Learning to Select Nodes in Bounded Suboptimal Conflict-Based Search for Multi-Agent Path Finding

Taoan Huang, Bistra Dilkina, Sven Koenig
University of Southern California
{taoanhua,dilkina,skoenig}@usc.edu

Abstract

Multi-Agent Path Finding is an NP-hard problem that is difficult for current approaches to solve optimally. Research has shown that bounded suboptimal solvers, such as Enhanced Conflict-Based Search (ECBS), are more efficient than optimal solvers in finding a feasible solution with suboptimality guarantees. ECBS is a tree search algorithm that repeatedly selects nodes from a focal list to expand the tree. In this work, we propose to use imitation learning and curriculum learning to learn node-selection strategies for different grid maps and agent sizes. We then deploy the learned models in ECBS and test their solving performance on unseen instances drawn from the same distribution as the one used in training. Our approach shows substantial improvement over the baselines on different grid maps.

1 Introduction

Multi-Agent Path Finding (MAPF) is the problem of finding a set of conflict-free paths for a team of moving agents on a given graph that minimizes the sum of path costs or the makespan. It has practical applications in video games [18], distribution centers [17, 11] and traffic controls [7]. MAPF is NP-hard [25] and optimal solvers, such as Conflict-Based Search (CBS) [22], do not scale well. Research has shown that Enhanced Conflict-Based Search (ECBS) [1] and its variants [4, 5] can find bounded suboptimal solutions fast. Based on CBS, ECBS uses focal searches [19] instead of best-first searches to guarantee bounded suboptimality. The high-level search of ECBS maintains a so-called focal list that contains a subset of nodes on the open list whose costs are at most $w \geq 1$ times the current lower bound of the optimal cost. From the focal list, ECBS could select an arbitrary node to expand but the common practice is to select one with the minimum d-value, which is a heuristic value computed for each node when it is generated and typically, a hard-coded function in the algorithm.

Instead of manually defining d-values, we apply machine learning techniques and propose a novel data-driven framework for learning node-selection strategies for the high-level focal search. We use imitation learning [6, 20, 21] and curriculum learning [2] to learn ranking functions from solving instances that can differentiate nodes that are closer to a solution within our desired suboptimality bound from those nodes that are far away from one. During the search, the ranking function takes a node's features as input and produces a real value as its d-value. By using our d-values, we can get closer to a desired solution every time we expand a node and, therefore, find the solution more quickly. In experiment, we show that our approach can scale to large problem instances beyond those that are solvable by ECBS, the state-of-the-art bounded suboptimal MAPF algorithm.

2 MAPF and Related Work

Given an undirected unweighted graph $G = (V, E)$, the *Multi-Agent Path Finding (MAPF) problem* is to find a set of conflict-free paths for a set of agents $\{a_1, \ldots, a_k\}$. Each agent a_i has a start vertex $s_i \in V$ and a goal vertex $t_i \in V$. Time is discretized into time steps, and, at each time step, every

1st Workshop on Learning Meets Combinatorial Algorithms @ NeurIPS 2020, Vancouver, Canada.

agent can either move to an adjacent vertex or wait at its current vertex. A conflict occurs when two agents are at the same vertex at the same time or they traverse the same edge in opposite directions between two consecutive time steps. The cost of an agent is defined as the number of time steps until it reaches its goal and no longer moves. Given $w \geq 1$, our goal is to find a conflict-free solution with the sum of costs that is at most w times the minimum sum of agents' costs.

CBS is a complete and optimal bi-level tree search on a binary search tree called *constraint tree* (CT). On the high level, CBS starts with a root node with an empty set of constraints and expands the CT by always expanding a node N with the lowest cost N_{Cost}. After choosing N, CBS identifies the set of conflicts N_{Conf} in N. If there are no conflicts, CBS returns the solution. Otherwise, it picks one conflict to resolve and adds two child nodes of N to the CT by adding to the set of constraints at N a new constraint in one of two conflicting agents to one of the child nodes and the other conflicting agent in the other child node. Then, we apply the low-level search to each child to re-plan the optimal path for the affected agent, and record the respective solution and its cost.

ECBS is a bounded-suboptimal version of CBS [1] that is guaranteed to find a w-approximate solution for given $w \geq 1$. ECBS uses focal searches [19] instead of best-first searches in both levels. Consider a node N. On the low level, ECBS runs a focal search for each agent such that the cost of the path found is at most w times the cost of its individually cost-minimal path w.r.t. the constraints at N. Let $N_{\mathsf{LB},i}$ be the lower bound on agent a_i's cost given by the low-level search for a_i when node N is generated and let $N_{\mathsf{LB}} = \sum_{i=1}^{k} N_{\mathsf{LB},i}$. On the high level, ECBS performs a focal search that maintains a focal list that contains all nodes N in $\mathcal{N}$ such that $N_{\mathsf{Cost}} \leq w\mathsf{LB}$, where $\mathcal{N}$ is the open list and $\mathsf{LB} = \min_{N \in \mathcal{N}} N_{\mathsf{LB}}$. Once a solution is found by always expanding a node in the focal list, it is guaranteed to be a w-approximate solution.

Related Work Optimal MAPF solvers, such as mixed-integer programming(MIP)-based approaches [14, 15], and CBS [22] and its variants [3, 9, 16], have been developed. Our work is more related to research on bounded suboptimal solvers. ECBS [1] is the current state-of-the-art bounded suboptimal solver. Variants of ECBS [5, 4] have been proposed to speed up ECBS to solve MAPF in environments with certain structures such as warehouses. Machine learning techniques have been applied to speed up branch-and-bound tree search. [10] is a closely related work that uses imitation learning to learn node-selection and node-pruning strategies for solving MIP. [23] scales up [10] by progressively increasing the instance sizes in the form of curriculum learning. Other related work includes learning to branch [13] and run primal heuristics [12] in tree search for solving MIP.

3 Learning Node-Selection Strategies for ECBS

We introduce our framework for learning node-selection strategies for the high-level focal search of ECBS. Our framework consists of two phases, model learning and ML-guided search.

3.1 Model Learning

We focus on solving instances with fixed graph G and w. We first learn node-selection strategies to solve instances with different numbers of agents $\{k_1, \ldots, k_m\}$ where $k_1 < \ldots < k_m$. For each k_i, we learn a node-selection strategy that assigns $\pi_i(\phi(N))$ to node N as its d-value, where $\phi(N)$ is the feature vector of N and π_i is a learned ranking function. Therefore, a desired ranking function would be one that assigns smaller d-values for nodes that are closer to a solution. Our training algorithm is a curriculum learning algorithm, as shown in Algorithm 1, that takes $\{k_1, \ldots, k_m\}$ and m sets of training instances $\{\mathcal{I}_1, \ldots, \mathcal{I}_m\}$ as input and outputs $\{\pi_1, \ldots, \pi_m\}$. Each instance in $\mathcal{I}_i$ includes k_i agents and is i.i.d. sampled from a given distribution. Initially, π_0 is set to an initial strategy π^* (e.g., strategies used in previous work [1]) (line 2). To obtain π_1 for k_1 agents, we deploy DAgger($\pi^*, \mathcal{I}_i$) [21] that learns a ranking function from instances in $\mathcal{I}_i$ using π^* as a starting point. To obtain π_i ($i > 1$), instead of starting from π^* again, we start learning from π_{i-1}. We obtain π_i ($1 < i \leq m$) iteratively by calling DAgger($\pi_{i-1}, \mathcal{I}_i$) that learns a ranking function using π_{i-1} as the initial node-selection strategy (line 3-4) until a stopping criterion is met (line 5-7) or $i = m$. If the stopping criterion is met before i reaches m, we terminate training (line 7) and simply set π_j to π_i for all $i < j \leq m$ (line 6).

DAgger($\pi^{(0)}, \mathcal{I}$) is shown in Algorithm 2. The input $\pi^{(0)}$ and $\mathcal{I}$ are the initial ranking function and the set of training instances. Initially, the dataset D is set to $\emptyset$ (line 1). The algorithm runs for R

Algorithm 1 Training Algorithm: Curriculum Learning

1: **Input:** Numbers of agents $\{k_1, \ldots, k_m\}$ and training instances $\{\mathcal{I}_1, \ldots, \mathcal{I}_m\}$
2: $\pi_0 \leftarrow \pi^*$
3: **for** $i = 1$ to m **do**
4: $\quad \pi_i \leftarrow \text{DAgger}(\pi_{i-1}, \mathcal{I}_i)$
5: $\quad$ **if** $\pi_i = \pi_{i-1}$ **then** ▷ Stopping criterion met
6: $\quad\quad \forall i < j \leq m, \pi_j \leftarrow \pi_i$
7: $\quad\quad$ **break**
8: **return** $\{\pi_1, \ldots, \pi_m\}$

Algorithm 2 $\text{DAgger}(\pi^{(0)}, \mathcal{I})$

1: $D = \emptyset$
2: **for** $i = 1$ to R **do**
3: $\quad$ **for** I in instance set $\mathcal{I}$ **do**
4: $\quad\quad D \leftarrow D \cup \text{CollectData}(I, \pi^{(i-1)})$ ▷ Call ECBS
5: $\quad \pi^{(i)} \leftarrow$ train a ranking function using D
6: **return** the best $\pi^{(j)}$

iterations (line 2). In iteration i, it collects training data from solving instances in $\mathcal{I}$ using the ranking function $\pi^{(i-1)}$ obtained in iteration $i-1$, aggregates it with D (line 4), and learns a new ranking function $\pi^{(i)}$ from D that minimizes a loss function over D (line 5). Finally, the algorithm returns the best ranking function (line 6) (returns $\pi^{(0)}$ if it cannot find a better ranking function than $\pi^{(0)}$).

Data Collection Given an instance I and a ranking function π, we run ECBS using ranking function π on I and record the entire tree $\mathcal{T}$. For each node $N \in \mathcal{T}$, we compute a set of nine atomic features $\{f_1, \ldots, f_9\}$. The full list of features is deferred to Appendix A. We also obtain the set of interaction features $\{f_i f_j : i \leq j\}$ and the final feature vector $\phi(N) \in \mathbb{R}^p$ is obtained by concatenating all atomic features and interaction features. During data collection, we run ECBS with a runtime limit to collect T solutions. If the search exceeds the runtime limit without any solution, we return an empty set of data. Otherwise, for each node N, we assign a label y_N to it based on the minimum distance N_d between N and any solution found within the subtree rooted at N ($N_d = \infty$ if no solution found) as follows: 0 if $N_d < 10$; 1 if $10 \leq N_d < 30$; 2 if $30 \leq N_d < 60$; 3 if $60 \leq N_d < \infty$; 4 otherwise.

Learning a Ranking Function We focus on learning a linear ranking function $\pi : \mathbb{R}^p \rightarrow \mathbb{R} : \pi(\phi(N)) = \boldsymbol{w}^\intercal \phi(N)$ with parameter $\boldsymbol{w} \in \mathbb{R}^p$ that minimizes the loss function $L(\boldsymbol{w}) = \sum_{\mathcal{T} \in D} l(y_\mathcal{T}, \hat{y}_\mathcal{T}) + \frac{C}{2}||\boldsymbol{w}||_2^2$ over training data D, where $y_\mathcal{T}$ is the ground-truth label vector of all nodes in $\mathcal{T}$, $\hat{y}_\mathcal{T}$ is the vector of predicted values resulting from applying π to the feature vector $\phi(N)$ of every node $N \in \mathcal{T}$, $l(\cdot, \cdot)$ is a loss function measuring the difference between $y_\mathcal{T}$ and $\hat{y}_\mathcal{T}$, and $C > 0$ is a regularization parameter. The loss function $l(\cdot, \cdot)$ is based on a weighted pairwise loss. Specifically, we consider the set of ordered node pairs $\mathcal{P}_\mathcal{T} = \{(N_i, N_j) \in \mathcal{Q}_\mathcal{T} : y_{N_i} > y_{N_j}\}$ where $\mathcal{Q}_\mathcal{T}$ is the set of node pairs (N_i, N_j) such that neither N_i nor N_j is an ancestor of the other in $\mathcal{T}$. The weight w_{N_i,N_j} of each pair $(N_i, N_j) \in \mathcal{P}_\mathcal{T}$ is set to $e^{-(d_i+d_j)/rd_{\max}}$, where d_i, d_j are the depths of N_i, N_j in $\mathcal{T}$ respectively, $d_{\max}$ is the maximum depth of $\mathcal{T}$ and r is a damping factor. The loss function $l(\cdot, \cdot)$ is the weighted fraction of swapped pairs, defined as

$$l(y_\mathcal{T}, \hat{y}_\mathcal{T}) = \frac{\sum_{(N_i,N_j) \in \mathcal{P}_\mathcal{T} : \pi(\phi(N_i)) \leq \pi(\phi(N_j))} w_{N_i,N_j}}{\sum_{(N_i,N_j) \in \mathcal{P}_\mathcal{T}} w_{N_i,N_j}}. \quad (1)$$

3.2 ML-Guided Search

After learning ranking functions $\{\pi_i : i \in [m]\}$, we deploy them in ECBS. Given an instance with the same underlying graph as training and k agents, we run ECBS with ranking function π_j where $j \in \arg\min_{i \in [m]}\{|k - k_i|\}$. When a node N is generated, we compute the feature vector $\phi(N)$ and set its d-value to $\pi_j(\phi(N))$. The overall complexity for computing the d-value is $O(|N_{\mathsf{Conf}}|)$.

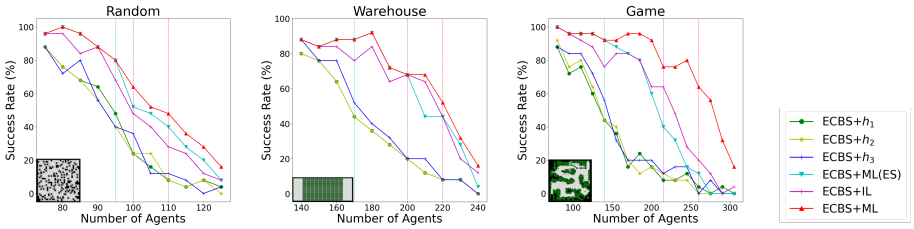

Figure 1: Success rates within the runtime limit of 5 minutes. For ECBS+ML, ECBS+ML(ES) and ECBS+IL, the vertical line with the same color indicates the number of agents in the last iteration that a ranking function is learned in Algorithm 1.

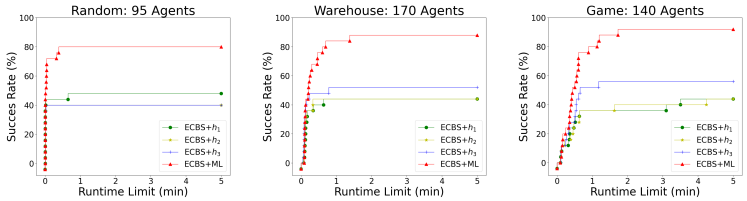

Figure 2: Success rates with a fixed number of agents varying the runtime limits for each map.

4 Experimental Results

In this section, we demonstrate the efficiency and effectiveness of our solver, ECBS with ML-guided node selection. We denote our solver as ECBS+ML. We define h_1, h_2 and h_3 to be the node-selection strategies that use the number of conflicts, the number of conflicting pairs of agents and the number of conflicting agents as the d-values, respectively. We compare against ECBS using h_i and denote them as ECBS+h_i ($i = 1, 2, 3$). We evaluate our approach on a random map, a warehouse map and a game map from the MAPF benchmark [24]. We report other experiment setups in Appendix B.

Figure 1 plots the success rates (the percentages of solved instances within the runtime limit of 5 minutes) on all grid maps. Overall, ECBS+ML significantly outperforms the three baselines in all grid maps. On the game map, in particular, the success rates of the baselines drop below 20% when the number of agents increases to 170 and the baselines could hardly solve instances with more than 245 agents, while the success rates of ECBS+ML stay above 76% for as many as 245 agents and ECBS+ML can solve 16% of the instances with 305 agents. To demonstrate the efficiency of ECBS+ML further, we show the success rates for different runtime limits on each map in Figure 2. We show one figure for each map with a fixed number of agents and defer the rest to Appendix C. Typically, the baselines and ECBS+ML tie on the easy instances but ECBS+ML gain an advantage on the hard ones, even more for instances with larger numbers of agents (as shown in Appendix C).

To assess the effect of curriculum learning, we perform two ablation analyses. First, we experiment with ECBS+ML with early stopping, denoted as ECBS+ML(ES). The number of agents in the last iteration of training in Algorithm 1 is set to the one where the success rate of ECBS+h_1 first drops below 60%. The success rates of ECBS+ML(ES) are shown in Figure 1. ECBS+ML(ES) is competitive with ECBS+ML and outperform all baselines on the random and warehouse maps, but its success rates on the game map drop dramatically after early stopping. The results imply that the learned strategies do not generalize well to larger numbers of agents in some maps and curriculum learning can help find improved strategies for those cases. Then, we experiment with ECBS+ML using only imitation learning, denoted as ECBS+IL. ECBS+IL uses the same training algorithm as ECBS+ML except that, for each number of agents, it learns a ranking function starting from the given initial ranking function for each agent size without relying on the previously-learned one. Overall, ECBS+IL can still outperform the baselines but not as significantly as ECBS+ML and its performance drops faster than ECBS+ML when the number of agents increases. The results show another two advantages of curriculum learning: (1) it enables learning for one to three more iterations than ECBS+IL by enabling the algorithm to collect more data for training since it starts with a higher success rate in DAgger$(\cdot, \cdot)$; (2) it obtains better node-selection strategies based on previously-learned strategies, as opposed to ECBS+IL that learns from scratch in every iteration.

References

[1] M. Barer, G. Sharon, R. Stern, and A. Felner. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Seventh Annual Symposium on Combinatorial Search*, 2014.

[2] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

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

[4] L. Cohen, T. Uras, and S. Koenig. Feasibility study: Using highways for bounded-suboptimal multi-agent path finding. In *Eighth Annual Symposium on Combinatorial Search*, 2015.

[5] L. Cohen, T. Uras, T. S. Kumar, H. Xu, N. Ayanian, and S. Koenig. Improved solvers for bounded-suboptimal multi-agent path finding. In *IJCAI*, pages 3067–3074, 2016.

[6] H. Daumé, J. Langford, and D. Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009.

[7] K. Dresner and P. Stone. A multiagent approach to autonomous intersection management. *Journal of Artificial Intelligence Research*, 31:591–656, 2008.

[8] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874, 2008.

[9] A. Felner, J. Li, E. Boyarski, H. Ma, L. Cohen, T. S. Kumar, and S. Koenig. Adding heuristics to conflict-based search for multi-agent path finding. In *International Conference on Automated Planning and Scheduling*, 2018.

[10] H. He, H. Daume III, and J. M. Eisner. Learning to search in branch and bound algorithms. In *Advances in Neural Information Processing Systems*, pages 3293–3301, 2014.

[11] 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*, 4(2):1125–1131, 2019.

[12] E. B. Khalil, B. Dilkina, G. L. Nemhauser, S. Ahmed, and Y. Shao. Learning to run heuristics in tree search. In *International Joint Conference on Artificial Intelligence*, pages 659–666, 2017.

[13] E. B. Khalil, P. Le Bodic, L. Song, G. L. Nemhauser, and B. Dilkina. Learning to branch in mixed integer programming. In *AAAI Conference on Artificial Intelligence*, 2016.

[14] E. Lam and P. Le Bodic. New valid inequalities in branch-and-cut-and-price for multi-agent path finding. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 184–192, 2020.

[15] E. Lam, P. Le Bodic, D. D. Harabor, and P. J. Stuckey. Branch-and-cut-and-price for multi-agent pathfinding. In *IJCAI*, pages 1289–1296, 2019.

[16] J. Li, E. Boyarski, A. Felner, H. Ma, and S. Koenig. Improved heuristics for multi-agent path finding with conflict-based search: Preliminary results. In *Annual Symposium on Combinatorial Search*, 2019.

[17] H. Ma, J. Li, T. S. Kumar, and S. Koenig. Lifelong multi-agent path finding for online pickup and delivery tasks. In *Conference on Autonomous Agents and MultiAgent Systems*, pages 837–845, 2017.

[18] H. Ma, J. Yang, L. Cohen, T. S. Kumar, and S. Koenig. Feasibility study: Moving non-homogeneous teams in congested video game environments. In *Artificial Intelligence and Interactive Digital Entertainment Conference*, 2017.

[19] J. Pearl and J. H. Kim. Studies in semi-admissible heuristics. *IEEE transactions on pattern analysis and machine intelligence*, (4):392–399, 1982.

[20] S. Ross and D. Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668, 2010.

[21] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

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

[23] J. Song, R. Lanka, A. Zhao, A. Bhatnagar, Y. Yue, and M. Ono. Learning to search via retrospective imitation. *arXiv preprint arXiv:1804.00846*, 2018.

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

[25] J. Yu and S. M. LaValle. Planning optimal paths for multiple robots on graphs. In *2013 IEEE International Conference on Robotics and Automation*, pages 3612–3617. IEEE, 2013.

Appendix: Learning Node-Selection Strategies in Bounded Suboptimal Conflict-Based Search for Multi-Agent Path Finding

A Feature List

For each node $N \in \mathcal{T}$, we compute the following set of atomic features $\{f_1, \dots, f_9\}$:

1. Features related to the conflicts: number of conflicts $|N_{\mathsf{Conf}}|$ (denoted as f_1), number of pairs of agents that have at least one conflict (denoted as f_2) and number of agents that have at least one conflict (denoted as f_3);
2. Features related to N_{Cost}: $f_4 := N_{\mathsf{Cost}}$, $f_5 := \frac{N_{\mathsf{Cost}}}{\mathsf{LB}}$, $f_6 := N_{\mathsf{Cost}} - \mathsf{LB}$, $f_7 := N_{\mathsf{Cost}} - S$, $f_8 := N_{\mathsf{Cost}}/S$ where S is the sum of costs of individually cost-minimal paths over all agents.
3. The depth of N (denoted as f_9).

B Experiment Setup

We implement ECBS in C++. The experiments are conducted on 2.4 GHz Intel Core i7 CPUs with 16GB RAM. For each instance, the runtime limit is set to 5 minutes in both data collection and testing. The number of solutions T collected during data collection is set to 10. The number of iterations R for DAgger is set to 10. The damping factor r for weight w_{N_i,N_j} is set to 0.3727. r is chosen such that the root node has weight 1 and a node at depth $0.6d_{\max}$ has weight $e^{-0.6/r} = 0.2$. Since we are using a pairwise loss, we suffer quadratic complexity ($O(|\mathcal{T}|^2)$) of the loss computation. To tackle this issue, we record only the first 10,000 nodes generated for each instance during data collection. We use an open-source ML package LIBLINEAR [8] in the step of training a ranking function in Algorithm 2 and use the default values for all parameters in LIBLINEAR where $C = 1$. The performance of our algorithms is not sensitive to hyperparameters and all these parameters are chosen without fine-tuning.

We evaluate our algorithms on three grid maps of different sizes and structures from the MAPF benchmarks[1] [24], including: (1) a random map "random-32-32-20", a 32×32 grid map with 20% randomly blocked cells; (2) a warehouse map "warehouse-10-20-10-2-1", a 163×63 grid map with 200 10×2 rectangle obstacles; (3) a game map "den520d", a 257×256 grid map from the video game *Dragon Age: Origins*. For testing, we use the "random" scenarios in the benchmarks, yielding 25 instances for each number of agents on each grid map. For training, we generate another 25 instances drawn from the same distribution as the "random" scenarios for each $\mathcal{I}_i$ in Algorithm 1. Parameters related to each individual grid map are listed in Table 2. For each grid map, we fix the suboptimality factor w, following similar reasonings in previous work [1] where small w values are chosen for large grid maps and vice versa. Our goal is to obtain a ranking function π_i ($i \in [m]$) for each number of agents in $\{k_1, \dots, k_m\}$. The training loss of the learned ranking functions are shown in Table 2, showing good ML performance of the learned ranking functions. We test the learned strategies on unseen instances with numbers of agents $k_1, \dots, k_m$, and report the improvement in solution performance, which is the end goal of this work.

C Additional Experimental Results

The success rates for different runtime limits varying the number of agents on the random, warehouse and game maps are shown in Figure 3.

[1] All data is publicly available at: `https://movingai.com/benchmarks/mapf/index.html`.

Map	Random	Warehouse	Game
w	1.1	1.05	1.005
m	10	11	16
k_1	75	140	80
k_m	125	240	305
$\|V\|$	819	5,699	28,178

Table 1: Parameters of each grid map. w is the suboptimality factor, m is the number of different numbers of agents we train and test on, k_1 is the number of agents that we start training on, k_m is the largest number of agents that we test on, and $|V|$ is the number of empty cells on the grid map. $k_2, \cdots, k_{m-1}$ are evenly distributed on $[k_1, k_m]$, i.e., $k_i = (i-1)(k_m - k_1)/(m-1) + k_1$.

Map	Random	Warehouse	Game
l_1	.0075	.0088	.0085
$l_{\lfloor m/2 \rfloor}$	.0330	.0166	.0131
l_m	.0653	.0283	.0204

Table 2: $l_i \in [0, 1]$ is the loss of π_i for k_i agents evaluated by Equation (1) averaged over all CTs in the training data.

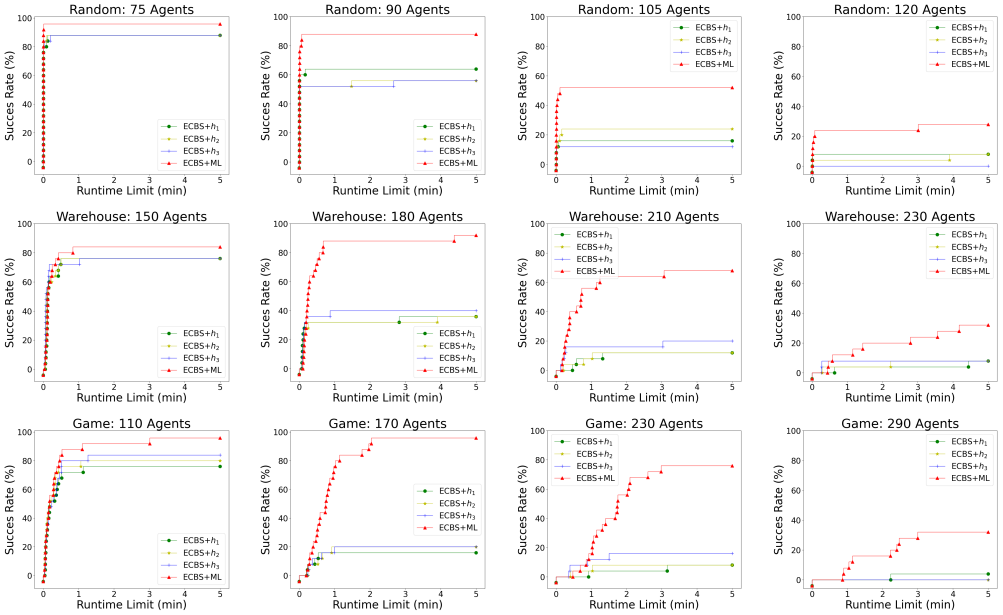

Figure 3: Success rates varying problem parameters with different runtime limits.