MAPFASTER: A Faster and Simpler take on Multi-Agent Path Finding Algorithm Selection

Jean-Marc Alkazzi[1,2,*], Anthony Rizk[1,3], Michel Salomon[2] and Abdallah Makhoul[2]

***Abstract*— Portfolio-based algorithm selection can help in choosing the best suited algorithm for a given task while leveraging the complementary strengths of the candidates. Solving the Multi-Agent Path Finding (MAPF) problem optimally has been proven to be NP-Hard. Furthermore, no single optimal algorithm has been shown to have the fastest runtime for all MAPF problem instances, and there are no proven approaches for when to use each algorithm. To address these challenges, we develop MAPFASTER, a smaller and more accurate deep learning based architecture aiming to be deployed in fleet management systems to select the fastest MAPF solver in a multi-robot setting. MAPF problem instances are encoded as images and passed to the model for classification into one of the portfolio's candidates. We evaluate our model against state-of-the-art Optimal-MAPF-Algorithm selectors, showing $+5.42\%$ improvement in accuracy while being $7.1\times$ faster to train. The dataset, code and analysis used in this research can be found at https://github.com/jeanmarcalkazzi/mapfaster.**

I. Introduction

Multi-Agent Path Finding (MAPF) is the task of finding collision-free paths that route agents from their start to goal locations in a known environment. This problem of planning paths is a task that needs to be solved in all domains where multiple mobile agents are involved. Practically, MAPF has many fields of application such as video games, traffic control, and robotics. For instance, agents can be mobile robots in a warehouse [1], Unmanned Aerial Vehicles (UAVs) for delivery services [2], characters in a game [3], [4], [5], [6], or even piping systems to be routed correctly in a constrained area [7]. One can easily understand the underlying difficulty of the MAPF problem due to the real-time coordination of agents that is required. Their locations will change constantly over time and the paths will have to be continuously updated in order to still reach the final destination while avoiding collisions with each other and with the obstacles encountered.

MAPF is NP-Hard to solve optimally, given that it is a general case of the 15-pieces puzzle problem which was proven to be NP-Hard [8]. Researchers are trying to move from benchmark suites [9] to integrating real-world constraints like robot kinematics [10], [11], [12], [13], and potential real-world disturbances of unforeseen delays due to slower robot execution or unpredictable path blockage [14], [15], [16]. Integrating real-world constraints is especially important for robot fleet management systems where adaptability to new situations and performance stability across MAPF problems are key to a successful system. Although a lot of research has been put into new MAPF approaches, there is no one fastest optimal algorithm for any given MAPF problem instance. Adequate algorithm selection based on the MAPF problem instance is therefore a necessity for a successful real-world system with varying constraints and needs.

Algorithm Selection is a meta-algorithmic technique for choosing the most suitable algorithm per input instance. Each problem has specific metrics to define the best performing solver (e.g. solving speed, accuracy, or a custom metric). What makes Algorithm Selection helpful is its ability to leverage the complementary strengths of the algorithms in the portfolio. It is therefore crucial that candidate algorithms behave differently on the input instances and, if chosen optimally, will resolve all input instances.

Using Deep Learning approaches to select the algorithm with fastest runtime for an input instance encoded as an image was originally tackled by Sigurdson *et al.* [18] focusing on single-agent path finding. It was later shown that using a similar approach to [18] while encoding start and end goals of each agent in the map image and using AlexNet model [19] for prediction can achieve SOTA performance on the benchmark dataset [20]. Kaduri *et al.* [21] later

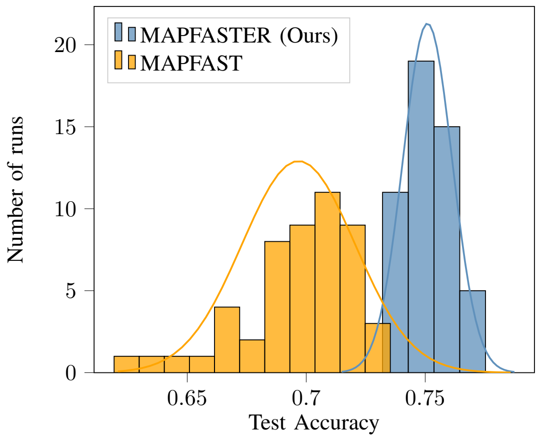

Fig. 1: Monte Carlo Cross-Validation Evaluation [17] with $k = 50$ runs comparing previous SOTA model's accuracy and stability with our proposed method.

[1]Jean-Marc Alkazzi and Anthony Rizk are with IDEALworks GmbH, Munich, Germany {jean-marc.alkazzi, anthony.rizk}@idealworks.com

[2]Jean-Marc Alkazzi, Michel Salomon, and Abdallah Makhoul are with FEMTO-ST Institute, UMR 6174 CNRS, Univ. Bourgogne Franche-Comté, Belfort, France {abdallah.makhoul,michel.salomon}@femto-st.fr

[3]Anthony Rizk is with Faculty of Engineering, Saint Joseph University of Beirut, Campus des sciences et technologies, Mar Roukos Dekwaneh, B.P. 1514 - Riad El Solh, Beirut 1107 2050, Lebanon {anthony.rizk1}@net.usj.edu.lb

*Corresponding author

demonstrated that manually engineered features extracted per instance and passed to an XGBoost [22] model could beat previous deep-learning based approaches in terms of choosing the fastest optimal solver per input instance. Ren *et al.* [23] further improved on previous work by adding single-agent shortest path embedding to the input images denoting their importance in increasing the accuracy from 67.10% up to 71.18% using a modified Inceptionv3 [24] body and a classification head. Introducing auxiliary tasks while training helped the model learn more relevant features from the input instance and was proposed as their final MAPFAST [23] architecture reaching a reported 76.89% of accuracy.

To tackle the MAPF Algorithm Selection for Fleet Management Systems, this paper proposes (1) MAPFASTER, a smaller and more accurate Deep Learning based architecture for MAPF Algorithm Selection, (2) An analysis of current benchmark dataset with potential shortcomings and improvements, (3) A new evaluation approach for more consistent benchmark results. MAPFASTER achieves a $+5.42\%$ accuracy improvement over previous SOTA models and a $7.1\times$ speedup of training due to its smaller architecture. Evaluating using the Monte Carlo Cross-Validation approach [17] helps in gaining insights on the stability of a proposed model for confident deployments in multi-robot production environments. It shows a maximum of 4.32% accuracy deviation between the worst and best runs for MAPFASTER compared to 11.6% for previous SOTA [23] showing significant improvement in model stability as can be noticed in Figure 1. We further outline our failed experiments to save researchers time and money, and end with proposed future work for the task of MAPF Algorithm Selection.

The remaining of this paper is organized as follows. Section II presents the benchmark specifications. In Section III we present our proposed model MAPFASTER. The evaluation and experimental results are provided in Section IV. An analysis section is presented in Section V. Section VI concludes the paper with proposed future work directions.

II. The Benchmark

The dataset used consists of MAPF problem instances encoded as images as the model's input. For each instance, the fastest algorithm to solve it is considered as its label. The model's task is therefore to predict which of the portfolio's algorithms would be the fastest solver for the given problem.

A. Dataset

The dataset used is based on the MAPF benchmarks [9]. For each of the 33 different maps, several scenarios are found with different number of agents and length of agent paths. A scenario defines a MAPF problem, outlining the missions per robots with their start and goal locations. For MAPF Algorithm Selection, 24,967 input instances were created, depicting a wide combination of different maps, number of agents, and paths' length. This dataset was originally created by Ren *et al.* [23] and used as the raw data in this paper.

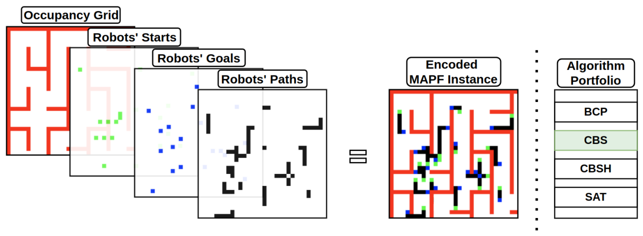

Fig. 2: Input images created by overlaying the occupancy grid, robots' start and goal locations, and their shortest path. Fastest solver in the portfolio is used as the dataset label, here being CBS.

1) Input Encoding: The MAPF Scenario instances are encoded as follows:

1) The original occupancy grid map representing the static obstacles is rendered as a white and red image, where white represents the free space and red represents the occupied space.
2) Agents' start locations are drawn on the original map as green dots while their goal locations are drawn as blue dots.
3) For each agent, its shortest path from start to goal location is calculated and drawn on top of the map in black. Shortest paths focus on each robot alone without considering potential collisions with other robots.

Figure 2 shows the different layers involved and the final input image representing a MAPF Scenario instance. This input will be used by the model to choose the fastest Optimal Solver from the Portfolio, in this example, CBS [25].

2) Data Pre-Processing: In the original work, the images were resized to 320×320 as the model's input. By analyzing the resized images, we found a squishing artifact was caused to several maps of different aspect ratio as depicted in Figure 4. Such a squishing effect would result in a loss of information, namely disappearing robots' start or goal location, or shortest paths being cut-off. To overcome this, we modify the resizing approach by using the Albumentations package [26] and keeping the aspect ratio while padding the images with black borders when necessary.

B. Algorithm Portfolio

Portfolio-based algorithm selection leverages the complementary strengths of the potential candidates. This suggests that the portfolio should be diverse and most importantly, the algorithms' strengths should be complimentary for an extended coverage of the input problems. In this work, we use the same algorithm portfolio proposed by Ren *et al.* [23] for a fair comparison of performance. The proposed portfolio contains **Search-Based** algorithms: CBS [25], CBSH [27], an **Optimization-Based** algorithm: BCP [28], and a **Satisfiability-Based** algorithm: SAT [29]. All instances in the dataset are solvable by at least one of the algorithms in the portfolio. This is a strong indication that the algorithms are complementary and together, cover the entirety of the input problems. A more in-depth look at the portfolio choice

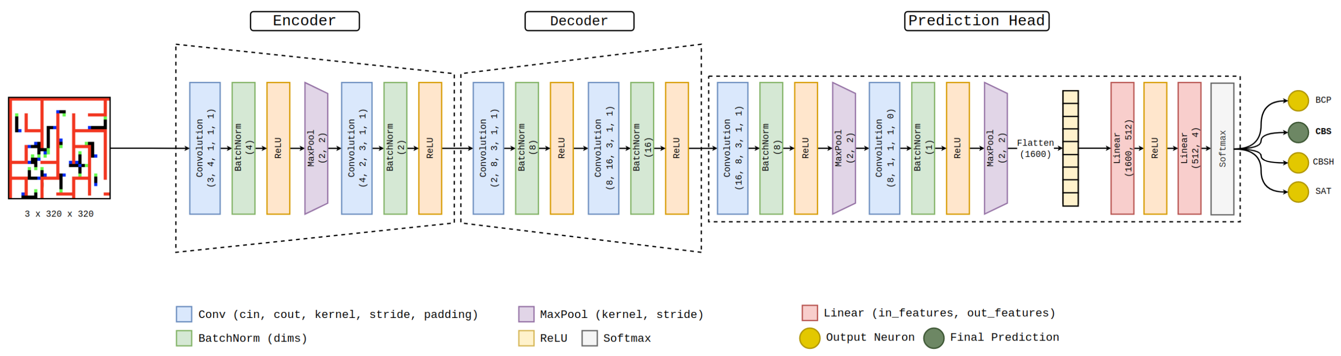

Fig. 3: MAPFASTER Model Architecture

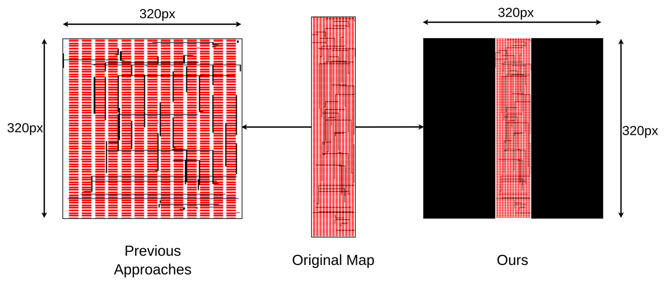

Fig. 4: Map resizing has squishing artifacts in the original dataset, which we solve by padding the resized image to keep the original aspect ratio.

was outlined by the original authors of the benchmark dataset [23].

III. MAPFASTER

A. *Model Architecture*

We cast the problem of algorithm selection as a multi-class classification problem where the goal of the model is to output the probability for each of the portfolio's algorithms of being the fastest on the given input instance.

As shown in Figure 3, our model architecture has 3 main modules: the encoder, the decoder, and the prediction head.

The encoder module is trained to compress the input instances to a lower dimensional representation, called the hidden state. The decoder uses a similar architecture as the encoder, but the decoder is trained to augment the compressed representation with additional features, in this case, through additional channels. The bottleneck created between the encoder and decoder modules forces the model to learn and keep the features which are most relevant to the algorithm selection task. The prediction head then uses this representation and outputs the probability for each algorithm of the portfolio for being the fastest one for the given input instance. We can see that the components involved in the architecture are basic layers without including any tricks of residual connections, attention mechanism, or any other advanced features. The goal behind this is to find the simplest architecture that can be easily trained and maintained while solving the MAPF problems with higher accuracy and stability. This is aimed at providing the research community with a baseline model which is fast to train, shows stable performance, and can be easily compared with other models given the provided code in our repository. The goal was not to find the smallest possible architecture to solve the problem, but a small enough one to showcase that architecture size did not play a role in the model's accuracy given the benchmark dataset.

B. *Model Training*

We initialize our model using Kaiming initialization [30] for the Convolutional layers' weights, $\gamma = 1$ and $\beta = 0$ for BatchNorm layers, a normal distribution with parameters $\mu = 0$ and $\sigma = 0.01$ for the Linear layers' weights, and a constant of 0 for all biases. The goal of the Kaiming initialization is to ensure that the variance of the activations are the same across every layer. Having this constant variance helps prevent exploding or vanishing gradients [30]. We use Cross Entropy Loss as the model's loss function and Adam [31] as the optimizer with a learning rate of 0.004. We train for 7 epochs with a batch size fixed at 64.

IV. EVALUATION

A. *Experimental Setup*

Our experiments were conducted on an NVIDIA RTX 2080 GPU. The proposed network was implemented in PyTorch v1.10.0 [32] with integration of Albumentations [26] for data pre-processing, and fastai [33] for model training and analysis. Training and inference were conducted on a single GPU to ensure fair comparison with single GPU training and inference of previous SOTA models [23]. We use wandb [34] to log the dataset used, as well as each training run and the final model weights to ensure correct reporting of results and reproducibility of our experiments.

B. *Metrics*

To assess the performance of our model and compare to previous approaches, we use accuracy, coverage, total runtime, and the custom score calculation proposed in [23]. An algorithm selector is deemed accurate when it chooses the fastest algorithm from the portfolio to solve the input problem instance. Coverage is defined as the percentage of problem instances where the algorithm selector did not

TABLE I: Reported Metrics per model over 50 runs with different random seeds following the Monte Carlo Cross-Validation [17] approach.

	Accuracy (%)				Coverage (%)				Score				Runtime (minutes)			
Model	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**
$MAPFAST_{cl}$	61.93	73.53	69.68	0.024	90.28	95.26	93.65	0.009	763.27	844.075	813.16	17.9	1480	2065	1675	122
$MAPFAST$	58.35	74.06	69.39	0.030	90.52	95.86	93.95	0.012	748.35	855.5	815.42	21.6	1390	1999	1636	149
MAPFASTER (Ours)	**73.21**	**77.53**	**75.1**	**0.010**	**94.03**	**96.84**	**95.63**	**0.005**	**837.76**	**879.85**	**856.85**	**9.15**	**1233**	**1627**	**1439**	**79**

necessarily choose the fastest algorithm, but chose one which would solve it in less than 5 minutes. Runtime is defined as the total time needed by the selected algorithms to solve input instances. A lower runtime shows that the algorithm selector chooses faster algorithms to solve the MAPF problems. In many cases, multiple algorithms can solve the same problem with very similar runtimes. Choosing any of the algorithms would be a valid choice for fleet management systems. Accuracy alone is therefore not the best indicator of real-world performance for an algorithm selector, and a higher coverage would not rule out a selector always choosing the slowest algorithm. Ren *et al.* [23] proposed a custom score to reward algorithm selectors based on the relative performance of their chosen optimal solvers. For instance, two selectors choosing different algorithms with similar runtimes will get a similar reward instead of only rewarding the one with a slightly faster algorithm selected.

C. Model Evaluation

To evaluate the model's performance, we use an *80:10:10* split ratio for training, validation, and test data respectively. For a fair evaluation of the performance and stability of both MAPFAST and MAPFASTER, we evaluate both models using repeated hold-out validation, also called Monte Carlo Cross-Validation [17]. The Monte Carlo Cross-Validation approach consists of splitting the data k times using k different random seeds, and reporting the average accuracy across the k runs to avoid misinterpreting a model's performance caused by a specific data split.

In our evaluation, we use the Monte Carlo Cross-Validation approach with $k = 50$ runs. After close inspection of the original MAPFAST code[35], we found that the provided data split function was causing a data leak. Although the seed was fixed and the data split was carefully designed to be deterministic for the same seed, a Python-related bug caused the original list of data samples to be different on each run, hence causing a difference in the data split. This, in turn, invalidated the evaluation of the model which, on each run, was evaluated on a new random split of the original dataset, potentially containing samples from the training data. To ensure fair and correct evaluation of MAPFAST, we dockerized the provided code with their proposed Python requirements and only modified the data split function to ensure that the data split is deterministic for the same seed. We further present the fixed MAPFAST code as well as scripts to replicate the original data leak as part of our open-source code. To prevent such issues from arising in our codebase, we implement several unit tests to ensure the integrity of the dataset used and deterministic splits per chosen random seed.

In Table I, we report the min, max, $mean$, and std of each metric previously proposed for MAPFAST, MAPFAST$_{cl}$, and MAPFASTER. As reported in [23], using the classification head only, MAPFAST$_{cl}$ achieves an accuracy of 71.18% whereas the final model with auxiliary tasks achieves an accuracy of 76.89%. After fixing the data leak and training again, we found the real accuracy to be on average 69.39% for MAPFAST and 69.68% for MAPFAST$_{cl}$. Nevertheless, MAPFAST was still the state-of-the-art at time of publication, beating XGBoost Cl. [21] by +2.57% in accuracy.

Looking at Table I, we can see that reporting a 58.35% accuracy for MAPFAST would be unfair, and reporting 77.53% accuracy for MAPFASTER would be an overestimation of +2.43% over the actual performance of our model of 75.1%. This shows the importance of the Monte Carlo Cross-Validation evaluation approach in the fair reporting of metrics. We also realize that adding the auxiliary tasks to MAPFAST$_{cl}$ does not improve the performance as reported in [23], but introduces a slightly higher instability of the model.

MAPFASTER outperforms the previous state-of-the-art approach across all metrics with +5.42% Accuracy, +1.68% Coverage, +41.43 points in Score, and a decrease of 12.04% in total Runtime. Our model is also smaller, with 0.8M trainable parameters vs 3.2M for MAPFAST, and can be trained in 5.28min compared to 37.5min for MAPFAST. This sheds the light on the actual complexity of the task and potential next steps to follow for the future of the field, which we tackle in the Analysis section of this paper.

V. Analysis

A. Model Performance

Model stability is a key factor to consider when working in production environments. The model's performance on real-world data must be as close as possible to its performance on the test data. This gives us more confidence that our test data is representative of the real-world data and that our model has stable performance characteristics. Our model shows a total deviation between min and max accuracy of 4.32% whereas MAPFAST shows a deviation of 11.6% depending on the randomness in data splits and model initialization. This shows the increased stability of our proposed approach and a more predictable range of performance as shown in

Figure 1. The fact that a model architecture with 4× less parameters was able to extract enough features to achieve a higher accuracy on 50 different data splits could indicate that additional model complexity may not be required to extract relevant features from the input images. Although we have not experimented with larger architectures, we believe they would achieve only slightly better accuracy on the reference dataset. However, we do not believe that such a model architecture would be sufficient to solve the problem and capture the complexity of the task in the real-world setting given the current shortcomings of the input instance encoding.

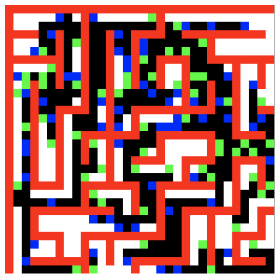

Fig. 5: Overcrowded Maze Map with 60 agents showing overlap of paths and potential difficulty for effective feature extraction.

B. The Benchmark Dataset

Algorithms like CBS [25] and CBSH [27] display different performance given the number of potential robot collisions, the map obstacle density, and the total number of robots among other factors. For instance, CBS performs on par with CBSH in high obstacle density maps with up to 40 robots, but fails to do so with 40 to 120 robots [27]. In more complex maps, BCP [28] shows a higher success rate than CBSH [27] whereas CBS [25] fails to solve the problem altogether [28]. This indicates that extracting such features from the input image may be helpful for an accurate algorithm selection. It may come as natural to carefully extract such features from the input instances and use them for algorithm selection, but that was shown by Kaduri *et al.* [21] to have a limited performance. We believe that the current benchmark dataset suffers from one main shortcoming, the lack of a time dimension. In a flat image, overlapping paths can hide potential conflicts, and overcrowded maps can be too noisy for relevant feature extraction as shown in Figure 5. An adequate representation of such features while keeping the data encoding process fast and memory efficient is important for deployments in fleet management systems and will be the focus of our future work.

C. Explored Paths

To save other researchers time, sanity, and cloud compute credits, we briefly outline the list of experiments we tried which did not improve the performance of our proposed model. This list, although neither extensive nor do we claim considerable evaluation of what was done per item, acts as an additional insight on potential future work and research paths to consider.

- **Adding SE Blocks**.
  Squeeze-and-Excitation Blocks [36] take the input data and multiply each channel by a learnable weight to help the model extract features from relevant parts of the input. Given that different aspects lie on different channels (red is obstacle, green is start, blue is goal), learning weights for each one seemed an intuitive improvement. However, implementing them did not improve the model's performance.
- **Using 1cycle training**.
  1cycle training, also known as super-convergence [37] leads to faster training times, and helps avoid overfitting. It did not help us with the small model proposed, but we still provide it in our open-source code for future researchers to try.
- **Coloring the robot paths differently**.
  Given the data ambiguity previously mentioned, we tried coloring each robot's shortest path with a different color to support better feature extraction. This did not help our model's performance and sometimes led to worse results. We provide the generated shortest paths for researchers to try out more tricks in the image encoding.
- **Including the originally proposed auxiliary tasks**.
  Ren *et al.* [23] proposed using two auxiliary tasks during training to improve the model's performance. One head would predict if each algorithm in the portfolio would solve the MAPF instance in less than 5min, the other head would output the pairwise comparison of the portfolio's algorithms predicted speed. This made the model bigger, training slower, and led to worse results overall.
- **Adding new auxiliary tasks**.
  We experimented with multiple different auxiliary tasks, including predicting the map obstacle density and the number of unique robots in the map. The intuition behind it comes from the analysis provided by [21], [23], [27] showing that the number of robots and the map obstacle density potentially affect which algorithm performs the fastest. This did not show considerable change in the model's performance, although we believe it may be useful for future research to revisit this point.

VI. Conclusion and Future Work

In this paper we have shown that our smaller model of 0.8M parameters, trainable in 5.28min, was able to outperform the previous SOTA model with 3.2M parameters, trainable in 37.5min. We also proposed using a more appropriate performance evaluation approach, the Monte Carlo cross-validation technique, to report true performance of the models regardless of the randomness involved in the dataset split and model initialization. Model stability is crucial for real-world deployments and our model showed a maximum deviation of 4.32% between worst and best training run, compared to 11.6% for previous SOTA approaches. Following a successful implementation of a considerably smaller model with higher accuracy and stability, we want to shed the light on the benchmark dataset as the focus of further research, compared to focusing on bigger or more complex models. We hope that the open-source code and analysis provided will help researchers explore different paths and gain new insights on the underlying complexity of the algorithm selection task in the muti-agent path finding problem setting.

Acknowledgment

This work has been supported by IDEALworks GmbH and the EIPHI Graduate School (contract "ANR-17-EURE-0002").

References

[1] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. K. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding in large-scale warehouses," in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '20. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2020, p. 1898–1900.

[2] F. Ho, R. Geraldes, A. Gonçalves, B. Rigault, B. Sportich, D. Kubo, M. Cavazza, and H. Prendinger, "Decentralized multi-agent path finding for UAV traffic management," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 2, pp. 997–1008, 2022.

[3] R. Graham, H. McCabe, and S. Sheridan, "Neural networks for real-time pathfinding in computer games," *The ITB Journal*, vol. 5, p. 21, 2004.

[4] M. Sinkar, M. Izhan, S. Nimkar, and S. Kurhade, "Multi-agent path finding using dynamic distributed deep learning model," *2021 International Conference on Communication information and Computing Technology (ICCICT)*, pp. 1–6, 2021.

[5] R. Bamal, "Collision-free path finding for dynamic gaming and real time robot navigation," *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 101–108, 2019.

[6] H. Ma, J. Yang, L. Cohen, T. K. S. Kumar, and S. Koenig, "Feasibility study: Moving non-homogeneous teams in congested video game environments," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2017.

[7] G. Belov, L. Cohen, M. G. de la Banda, D. Harabor, S. Koenig, and X. Wei, "Position paper: From multi-agent pathfinding to pipe routing," *ArXiv*, vol. abs/1905.08412, 2019.

[8] O. Goldreich, *Finding the Shortest Move-Sequence in the Graph-Generalized 15-Puzzle Is NP-Hard*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1–5.

[9] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, E. Boyarski, and R. Bartak, "Multi-agent pathfinding: Definitions, variants, and benchmarks," *The 12th Annual Symposium on Combinatorial Search (SoCS)*, pp. 151–158, 2019.

[10] W. Hönig, T. K. S. Kumar, L. Cohen, H. Ma, S. Koenig, and N. Ayanian, "Path planning with kinematic constraints for robot groups," *ArXiv*, vol. abs/1704.07538, 2017.

[11] L. Wen, Z. Zhang, Z. Chen, X. Zhao, and Y. Liu, "CL-MAPF: Multi-agent path finding for car-like robots with kinematic and spatiotemporal constraints," *ArXiv*, vol. abs/2011.00441, 2020.

[12] K. S. Yakovlev, A. Andreychuk, and V. Vorobyev, "Prioritized multi-agent path finding for differential drive robots," *2019 European Conference on Mobile Robots (ECMR)*, pp. 1–6, 2019.

[13] Z. A. Ali and K. S. Yakovlev, "Prioritized SIPP for multi-agent path finding with kinematic constraints," *ArXiv*, vol. abs/2108.05145, 2021.

[14] W. Hönig, S. Kiesel, A. Tinka, J. W. Durham, and N. Ayanian, "Persistent and robust execution of MAPF schedules in warehouses," *IEEE Robotics and Automation Letters*, vol. 4, pp. 1125–1131, 2019.

[15] D. Atzmon, A. Felner, R. Stern, G. Wagner, R. Barták, and N.-F. Zhou, "k-Robust multi-agent path finding," in *The 10th Annual Symposium on Combinatorial Search (SoCS)*, 2017.

[16] H. Ma, T. K. S. Kumar, and S. Koenig, "Multi-agent path finding with delay probabilities," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, 2017.

[17] Q. Xu and Y. Liang, "Monte Carlo cross validation," *Chemometrics and Intelligent Laboratory Systems*, vol. 56, pp. 1–11, 2001.

[18] D. Sigurdson and V. Bulitko, "Deep learning for real-time heuristic search algorithm selection," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2017.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, pp. 84 – 90, 2012.

[20] D. Sigurdson, V. Bulitko, S. Koenig, C. Hernandez, and W. Yeoh, "Automatic algorithm selection in multi-agent pathfinding," *ArXiv*, vol. abs/1906.03992, 2019.

[21] O. Kaduri, E. Boyarski, and R. Stern, "Algorithm selection for optimal multi-agent pathfinding," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, no. 1, pp. 161–165, 2020. [Online]. Available: https://ojs.aaai.org/index.php/ICAPS/article/view/6657

[22] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[23] J. Ren, V. Sathiyanarayanan, E. Ewing, B. Senbaslar, and N. Ayanian, "MAPFAST: A deep algorithm selector for multi agent path finding using shortest path embeddings," in *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '21. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2021, p. 1055–1063.

[24] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2016.308

[25] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370214001386

[26] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, "Albumentations: Fast and flexible image augmentations," *Information*, vol. 11, no. 2, 2020. [Online]. Available: https://www.mdpi.com/2078-2489/11/2/125

[27] A. Felner, J. Li, E. Boyarski, H. Ma, L. Cohen, T. K. S. Kumar, and S. Koenig, "Adding heuristics to conflict-based search for multi-agent path finding," in *ICAPS*, 2018.

[28] E. Lam, P. Le Bodic, D. D. Harabor, and P. J. Stuckey, "Branch-and-cut-and-price for multi-agent pathfinding," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 7 2019, pp. 1289–1296. [Online]. Available: https://doi.org/10.24963/ijcai.2019/179

[29] P. Surynek, A. Felner, R. Stern, and E. Boyarski, "Efficient SAT approach to multi-agent path finding under the sum of costs objective," in *ECAI*, 2016.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec 2015. [Online]. Available: http://dx.doi.org/10.1109/ICCV.2015.123

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *ArXiv*, vol. abs/1412.6980, 2017.

[32] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," *ArXiv*, vol. abs/1912.01703, 2019.

[33] J. Howard and S. Gugger, "Fastai: A layered api for deep learning," *Information*, vol. 11, no. 2, 2020. [Online]. Available: https://www.mdpi.com/2078-2489/11/2/108

[34] L. Biewald, "Experiment tracking with weights and biases," 2020, software available from wandb.com. [Online]. Available: https://www.wandb.com/

[35] R. Jingyao and S. Vikraman, "MAPFAST," *GitHub repository*, 2020.

[36] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-excitation networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, pp. 2011–2023, 2020.

[37] L. N. Smith and N. Topin, "Super-convergence: Very fast training of residual networks using large learning rates," *ArXiv*, vol. abs/1708.07120, 2017.