Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)

Algorithm Selection for Optimal Multi-Agent Pathfinding

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), SSL, Palo Alto, USA
kaduro@post.bgu.ac.il, eli@boyar.ski, roni.stern@gmail.com

Abstract

The challenge of finding an optimal solution to a multi-agent path finding (MAPF) problem has attracted significant academic and industrial interest in recent years. While the problem is NP-Hard, modern optimal MAPF algorithms can scale to solve problems with hundreds of agents. Nevertheless, no single optimal MAPF algorithm dominates all benchmarks problems, and there are no clear, provable, guidelines for when each algorithm should be used. To address this, we present the first successful Algorithm Selection (AS) model for optimal MAPF. We propose two approaches for learning an AS model. The first approach uses a standard supervised learning algorithm with a set of handcrafted MAPF-specific features. The second approach, casts a MAPF problem to an image and applies a deep Convolutional Neural Network to classify it. We evaluate both approaches over a large dataset and show that using an AS model to select which algorithm to use for each instance results in solving more problems and in a shorter runtime compared to the state of the art.

1 Introduction

A *classical multi-agent pathfinding (MAPF)* problem with k agent (Stern et al. 2019) 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. In every time, an agent either *waits* in its current vertex or *moves* to one of the vertices adjacent to it. A solution to a classical MAPF problem is a sequence of wait/move actions for each agent such that the agents reach their targets without colliding with each other. In this work, we focus on the problem of finding an *optimal* solution to a given MAPF problem. This problem is known to be NP Hard for various common optimization criteria (Surynek 2010; Yu and LaValle 2013). Nevertheless, Classical MAPF and its many extensions have important applications in robotics, autonomous vehicles, and automated warehouses, and hence, many optimal MAPF algorithms have been proposed (Felner et al. 2017; Ma and Koenig 2017). Nevertheless, no algorithm has emerged to dominate all others. In fact, very little is known for how to choose which optimal MAPF algorithm to use for a given classical MAPF problem.

To close this gap, we propose a data-driven approach to learn an *Algorithm Selection* (AS) model for optimal MAPF. An AS model accepts a portfolio of optimal MAPF algorithms $\mathcal{A}$ and a MAPF problem Π, and outputs an algorithm in $\mathcal{A}$ for solving Π. Our main objective is to create an AS model that chooses an algorithm that solves Π in minimal runtime. We explore two approaches for creating such a model. In the first approach, we use XGBoost (2016), a state of the art tree-based supervised learning algorithm, with a set of handcrafted MAPF-specific features. In the second approach, we follow the work of Sigurdson et al. (Sigurdson et al. 2019) and cast the given MAPF problem to an image, and then train a deep Convolution Neural Networks (CNN) to select the appropriate optimal MAPF algorithm. We evaluate the AS models created by both approaches over a dataset with 39,000 samples across 28 grid types. The results show that the best AS model selects the best algorithm in over 65% of the cases, and using the selected MAPF algorithm significantly outperforms all the state of the art optimal MAPF algorithms in our portfolio. For example, using the best AS model we were able to solve 93% of all problems in our benchmark under 5 minutes, while the best MAPF algorithm solved only 82%.

Our portfolio contains only search-based MAPF algorithms, i.e., it does not include MAPF algorithms that are based on compilation to Boolean Satisfiability (SAT) (Surynek et al. 2016; Barták and Svancara 2019), Constraints Programming (Barták et al. 2017), and SMT (Surynek 2019; Erdem et al. 2013). Such algorithms are known to be highly effective in some domains, while less so in others. However, our approach can be easily applied to include other MAPF algorithms in the portfolio.

To the best of our knowledge, the only prior work on AS for MAPF is by Sigurdson et al. (2019). They focused on finding any solution to a given MAPF problem while we aim to find optimal solutions, and thus our portfolio consists only optimal MAPF algorithm. Also, we explore a range of approaches to learn an AS model. To allow future researchers to reproduce our results, we made our source code and dataset

publicly available at github.com/OmriKaduri/MAPF-Classification/tree/1.0/classification/src.

2 Background

In this work, we consider classical MAPF problems in a graph that represents a 4-neighborhood grid (Stern et al. 2019). Time is discretized and 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.[1] 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. An optimal MAPF algorithm is an algorithm that is guaranteed to return a lowest-cost solution.

2.1 Algorithms in the Portfolio

In this work, we used the following diverse set of optimal search-based MAPF algorithms: (1) A* (Hart, Nilsson, and Raphael 1968) with the Operator Decomposition (OD) and Independence Detection (ID) MAPF-specific enhancements (Standley 2010), (2) Enhanced Partial Expansion A* (EPEA*), which is an A* variant designed for state spaces with a large branching factor that was shown to be effective for MAPF (Goldenberg et al. 2014), (3) Increasing Cost Tree Search (ICTS) (Sharon et al. 2013), (4) Conflict-Based Search (CBS) (Sharon et al. 2015), (5) MA-CBS, which improves CBS by merging agents to a meta-agent when needed, and (6) a state-of-the-art variant of CBS that includes a recently proposed heuristic (Li et al. 2019) and conflict bypassing and prioritization (Boyarski et al. 2015). The latter will be referred to in this paper as CBS-H. For an overview of search-based optimal MAPF algorithms including these algorithms see Felner et al. (2017).

2.2 Algorithm Selection (AS)

Algorithm Selection (AS) is the problem of selecting the best algorithm from a given set of algorithms on a per-instance basis (Rice 1976). AS has been successfully applied to a range of optimization and satisfaction problems (Kotthoff 2016; Kerschke et al. 2019). A notable example is SATZilla (Xu et al. 2012), which is a highly successful algorithm for generating an AS model for solving SAT problems.

Contemporary methods for generating effective AS models are based on supervised machine learning. Such methods can be split into two major approaches. The first approach is based on *effort estimation*. That is, the data in the training set is used to learn a regression model for each algorithm $A \in \mathcal{A}$ that aims to predict the runtime it will take A to solve a given problem. The corresponding AS model chooses the algorithm predicted to require the least amount of runtime to solve the given problem. A different approach to learn an AS model is to directly train a multi-class classifier to predict who is the best algorithm for a given problem. To differentiate between these two approaches, we refer to the former as the *regression approach* and the latter as the *classification approach*.

[1] 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: MAPF problem represented as an image.

Both approaches are known to have limitations. The regression approach is inherently more difficult, since the space of mistakes is larger (all possible runtimes), and predicting runtimes is considered a noisy target for regression models. The classification approach outputs a single algorithm for a given MAPF problem, so it cannot be sensitive to cases where two algorithms perform similarly. There are more sophisticated approaches that aim to mitigate these limitations. For example, Xu et al. (2012) proposed to mitigate the limitation of the classification approach by learning a cost-sensitive binary classification model for every pair of solvers (Xu et al. 2012). To select an algorithm for a given problem Π, the solvers are ranked according to the number of times each of them has been chosen by the classifiers for Π, and the highest-ranking solver is returned. We experimented with this approach in our domain and did not observe any improvement. Since this approach is also significantly more costly in terms of training time, we focus in the rest of this paper on the more fundamental regression and classification approaches.

3 Learning AS Models for Optimal MAPF

The first learning approach we explore uses a state-of-the-art tree-based learning algorithm, namely XGBoost (Chen and Guestrin 2016), with a set of handcrafted features specifically designed for MAPF. A key factor in the effectiveness of XGBoost, and in general supervised learning algorithms, is the set of features extracted from every instance. We created the following sets of MAPF-specific features.

3.1 Handcrafted MAPF Features

The first set of features describes the grid. It includes (1) the number of rows and columns in the grid (denoted GridRows and GridColumns), (2) the number of blocked cells in the grid (denoted NumOfObstacles), and (3) the ratio of blocked cells (denoted ObstacleDensity), i.e., $\frac{NumOfObstacles}{GridRows \cdot GridCols}$. The next set of features is derived from the number of agents (k) and its relation to the grid size. This set includes (1) the number of agents (denoted NumOfAgents), (2) the

Model	Accuracy	Coverage	Total RT
A* +OD+ID	0.00	0.73	12,459
EPEA*	0.12	0.79	9,871
ICTS	0.10	0.78	10,570
CBS	0.00	0.59	17,447
MA-CBS	0.26	0.53	19,952
CBS-H	0.50	0.83	8,016
Random	0.16	0.71	13,215
CNN Rg.	0.51	0.88	6,210
XGBoost Rg.	0.51	0.82	8,410
CNN Cl.	0.55	0.91	5,122
XGBoost Cl.	0.66	0.93	4,307
Oracle	1.00	100.00	1,649

Figure 2: Results for all models across all the test set.

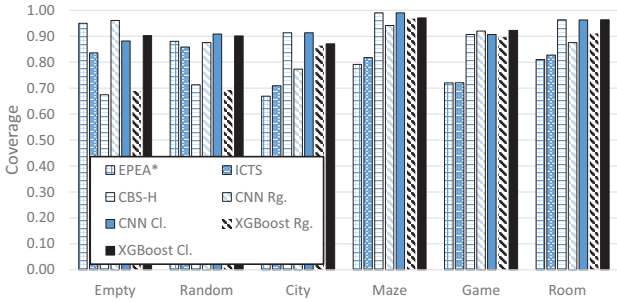

Figure 3: Coverage results, split by grid types.

branching factor of a naive A* search (denoted BranchingFactor), i.e., 5^k, and (3) the number of agents divided by the number of cells that are not blocked by an obstacles in the grid (denoted AgentSparsity). The last set of features is derived from the shortest distances between the agents' sources and targets. This set includes the average, max., and min. distance between agents' sources and targets, i.e., $\sum_{i=1}^{k} \delta\big(s(i), t(i)\big)/k$, $\max_{i=\{1\ldots k\}} \delta\big(s(i), t(i)\big)$, and $\min_{i=\{1\ldots k\}} \delta\big(s(i), t(i)\big)$, respectively, where $\delta(v, v')$ is the length of the shortest path from v to v'. We call these features Avg/Max/MinDistanceToGoal, respectively. The AvgStartDistances feature is the average distance between the agents' source vertices, i.e.,

$$\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} 2\delta\big(s(i), s(j)\big)/\big(k \cdot (k-1)\big) \quad (1)$$

The AvgGoalDistances feature is defined similarly with respect to the average distance between the agents' target vertices. The CellsAtSPRatio feature is the ratio S/T where S is the number of grid cells that are on a shortest path from the source to the target of at least one agents, and T is the total number of grid cells that are not blocked by an obstacle.

We used the above sets of features and created two AS models with XGBoost (Chen and Guestrin 2016), one following the classification approach for AS and the other following the regression approach. We call the resulting AS models *XGBoost Classification* (XGBoost Cl.) and *XGBoost Regression* (XGBoost Rg.), respectively.

3.2 Deep Learning MAPF Images

The second learning approach we explore maps a given MAPF problem to an image and then uses a Convolutional Neural Network (CNN) to classify it. Specifically, we follow Sigurdon et al.'s (Sigurdson et al. 2019) recent work on AS for non-optimal MAPF, and use an image to represent a MAPF problem, as follows. Blocked and unblocked cells are represented by white and black pixels, respectively. Source and target vertices are represented by green and red pixels, respectively. We resize the resulting image such that it fits into the network input layer, which in our case was set to 224x224x3.[2] An example of such an image is Figure 1.

There are many CNN architectures for image classification. We used the VGG-16 architecture (Simonyan and Zisserman 2014), which is a modern CNN architecture that is commonly used in the image recognition literature. In addition, to improve runtime and model accuracy, we made the following adjustments to this architecture. First, we added a global average pooling (GAP) layer (Lin, Chen, and Yan 2014) after the last convolution layer. After the GAP layer, we added a fully connected layer with 64 neurons to gradually reduce dimensionality towards the output layer, which contained 6 neurons, one per algorithm in our portfolio.

Using the MAPF image and the CNN architecture described above, we created two AS models: one following the classification approach and the other following the regression approach. We call the resulting AS models *CNN Classification* (CNN Cl.) and *CNN Regression* (CNN Rg.). We note that for CNN Rg., we trained a single CNN so that it accepts a MAPF problem and every output neuron is associated with one of the MAPF algorithms in our portfolio and should output the runtime for the corresponding algorithm on the given MAPF problem. To do so, we use the multi-output regression learning (Borchani et al. 2015), which is known to work well with neural networks.

3.3 Dataset

To train and evaluate the AS models we proposed, we used the publicly available grid-based MAPF benchmark, detailed in (Stern et al. 2019). This benchmark contains 28 grids including 10 grids from popular video games, 3 city maps (Berlin, Boston, Paris), 4 maze-like grids, 3 grids that arranged as rooms with narrow doors between them, 4 open grids, and 4 open grids with randomly placed obstacles.

This benchmark also includes 25 *scenario* files for each grid. A scenario file contains source and target locations for up to 1,000 agents (where possible). We use this scenario file 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. We recorded the runtime of every algorithm in every run, and whether the algorithm has reached a timeout or not. The resulting dataset consists over 39,000

[2]The last dimension indicates the color of the pixel

instances. Since scenarios are solved incrementally (creating new data by adding more agents to the same problem) we ensured that data from the same scenario is never used for both training and testing. Specifically, we used data from 30% of the scenarios for testing and data from the remaining scenarios for training.

Our dataset poses a challenge to the regression approach for AS: how to consider the runtime of an algorithm that failed to solve a problem? This type of data is called *censored data*, and methods to deal with it have been studied, e.g., for survival analysis. Standard methods include removing the censored data, treating it as uncensored (i.e., setting the timeout time as the runtime), as well as a sophisticated iterative method (Schmee and Hahn 1979)). Treating such data as uncensored yielded the highest coverage (see definition of coverage below) in our experiments.

4 Experimental Results

We used the following metrics to evaluate the proposed AS models: accuracy, coverage, and total runtime. *Accuracy* is the ratio of problems that the AS model correctly selected the fastest algorithm in the portfolio for this problem. *Coverage* is the ratio of problems solved within the 5 minute timeout using the algorithm selected by the evaluated model. *Total runtime* (Total RT) is the overall runtime, in minutes, it took to solve all problems in our test set, choosing for every problem the algorithm selected for it by the evaluated model.

4.1 Accuracy, Coverage, and Total RT Results

Table 2 shows the accuracy, coverage, and total RT for all evaluated algorithms across all scenarios in our test set. For reference, the upper portion of the table shows the results of choosing the same algorithm for all problems. Thus, accuracy in this context is the ratio of problems in which that algorithm solved first. *Oracle* and *Random* are also given for comparison purposes: Oracle always chooses the correct MAPF algorithm and Random chooses one randomly. The accuracy of Oracle is 1 and that of Random is one over the number of algorithms in our portfolio.

Table 2 shows that most AS models yield better results than every individual algorithm in our portfolio across all parameters. For example, the coverage and total RT of CBS-H is 0.83 and 8,016, respectively, while both CNN Cl. and XGBoost Cl. had a coverage higher than 0.9 and a total RT lower than 5,122. Comparing the different AS models, we see that classification approach yields better results compared to the regression approach. For example, the coverage of XGBoost Rg. is 0.82 while it is 0.93 for XGBoost Cl. Comparing XGBoost and CNN Cl. models shows that while XGBoost is significantly more accurate, it only has a modest advantage in terms of coverage and total RT. We conjecture that the advantage of XGBoost over the CNN is due to the fact that the image we used to represent a MAPF problem for the CNN loses valuable information such as the distinction between the source and target of different agents.

Figure 3 shows coverage (y-axis) of the best models from each family of algorithms for different grid types. The results highlight the importance of AS in optimal MAPF,

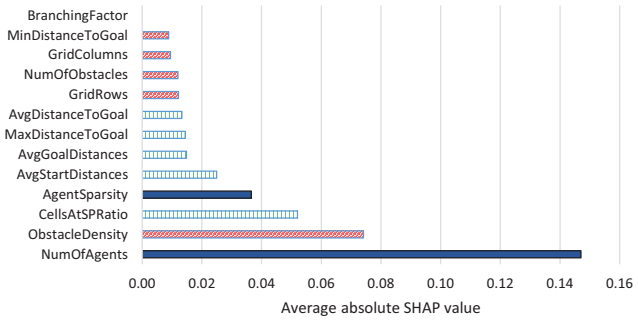

Figure 4: Feature importance calculated using SHAP values.

showing that different algorithms excel in different types of grids. For example, consider the performance of EPEA* and CBS-H. CBS-H performs very well in City, Maze, Game, and Room grids, while EPEA* performs poorly there. However, in Empty and Random grids CBS-H performs poorly and EPEA* excels. The performance of XGBoost Cl. and CNN Cl. is very similar in all grid types, with coverage that is either the highest or very close to it, thus demonstrating the robustness of our AS models.

4.2 Feature Importance

Next, we explore the impact of our handcrafted features on the performance of XGBoost Cl., which had the overall highest coverage and accuracy. To this end, we used *TreeExplainer*, which is a publicly-available tool for analyzing tree-based models. TreeExplainer relies on computing SHapley Additive exPlanation (SHAP) values (Lundberg and Lee 2017). The SHAP value of a pair of feature and sample (in our case, MAPF problem) is the average of the marginal contributions across all permutations of the given feature to the label given to the sample by the model. The intuitive meaning of a high absolute SHAP value is that the feature had a significant impact on why the sample was given its label.

Figure 4 shows for every feature the mean of the absolute SHAP values over all samples and labels. As can be seen, the three most important features – NumOfAgents, ObstacleDensity, and PointsAtSPRatio – include a feature from each of the three feature families described in Section 3.1. The key feature, as expected, is the number of agents. Of less importance are features that describe the grid itself, e.g., GridRows, NumOfObstacles, and GridColumns. Since BranchingFactor directly correlates with NumOfAgents, its importance is negligible.

5 Discussion and Conclusion

To the best of our knowledge, this is the first research on Algorithm Selection for optimal MAPF. Two high-level approaches for Algorithm Selection – regression and classification – as well as two types of learning algorithms – XGBoost with handcrafted features and CNN with image-based features – were explored. We implemented and evaluated these Algorithm Selection approaches on a portfolio of 6 state-of-the-art search-based optimal MAPF algorithms and a comprehensive, publicly available MAPF benchmark that

includes 28 grids, resulting in a dataset of 39,000 instances. Our analysis shows that indeed Algorithm Selection can be successfully applied to optimal MAPF, yielding an optimal MAPF solver that outperforms all the optimal MAPF algorithms in its portfolio. The most immediate direction of future work is to include additional non-search-based solvers such as MDD-SAT (Surynek et al. 2016) and BCP (Lam et al. 2019) in our algorithm portfolio, as well as adding the map-specific metrics suggested by Sturtevant (Sturtevant 2012). Future work can also combine feature extraction using deep learning with the manually extracted features. Furthermore, in order to address more general MAPF problems (i.e., not only classical MAPF), graph embedding techniques may be superior to regular CNNs.

6 Acknowledgments

This research was supported by ISF grant #210/17 to Roni Stern.

References

Barták, R., and Svancara, J. 2019. On sat-based approaches for multi-agent path finding with the sum-of-costs objective. In *the International Symposium on Combinatorial Search (SOCS)*, 10–17.

Barták, R.; Zhou, N.; Stern, R.; Boyarski, E.; and Surynek, P. 2017. Modeling and solving the multi-agent pathfinding problem in picat. In *IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 959–966.

Borchani, H.; Varando, G.; Bielza, C.; and Larrañaga, P. 2015. A survey on multi-output regression. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 5(5):216–233.

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)*.

Chen, T., and Guestrin, C. 2016. Xgboost: A scalable tree boosting system. In *International Conference on Knowledge Discovery and Data mining (KDD)*, 785–794.

Erdem, E.; Kisa, D. G.; Oztok, U.; and Schüller, P. 2013. A general formal framework for pathfinding problems with multiple agents. In *AAAI*.

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.

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.

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.

Kerschke, P.; Hoos, H. H.; Neumann, F.; and Trautmann, H. 2019. Automated algorithm selection: Survey and perspectives. *Evolutionary computation* 27(1):3–45.

Kotthoff, L. 2016. Algorithm selection for combinatorial search problems: A survey. In *Data Mining and Constraint Programming*. Springer. 149–190.

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-19)*, 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.

Lin, M.; Chen, Q.; and Yan, S. 2014. Network in network. In *International Conference on Learning Representations (ICLR)*.

Lundberg, S. M., and Lee, S.-I. 2017. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, 4765–4774.

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

Rice, J. R. 1976. The algorithm selection problem. In *Advances in computers*, volume 15. Elsevier. 65–118.

Schmee, J., and Hahn, G. J. 1979. A simple method for regression analysis with censored data. *Technometrics* 21(4):417–432.

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.

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

Sigurdson, D.; Bulitko, V.; Koenig, S.; Hernández, C.; and Yeoh, W. 2019. Automatic algorithm selection in multi-agent pathfinding. *CoRR*.

Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Standley, T. S. 2010. Finding optimal solutions to cooperative pathfinding problems. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

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.

Sturtevant, N. R. 2012. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games* 4(2):144–148.

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.

Surynek, P. 2010. An optimization variant of multi-robot path planning is intractable. In *AAAI*.

Surynek, P. 2019. Unifying search-based and compilation-based approaches to multi-agent path finding through satisfiability modulo theories. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1177–1183.

Xu, L.; Hutter, F.; Shen, J.; Hoos, H. H.; and Leyton-Brown, K. 2012. SATzilla2012: Improved algorithm selection based on cost-sensitive classification models. *Proceedings of SAT Challenge* 57–58.

Yu, J., and LaValle, S. M. 2013. Structure and intractability of optimal multi-robot path planning on graphs. In *AAAI*.