Experimental Evaluation of Classical Multi Agent Path Finding Algorithms

Omri Kaduri,[1] **Eli Boyarski,**[1] **Roni Stern**[1,2]
[1]Ben Gurion University of the Negev, SISE Dept., Be'er Sheva, Israel
[2]Palo Alto Research Center (PARC), ISL, Palo Alto, USA
kaduro@post.bgu.ac.il, eli@boyar.ski, sternron@post.bgu.ac.il

Abstract

Modern optimal multi-agent path finding (MAPF) algorithms can scale to solve problems with hundreds of agents. To facilitate comparison between these algorithms, a benchmark of MAPF problems was recently proposed. We report a comprehensive evaluation of a diverse set of state-of-the-art optimal MAPF algorithms over the entire benchmark. The results show that in terms of *coverage*, the recently proposed Lazy CBS algorithm outperforms all others significantly, but it is usually not the fastest algorithm. This suggests algorithm selection methods can be beneficial. Then, we characterize different setups for algorithm selection in MAPF, and evaluate simple baselines for each setup. Finally, we propose an extension of the existing MAPF benchmark in the form of different ways to distribute the agents' source and target locations.

1 Introduction

A *multi-agent pathfinding (MAPF)* problem with k agents is defined by a tuple $\langle G, s, t\rangle$, where $G = (V, E)$ is an undirected graph, $s : [1, \ldots, k] \rightarrow V$ maps an agent to its source vertex, and $t : [1, \ldots, k] \rightarrow V$ maps an agent to its target vertex. Each agent starts in its source vertex. Time is discretized and in every time step, each agent either *waits* in its current vertex or *moves* to one of the vertices adjacent to it. A solution to a MAPF problem is a sequence of wait/move actions for each agent such that the agents reach their targets without colliding with each other. MAPF has important applications in robotics (Veloso et al. 2015), autonomous vehicles, and automated warehouses (Wurman, D'Andrea, and Mountz 2008) and other fields (Morris et al. 2016; Ma et al. 2017). This problem of finding an optimal solution to a given MAPF problem, for various common optimization criteria, is known to be NP Hard (Surynek 2010; Yu and LaValle 2013). Nevertheless, many powerful optimal MAPF algorithms have been proposed (Felner et al. 2017; Ma and Koenig 2017), which in some cases can scale to problems with more than 100 agents. However, no algorithm has emerged to dominate all others.

Recently, a benchmark of MAPF problems in grid environments have been proposed (Stern et al. 2019), which includes a diverse set of 33 grids. The **first contribution** of this work is a comprehensive evaluation of a diverse set of state-of-the-art optimal MAPF algorithms, namely, EPEA* (Goldenberg et al. 2014), ICTS (Sharon et al. 2013), CBS-H (Boyarski et al. 2015), MDD-SAT (Surynek et al. 2016), and Lazy CBS (Gange, Harabor, and Stuckey 2019), over the entire benchmark. Running this evaluation required more than 45 days. The results of this evaluation show that the newest algorithm — Lazy CBS— is able to solve more problems under a specified time limit (5 minutes in our case), while CBS-H is more often the fastest algorithm. In general, the identity of the fastest algorithm varies between problems from different grid types and even for problems on the same grid. This suggests that using an *Algorithm Selection* (AS) method to find the best optimal MAPF algorithm for a given MAPF problem, can be beneficial.

Our **second contribution** is to distinguish between three AS for MAPF setups, called *in-grid*, *in-grid-type*, and *between-grids* AS. Prior work on AS for optimal MAPF (Kaduri, Boyarski, and Stern 2020) applied learning-based techniques to in-grid AS, and only included search-based solvers. We show that simple baseline approaches that do not require machine learning work well for all AS setups, when applied to the current MAPF benchmark.

The **third contribution** of this work is an extension to the current MAPF benchmark in the form of additional types of scenarios in which the agents source and target locations are distributed in different ways. The specific distributions we created have been proposed by Sigurdson et al. (2019) and can be seen in Figure 2. Finally, we report the results of all the algorithms in our portfolio as well as the AS baselines on the extended MAPF benchmark. The results show that the source and target locations' distributions can have significant affect on the performance of MAPF algorithm, suggesting interesting directions for future work. Our source code and dataset are publicly available.[1]

2 Data Collection

Next, we describe our data collection process, which consists of running a portfolio of MAPF algorithms on a suite

[1]https://github.com/OmriKaduri/mapf-selection

Model	Fastest			Coverage		
	All	Even	Rand.	All	Even	Rand.
EPEA*	9.63	8.49	10.66	55.26	55.13	55.57
ICTS	4.23	2.73	5.21	49.60	48.99	50.32
CBS-H	**46.95**	**45.02**	**48.75**	83.20	66.01	**98.49**
MDD-SAT	2.11	2.48	1.72	57.92	57.48	58.54
Lazy CBS	37.26	41.28	33.66	**92.36**	**91.45**	93.17
Best-at-grid(C)	49.45	49.98	48.47	**97.62**	**97.43**	**98.55**
Best-at-grid(F)	**60.09**	**61.77**	**60.29**	92.46	91.53	97.80
Best-at-grid-type(C)	51.18	40.88	48.71	93.95	91.64	98.51
Best-at-grid-type(F)	57.47	58.00	58.11	92.57	89.81	98.48

Table 1: Results for all the algorithms in our portfolio and the AS baselines over the standard MAPF benchmarks.

of MAPF problems and measuring their performance. We consider classical MAPF problems (Stern et al. 2019), where each action – wait or move – takes one time step. A collision between single-agent plans occurs if there are any vertex, edge, or swapping conflicts between them, i.e., the agents cannot occupy the same vertex, the same edge, or swap locations, at the same time, respectively. When an agent reaches its target, it stays there and blocks other agents from passing through that vertex.[2] The *cost* of a solution to a MAPF problem is the number of move/wait actions all agents perform until all agents reach their target. This is known as the *sum-of-costs* objective. All the MAPF algorithms we consider are optimal, i.e., they return only lowest-cost solutions.

Our portfolio of algorithms comprises the following diverse set of MAPF algorithms: (1) Enhanced Partial Expansion A* (EPEA*) (Goldenberg et al. 2014); (2) Increasing Cost Tree Search (ICTS) (Sharon et al. 2013); (3) MDD-SAT (Surynek et al. 2016); (4) Lazy CBS (Gange, Harabor, and Stuckey 2019); and (5) a state-of-the-art implementation of the Conflict Based Search(CBS) algorithm (Sharon et al. 2015) that includes a recently proposed heuristic (Li et al. 2019), conflict bypassing, and conflict prioritization (Boyarski et al. 2015). We refer to the latter as CBS-H.

EPEA* is a an A* variant designed for domains with a large branching factor that was shown to be effective for MAPF. ICTS and CBS-H also apply graph search algorithms, but they search in different search spaces. MDD-SAT works by compiling MAPF problems to sequences of Boolean Satisfiability (SAT) problems and solving them with an off-the-shelf SAT solver. Lazy CBS works by integrating techniques from CBS and Constraint Programming.

We run each of the algorithms in our portfolio on a suite of MAPF problems from the publicly available grid-based MAPF benchmark (Stern et al. 2019). This benchmark contains 33 grids arranged into seven types: grids from video games (denoted as **game** grids), city maps (**city**), maze-like grids (**maze**), grids arranged as rooms with narrow doors between them (**room**), open grids (**empty**), open grids with randomly placed obstacles (**random**), and grids that are inspired by the structure of warehouses (**warehouse**).

For each grid, the benchmark contains two sets of *scenario* files, each consisting of 25 files. A scenario file contains source and target locations for up to 1,000 agents, where possible. In the first set of scenario files, denoted *Random*, the agents source and target locations are located purely randomly. In the second set of scenario files, denoted *Even*, the agents' source and target locations are evenly distributed in buckets of 10 agents according to their distance. This creates MAPF problems with an even mix of short and long single-agent plans. We use these scenario files as suggested by Stern et al. (2019), that is, we use each algorithm to solve MAPF problems on the chosen grid with one agent, two agents, and so on until the runtime required to solve the problem reaches a timeout of 5 minutes. Problems that no algorithm in our portfolio could solve under this time limit were discarded from the analysis. We recorded the runtime of every algorithm in every run, and whether the algorithm has reached a timeout or not. Since all algorithms guarantee optimality, we do not report solution costs. The resulting dataset consists of over 190,000 solved MAPF problems, which includes more than 100,000 problems from *Random* scenario files, and 89,000 problems from *Even* scenario files. Creating this dataset required approximately 45 days of computation on a single computer. Our experiments are conducted on a Linux machine with 2.60GHz Intel Xeon Core E5-2630 with 256GB RAM.

In our analysis, we focus on the following metrics to aggregate the performance of a given MAPF algorithm on a set of problems. The first metric is **Fastest**, which is the percentage of problems the chosen algorithm solved the fastest. The second metric is **Coverage**, which is the percentage of problems the algorithm solved within the 5 minute timeout out of the total number problems solved within this timeout by all algorithms. To avoid biases in favor of larger grids, we normalized the above metrics for each grid, and report the mean of these values.

3 Analysis

The first five rows in Table 1 show the Fastest and Coverage results for all the evaluated algorithms across all the problems in our dataset. The columns "Even", "Rand.", and "All" show the results for the Even scenario files, Random scenario files, or both, respectively. Highlighted in bold are the results of the best algorithm in each metric and set of scenario files. Consider first the results over all the problems in our dataset (the data in the "All" columns). Lazy CBS outperforms all other algorithms by a large margin in terms of Coverage. Moreover, in our dataset less than 9% of the problems were solved by other algorithms and not by Lazy CBS. Thus, one may say that among the algorithms in our portfolio there is, finally, a universal winner: Lazy CBS. However, in terms of the Fastest metric, the results are more diverse. While CBS-H is most often the fastest algorithm (46.95 Fastest value), in the majority of the cases some other algorithm was the fastest. For example, in 37.26% of the cases Lazy CBS was the fastest, and for 9.63% of the problems EPEA* is the fastest. The latter result is surprising, since EPEA* has been developed almost 10 years ago.

[2]An agent can reach its target and then move away from it to allow another agent to pass. The agent will have to return to its target later, since eventually all agents must end up in their target.

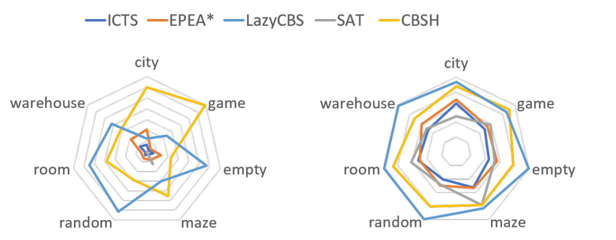

Figure 1: Fastest (left) and Coverage (right) per grid type.

Next, consider the difference between the results for the Even and Random scenarios. While most algorithms exhibit similar results for both types of scenario files, the performance of CBS-H is drastically different. For Even scenarios, its coverage is only 66.01, while for Random scenarios its coverage is 98.49, which is the highest coverage among all algorithms. A possible explanation for this is that Random scenarios are less likely to produce MAPF problems where the source and target of a given agent are very close to each other. Such cases may pose a challenge to CBS-H, since conflicts for such agents are harder to resolve, and moving in and out of an agent's target is known to be challenging for CBS. In contrast, Lazy CBS employs constraint propagation and conflict learning techniques that allows it to better handle problems with many conflicts.

3.1 Results per Grid Types

Figure 1 shows the Fastest (left) and Coverage (right) results for the different MAPF algorithms across different grid types for all the scenarios. In terms of Coverage, Lazy CBS and CBS-H perform similarly for game, city, and maze grids, but Lazy CBS achieves significantly higher Coverage for warehouse, room, random, and empty. Considering also the Fastest results, we see that CBS-H is almost always the fastest in the city and game grids, while Lazy CBS is usually the fastest for warehouse, room, random, and empty. Here too, it is worthwhile to note the surprising behavior of EPEA*, which is the fastest algorithm in a non-negiblibe portion of the problems in city and warehouse grids. On the other hand, EPEA* on the other grid types, as well as MDD-SAT and ICTS on all grid types, are not competitive in terms of their Coverage and Fastest results.

3.2 A Universal Winner for Optimal MAPF?

One may conclude that Lazy CBS is the best optimal MAPF algorithm in our portfolio: it has significantly higher overall Coverage and is often the fastest algorithm. This is not, however, the conclusion we draw from our analysis.

First, the literature on optimal MAPF is growing rapidly, and therefore we were not able to include all state of the art improvements and algorithms for MAPF. In particular, our best CBS implementation, CBS-H, does not include recently proposed symmetry-breaking techniques (Zhang et al. 2020). In addition, our portfolio does not include BCP (Lam et al. 2019), which reports some improvements over Lazy CBS. Second, a deeper analysis of our dataset reveals that in some grids, Lazy CBS was significantly outperformed by all other algorithms. This is the case in the *orz900d* game grid. In this grid, Lazy CBS performs poorly across all 50 different scenario files. Specifically, while Lazy CBS cannot solve for MAPF problems in this grid with more than 5 agents on average, other solvers solve for problems with almost 100 agents. Similarly, in the *w_woundedcoast* game grid, Lazy CBS was outperformed in 49 out of 50 scenario files. These grids are the largest grids in the benchmark, but we leave further analysis for future work.

Another conclusion that one may draw from our results is that MDD-SAT and ICTS perform poorly and should be discarded. This conclusion is also premature. In some cases, MDD-SAT significantly outperforms all other algorithms. For example, in the *maze-32-32-4* grid there is a scenario in which MDD-SAT is able to solve for 22 agents while the next best algorithm is able to only solve for 5 agents. On average, in this specific maze grid MDD-SAT is able to solve problems with 6.83 more agents than the next best algorithm. Note that this grid is particularly challenging, and on average, the maximal number of agents solved by any algorithm for a given scenario file is only 26.58 for this grid.

4 Algorithm Selection for MAPF

The above observations and results suggest that choosing intelligently which optimal MAPF algorithm to choose for a given MAPF problem is a worthwhile endeavor. Recently Kaduri et al. (Kaduri, Boyarski, and Stern 2020) applied AS techniques to achieve this, using supervised learning to train a classifier that chooses the *best* optimal MAPF algorithm for a given MAPF problem, where *best* here means maximizing one of the metrics defined above. While they achieved remarkable results, their algorithm portfolio only included search-based MAPF algorithm. Specifically, they did not consider using MDD-SAT or Lazy CBS. Also, in the setup they considered all the grids were provided during training. That is, they evaluated the performance of their AS algorithms on MAPF problems created on the same grids as those they trained on, only with different agent configurations. This setup is reasonable in some cases, but it cannot be used if one receives a MAPF problem on grids that were not available during training. In general, research on AS for MAPF must state what is the relation between the MAPF problems available during training and the MAPF problems used for evaluation purposes, i.e., testing.

4.1 MAPF Algorithm Selection Setups

To support future research on this topic, we define the following AS for MAPF setups: (1) **In-grid AS**, where all grids used for testing are also provided during training, yet with different agent configurations (i.e., source and target locations); (2) **In-grid-type AS**, where all grid types used for testing are also provided during training, yet the specific grids used for testing are different; and (3) **Between-grid-type AS**, where grid types used for testing are different from those used for training.

For a given AS setup (in-grid, in-grid-type, and between-grid-type) and evaluation metric (Coverage or Fastest), there

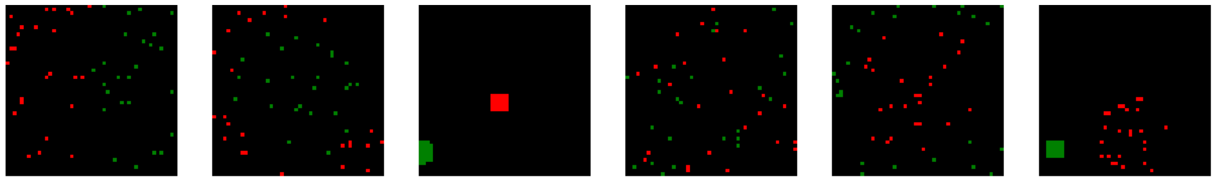

Figure 2: Example of Sigurdson et al.'s (2019) agent distributions. From left to right: CS, IO, TT, SS, OI, and TW.

is a natural AS baseline that maximizes the chosen metric on the relevant problems in the training set. For the in-grid setup, we denote by *Best-at-grid(C)* and *Best-at-grid(F)* the baselines that maximizes the Coverage and Fastest metrics, respectively. For example, given a MAPF problem Π, the best-at-grid(C) baseline selects the algorithm that yielded the highest average Coverage in the training problems that shared the same grid as Π. Similarly, we denote by *Best-at-grid-type(C)* and *Best-at-grid-type(F)* the corresponding baselines for the in-grid-type setup. Next, we report on the performance of the four AS baselines defined above. Note that a perfect AS aimed towards optimizing the Fastest metric would always have perfect score of 100% for both metrics (C and F).

4.2 Algorithm Selection Baselines Results

To evaluate these AS baselines, we performed a 3-fold cross validation on our dataset. The last 4 lines in Table 1 show the average Coverage and Fastest results for our baselines.

The results show that all baselines often fail to identify the fastest algorithm. For example, the best-at-grid(F) baseline identifies the fastest algorithm in 60.09% of the problems. Thus, an AS method may result in identifying the fastest algorithm in significantly more case. However, if one aims to maximize Coverage, the results show that there is little to gain from developing algorithm selection methods for all algorithm selection setups. For example, best-at-grid(C) already yields Coverage higher than 97%. This means a perfect AS algorithm for the in-grid setup, i.e., one that always chooses the best algorithm for each problem, will be able to solve less than 3% more problems from this dataset over the best-at-grid baseline. Similarly, a perfect AS for the in-grid-type setup can solve less than 9% more problems from this dataset over the best-at-grid-type baseline. The baseline for maximizing Coverage in the between-grid setup is to always choose Lazy CBS, since it has the highest Coverage. The baseline also achieves a very high Coverage (over 91%).

5 Extending the MAPF Benchmark

The distinction between Even and Random scenario files is a step towards understanding the impact of agent locations distributions on the performance of different MAPF algorithms. As shown in Table 1, agent locations distributions has already a significant impact on the performance of CBS-H, where it provided the highest coverage Coverage for Random scenarios (98.49) but a much lower Coverage for Even scenarios (66.01). To this end, we propose to extend the grid-based MAPF benchmark (Stern et al. 2019) with additional types of scenario files that include more diverse agent locations' distributions. Specifically, the new types of scenario files we created follows the agent locations' distributions proposed by Sigurdson et al. (2019): (1) **Cross Sides (CS)**: all agents start on one side and traverse to the other side; (2) **Swap Sides (SS)**: half the agents start on one side and the other half start on the other side, targets are randomly selected in the side opposite of their sources; (3) **Inside Out (IO)**: agents start near the center of the grid and are assigned targets near the outer edges of the grid; (4) **Outside In (OI)**: agents start near the outer edges of the grid and are assigned targets near the center of the grid; (5) **Tight to Tight (TT)**: agents start together and are assigned targets that are as close as possible elsewhere on the grid; (6) **Tight to Wide (TW)**: agents start close together and are assigned targets that are spread out in the same general area of the grid. Table 2 shows examples of the different scenario types. Green dots mark the agents' sources and red dots their targets.

We run all our algorithms on these newly created scenario files. The results are shown in Table 2. Here too Lazy CBS performs well in terms of Coverage and CBS-H performs better according to the Fastest metric. However, the relative performance of the different algorithm vary significantly w.r.t. the type of scenarios. For example, the Coverage of Lazy CBS is 90.51 for IO scenarios while it drops to 83.81 for TT scenarios. Also, it is the fastest algorithm in 42.83% of the problems from CS scenarios and only 21.35% for problems from TW scenarios. Similarly, the Coverage of EPEA* and CBS-H are 74.89 and 74.53, respectively, for TW scenarios, and it drops to 56.23 and 51.58, respectively, CS scenarios. The results also show that here, there is larger potential for improvements using AS methods. For example, the best-at-grid(C) achieves Coverage between 89.76 and 94.48 while it achieved an almost perfect Coverage (98.55) in the original dataset.

6 Discussion and Conclusion

To the best of our knowledge, we performed the first comprehensive evaluation of search-based and compilation-based algorithms for optimal MAPF over the grid-based MAPF benchmark.

The results of this evaluation reveal that: (1) Lazy CBS significantly outperform all other algorithms in terms of Coverage; (2) CBS-H is more often the fastest algorithm to solve a given a problem; but (3) in many cases other algorithms are faster. Then, we introduce three types of AS setups, and describe baselines for each. Surprisingly, these baselines work remarkably well on the current benchmark, leaving little room for advanced AS, in terms of coverage. Therefore, we generated an extension of the current

Model	Fastest							Coverage						
	All	CS	SS	IO	OI	TT	TW	All	CS	SS	IO	OI	TT	TW
EPEA*	13.50	12.00	10.20	14.38	11.55	7.92	19.08	59.67	56.23	57.90	58.26	53.38	63.10	74.89
ICTS	4.89	4.22	4.47	3.23	5.55	4.48	7.08	55.39	54.17	53.04	52.89	50.32	62.49	68.40
CBS-H	**42.47**	38.73	**42.58**	**41.90**	**41.05**	**65.17**	**50.64**	55.93	51.58	55.07	53.46	54.11	72.04	74.53
MDD-SAT	2.22	2.22	4.17	3.06	3.65	2.79	1.84	56.02	55.63	58.29	55.90	57.66	57.94	70.62
Lazy CBS	36.95	**42.83**	38.58	37.43	38.20	19.64	21.35	**89.74**	**89.59**	**89.42**	**90.51**	**89.09**	**83.81**	**88.36**
Best-at-grid(C)	44.97	45.38	43.66	46.33	44.76	27.13	32.19	**93.41**	**91.10**	89.76	**94.48**	91.40	**92.36**	**90.14**
Best-at-grid(F)	**52.91**	52.77	51.63	51.58	46.20	44.86	49.99	81.66	85.49	85.00	81.75	82.15	79.53	84.88
Best-at-grid-type(C)	39.43	44.63	39.45	34.89	37.61	52.44	28.36	88.8	88.32	**88.98**	87.58	**90.43**	87.95	87.61
Best-at-grid-type(F)	52.61	**55.86**	**54.52**	**51.78**	**48.72**	**59.82**	**51.84**	78.22	79.05	79.54	77.86	77.03	80.32	86.54

Table 2: Results for all the algorithms in our portfolio and the AS baselines over the extended MAPF benchmarks.

benchmark where the agents source and target locations are distributed in various ways, as proposed by Sigurdson et al. (Sigurdson et al. 2019). Future work will add the results of other state of the art optimal MAPF algorithms to our datasets, namely BCP (Lam et al. 2019) and symmetry breaking techniques for CBS (Zhang et al. 2020).

References

Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Tolpin, D.; Betzalel, O.; and Shimony, E. 2015. ICBS: improved conflict-based search algorithm for multi-agent pathfinding. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Felner, A.; Stern, R.; Shimony, S. E.; Boyarski, E.; Goldenberg, M.; Sharon, G.; Sturtevant, N. R.; Wagner, G.; and Surynek, P. 2017. Search-Based Optimal Solvers for the Multi-Agent Pathfinding Problem: Summary and Challenges. In *Symposium on Combinatorial Search (SoCS)*, 29–37.

Gange, G.; Harabor, D.; and Stuckey, P. J. 2019. Lazy CBS: Implicit conflict-based search using lazy clause generation. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 155–162.

Goldenberg, M.; Felner, A.; Stern, R.; Sharon, G.; Sturtevant, N.; Holte, R. C.; and Schaeffer, J. 2014. Enhanced partial expansion A. *Journal of Artificial Intelligence Research* 50: 141–187.

Kaduri, O.; Boyarski, E.; and Stern, R. 2020. Algorithm Selection for Optimal Multi-Agent Pathfinding. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 161–165.

Lam, E.; Le Bodic, P.; Harabor, D.; and Stuckey, P. J. 2019. Branch-and-cut-and-price for multi-agent pathfinding. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1289–1296.

Li, J.; Felner, A.; Boyarski, E.; Ma, H.; and Koenig, S. 2019. Improved heuristics for multi-agent path finding with conflict-based search. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 442–449.

Ma, H.; and Koenig, S. 2017. AI buzzwords explained: multi-agent path finding (MAPF). *AI Matters* 3(3): 15–19.

Ma, H.; Yang, J.; Cohen, L.; Kumar, T. K. S.; and Koenig, S. 2017. Feasibility Study: Moving Non-Homogeneous Teams in Congested Video Game Environments. In *Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 270–272.

Morris, R.; Pasareanu, C. S.; Luckow, K. S.; Malik, W.; Ma, H.; Kumar, T. S.; and Koenig, S. 2016. Planning, Scheduling and Monitoring for Airport Surface Operations. In *AAAI Workshop: Planning for Hybrid Systems*.

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

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2013. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence* 195: 470–495.

Sigurdson, D.; Bulitko, V.; Koenig, S.; Hernández, C.; and Yeoh, W. 2019. Automatic Algorithm Selection In Multi-agent Pathfinding. *CoRR* URL http://arxiv.org/abs/1906.03992.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Barták, R.; and Boyarski, E. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *the International Symposium on Combinatorial Search (SOCS)*, 151–159.

Surynek, P. 2010. An Optimization Variant of Multi-Robot Path Planning Is Intractable. In *AAAI*.

Surynek, P.; Felner, A.; Stern, R.; and Boyarski, E. 2016. Efficient SAT approach to multi-agent path finding under the sum of costs objective. In *European Conference on Artificial Intelligence (ECAI)*, 810–818.

Veloso, M. M.; Biswas, J.; Coltin, B.; and Rosenthal, S. 2015. CoBots: Robust Symbiotic Autonomous Mobile Service Robots. In *IJCAI*, 4423.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2008. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine* 29(1): 9.

Yu, J.; and LaValle, S. M. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *AAAI*.

Zhang, H.; Li, J.; Surynek, P.; Koenig, S.; and Kumar, T. S. 2020. Multi-agent path finding with mutex propagation. In *International Conference on Automated Planning and Scheduling*, volume 30, 323–332.