Onaindia, E.; Smith, D.; and Srivastava, S., eds., *Proceedings of the Twenty-ninth International Conference on Automated Planning and Scheduling (ICAPS'19)*, 155–162. AAAI Press.

Garey, M.; and Johnson, D. 1983. Crossing Number is NP-Complete. *SIAM Journal on Algebraic Discrete Methods*, 4(3): 312–316.

Husár, M.; Švancara, J.; Obermeier, P.; Barták, R.; and Schaub, T. 2022. Reduction-based Solving of Multi-agent Pathfinding on Large Maps Using Graph Pruning. In Faliszewski, P.; Mascardi, V.; Pelachaud, C.; and Taylor, M., eds., *Proceedings of the Twenty-first International Conference on Autonomous Agents and Multiagent Systems (AAMAS'22)*, 624–632. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS).

Morris, R.; Pasareanu, C.; Luckow, K.; Malik, W.; Ma, H.; Kumar, T.; and Koenig, S. 2016. Planning, Scheduling and Monitoring for Airport Surface Operations. In *The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence: Planning for Hybrid Systems*, 608–614.

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

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2011. The Increasing Cost Tree Search for Optimal Multi-Agent Pathfinding. In Walsh, T., ed., *Proceedings of the Twenty-second International Joint Conference on Artificial Intelligence (IJCAI'11)*, 662–667. IJCAI/AAAI Press.

Silver, D. 2005. Cooperative Pathfinding. In Young, R.; and Laird, J., eds., *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE'05)*, 117–122. AAAI Press.

Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T.; Barták, R.; and Boyarski, E. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In (Surynek and Yeoh 2019), 151–159.

Surynek, P. 2015. On the Complexity of Optimal Parallel Cooperative Path-Finding. *Fundamenta Informaticae*, 137(4): 517–548.

Surynek, P. 2022. Problem Compilation for Multi-Agent Path Finding: a Survey. In Raedt, L., ed., *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI'22)*, 5615–5622. ijcai.org.

Surynek, P.; and Yeoh, W., eds. 2019. *Proceedings of the Twelfth International Symposium on Combinatorial Search (SOCS'19)*. AAAI Press.

Švancara, J.; Atzmon, D.; Strauch, K.; Kaminski, R.; and Schaub, T. 2024. Which Objective Function is Solved Faster in Multi-Agent Pathfinding? It Depends. In Rocha, A.; Steels, L.; and Jaap van den Herik, H., eds., *Proceedings of the Sixteenth International Conference on Agents and Artificial Intelligence (ICAART'24)*, 23–33. SciTePress.

Svancara, J.; Obermeier, P.; Husár, M.; Barták, R.; and Schaub, T. 2023. Multi-Agent Pathfinding on Large Maps Using Graph Pruning: This Way or That Way? In Rocha, A.; Steels, L.; and van den Herik, H., eds., *Proceedings of the Fifteenth International Conference on Agents and Artificial Intelligence (ICAART'23)*, 199–206. SciTePress.

Wurman, P.; D'Andrea, R.; and Mountz, M. 2008. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine*, 29(1): 9–20.

Yu, J.; and LaValle, S. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In desJardins, M.; and Littman, M., eds., *Proceedings of the Twenty-seventh National Conference on Artificial Intelligence (AAAI'13)*, 1443–1449. AAAI Press.