No Panacea in Planning: Algorithm Selection for Suboptimal Multi-Agent Path Finding

Weizhe Chen,[1] **Zhihan Wang,** [1] **Jiaoyang Li,** [2] **Sven Koenig,** [1] **Bistra Dilkina** [1]

[1] University of Southern California
[2] Carnegie Mellon University
weizhech@usc.edu, zhihanwa@usc.edu, jiaoyangli@cmu.edu, skoenig@usc.edu, dilkina@usc.edu

Abstract

Since more and more algorithms are proposed for multi-agent path finding (MAPF) and each of them has many hyperparameters to be specified, choosing one for a specific scenario that fulfills some requirements is a very important task. Previous research in algorithm selection for MAPF mostly focused on optimal algorithms and showed that machine learning could help. In this paper, we study algorithm selection for general solvers for MAPF, which further includes choosing between different suboptimal algorithms and choosing between solvers from different hyperparameters of the same algorithm. We formulate the problem as a group of prediction problems with different optimization objectives, which handle the new tradeoff between runtime and solution quality introduced by suboptimal algorithms, and different metrics to evaluate the learning model. Then we propose a group of learning tasks to solve these production problems. We identify the issue of always using resize for inputs. We use extensive experiments to show how different learning tasks should be used for different problems. We also discuss how to choose machine learning models for MAPF algorithm selection to balance the model size and the final performance.

Introduction

Multi-agent path finding (MAPF) is the problem that considers how to generate a set of collision-free paths for a team of agents given the start and goal locations of each agent while minimizing some optimization objectives (such as travel times, travel distances, etc.) (Stern et al. 2019). In recent years, MAPF has been an emerging research domain that has gotten a lot of focus because of its wide applications in warehouses, airport schedules, and autonomous driving. While the problem is proved to be NP-hard in certain cases (Yu and LaValle 2013; Ma et al. 2016; Nebel 2020) and also hard in practice in other cases, many algorithms are created to find optimal, bounded suboptimal or suboptimal solutions. For the same standard benchmark, new algorithms can solve the scenarios faster and faster or give better solutions within the same time limit. However, there is no silver bullet in the MAPF research community: Even the latest algorithm does not perform the best in every scenario (Okumura et al. 2022), and different hyperparameters are needed for the best performance in different scenarios. Due to the extremely large number of possible combinations between algorithms and hyperparameters, running them all one by one or in parallel is very impractical. Therefore, for real-world deployment, when there is a previously known map with a strict time limit and solution quality requirement, choosing the specific algorithm and specific hyperparameters is a very important task, and algorithm selection in MAPF has become a focus for all potential users in both research and industry.

Researchers have already shown that machine-learning-based algorithm selection can learn a way to efficiently propose the fastest algorithm in specific scenarios (Kaduri, Boyarski, and Stern 2020). However, most previous works focused on only optimal algorithms with their speeds as the sole algorithm selection criteria and did not consider the emerging research on suboptimal algorithms. Since suboptimal algorithms do not need to find the optimal solution, their solutions can be extensively more different from each other than those of optimal algorithms, and considering both solution cost and runtime naturally make the selection problem a bi-objective optimization problem. That is why studying algorithm selection problem for suboptimal algorithms are naturally more difficult than for optimal algorithms. In practice, users usually want to make sure that the cost of the solution is not too bad or want to trade off between the runtime of the algorithm and the cost of the solution. However, different users may have different tradeoff preferences, making giving one algorithm-selection algorithm very hard.

In this paper, we address the algorithm-selection problem for solvers for MAPF without the specific limit for focusing on optimal algorithms only and without the specification that is selecting algorithms between completely different algorithms but also selecting between different hyperparameters for a single algorithm. We formulate the problem as a prediction problem and use image-based machine learning to make the prediction. We propose multiple groups of objectives where each objective can be a way to trade off the cost and the runtime. We realize that there are large differences between different objectives and different learning metrics. So, unlike other works, we specifically propose a group of different learning tasks for different metrics, instead of using the same training scheme and evaluating it on multiple tasks. Based on a newly constructed dataset built on the stan-

dard MAPF benchmark (Stern et al. 2019), we use extensive experiments to show that using the same loss function all the time, which is what the prior works on algorithm selection for optimal MAPF did, can make the model perform bad for specific objectives in algorithm selection for suboptimal MAPF. We also show how different machine learning models can make a difference in the algorithm selection performance. We further use an ablation study to identify how different features should be treated differently during the rescaling period. We also discuss how to choose the neural network used in the algorithm selection problem for MAPF that is fast, easy to train, and performs well.

The main contributions of this paper are:

1. We are the first to study the algorithm selection problem that considers both runtime and solution costs in suboptimal MAPF algorithms.
2. We are the first to propose and verify that different ways of training should be used for different metrics with different objectives in algorithm selection for MAPF.
3. We are the first to evaluate many modern computer vision models for algorithm selection for MAPF and the first to show that feature-sensitive pre-processing in the phase of machine learning is important.
4. We are the first to address the problem of hyperparameter selection in MAPF.

Related Work

The multi-agent path finding (MAPF) problem is the problem of finding a set of conflict-free paths for a set of agents in a known environment while minimizing their travel times. Specifically, in this paper, we consider exactly the same problem as (Stern et al. 2019; Li et al. 2022), which is a four-connected grid map, where each agent is given a start cell and a goal cell. A scenario is defined as the combination of the description of the map, which is the size of the map and which cells have obstacles, the start cells, and the goal cells of each agent. At each timestep, an agent can move to an adjacent cell or stay in its current cell. A conflict happens if two agents result in the same cell at the same timestep. Each agent remains at its goal cell after it arrives until every agent arrives at their goals. The quality of one solution is the total sum of travel times of each agent, which is also known as the sum of costs in the MAPF community. While there are a few works that directly generated solutions from machine learning (Laurent et al. 2020), MAPF is nowadays mainly solved with more classic methods like heuristic search algorithms (Sharon et al. 2015), rule-based algorithms (Han and Yu 2020), and reduction-based algorithms (Surynek et al. 2016). There are mainly two groups of algorithms, namely the optimal algorithms and the suboptimal algorithms. The optimal algorithms like Conflict-Based Search (CBS) (Sharon et al. 2015) are guaranteed to generate a solution that is optimal but usually requires a long runtime. Suboptimal algorithms usually generate good-quality solutions faster but do not guarantee the founded solution to be the best one. Within suboptimal algorithms, the algorithms can be further classified into two groups, namely, ones that still guarantee the solution quality to be within a suboptimality bound like Explicit Estimation CBS (EECBS) (Li et al. 2021) and ones that do not like Prioritized Planning (PP) (Silver 2005), Parallel Push-and-Swap (PPS) (Sajid, Luna, and Bekris 2012), and Priority Inheritance with Backtracking (PIBT+) (Okumura et al. 2022). In this paper, we consider optimal algorithms and both types of suboptimal algorithms as candidate algorithms.

Algorithm selection is the problem of selecting a specific algorithm for a specific scenario (Smith-Miles 2008). It has been successfully used in many optimization problems, including satisfiability and traveling salesman problem (TSP) (Kerschke et al. 2018; Xu et al. 2012). The rich studies in those domains have built up a standard way of measuring the performance of selected algorithms for their specific way of usage. However, algorithm selection in MAPF is still at an early stage, where most papers (Sigurdson et al. 2019; Ren et al. 2021) just show that, with a specific technique, the selection algorithm could be helpful in choosing a correct solver algorithm but lacked thorough performance metrics. (Sigurdson et al. 2019) is the first to introduce the algorithm selection problem into MAPF. They proposed a modified version of AlexNet (Krizhevsky, Sutskever, and Hinton 2012) to make their prediction and showed that it is possible to predict the fastest algorithm in MAPF. (Kaduri, Boyarski, and Stern 2020) improved the results by using VGGNet (Simonyan and Zisserman 2015) and gradient boosted decision tree with XGBoost (Chen and Guestrin 2016). (Ren et al. 2021) proposed MAPFAST, which added more features related to the shortest path between the start and goal locations of each agent. They also added two auxiliary output channels into the loss function to help find the fastest algorithm, and used an inception-based neural network (Szegedy et al. 2015) to train the model. (Kaduri, Boyarski, and Stern 2021) empirically showed that different types of maps have different preferences for different algorithms, and thus verified the usefulness of algorithm selection. Previous works focused on either only the accuracy of finding the fastest or only the coverage rate, which is how likely the chosen algorithm can finish in a given time limit. Since both criteria are only related to the runtime of the algorithms, in this paper, we focus on not only optimal algorithms but also suboptimal algorithms, and consider how to take both runtime and cost into account at the same time. We also explore more modern neural network architectures to verify if modern architecture can help in prediction.

In this paper, we formulate the problem as an image-based prediction problem, a popular topic in computer vision. Many works can be used to solve the problem. AlexNet (Krizhevsky, Sutskever, and Hinton 2012) was the first model that shows that deep learning can achieve success in class prediction. VGGNet (Simonyan and Zisserman 2015) was then proposed to show that using a small kernel size can stably predict good results while keeping the total number of training parameters low. GoogLeNet (Szegedy et al. 2015) was proposed about the same time for the same purpose but proposed an inception unit as their solution. ResNet (He et al. 2016) proposed to use skip links between layers to solve the vanishing gradient problem, making very deep architectures possible, and has become one of the most

commonly used models in computer vision. In recent years, Transformer (Vaswani et al. 2017) has been more and more popular in all machine learning research, including computer vision. Vision Transformer (ViT) (Dosovitskiy et al. 2021) proposed to split the image into smaller blocks for transformer to use, and is now a popular and successful model in computer vision. In this paper, we choose to test four of the aforementioned neural networks that are popular in computer vision and algorithm selection in MAPF research.

Preliminary

Algorithm selection in MAPF is the problem of choosing a suitable algorithm that is the best in a given scenario . Generally, algorithm selection includes both selecting completely different algorithms and choosing the hyperparameter(s) of a fixed algorithm. For clarity, we use the word solver to refer to these different candidates, but we still call the problem algorithm selection to be consistent with other papers that solve the same problem. The input for the algorithm selection includes exactly the same information as a MAPF solver knows without adding any information from each candidate solver. The selection algorithm is then required to output the best solver. Typically, machine-learning-based selection algorithms first transform the scenario information to desired formats and features they would like to use as input and then use the learning model to output the answer directly. We follow the same workflow in this paper.

Previous works on algorithm selection for MAPF mostly consider optimal solvers only and thus set up runtime as the only objective for choosing the solver. However, using a single objective is not applicable to more general algorithms. Although suboptimal solvers generally run faster than optimal solvers and can solve many scenarios that optimal solvers cannot solve, they do not have any guarantees on their solution qualities. Thus how to trade off between their solution qualities and their runtimes has been a long focus for potential users. In reality, people may have their own way of judging how important some objectives are compared to others. They might want to use a weighted sum to estimate a score for the solver and want the best one, or they just want to ensure the solution they get is not so bad in all objectives. These tasks are similar but different, and most previous research in algorithm selection (not limited to those for MAPF) optimizes just accuracy after defining the score (Heins et al. 2021). While that is acceptable for algorithm selection for TSP and SAT because their specific way of application does not care about runtime and cost at the same time, we need to care about them at the same time in MAPF.

In this paper, we formulate our algorithm selection as a standard image-based-input prediction problem solved by computer vision techniques. It is remarkable that, in the last decade, computer vision has earned a great improvement with tens of thousands of papers every year. There are many new neural network models created and evaluated on standard datasets like ImageNet (Yang et al. 2020) and CIFAR-10 (Krizhevsky and Hinton 2009), and many models have successfully been used in different applications like autonomous driving, face recognition, and many novel applications. These successful applications, in turn, encourage the investment and development of computer vision. Furthermore, these models have successfully shown that with specific data augmentation, raw image input without human-involved complex pre-processing can be as good as using more manually designed features. That is why we would like to explore many modern computer vision models to see if they can directly generate some good results. However, as we will later show in the experiment section, we find that there is no panacea to planning: Using the same model with the same loss function does not work for all objectives.

Algorithm Selection for Suboptimal MAPF

Dataset

Because there is no standard dataset for algorithm selection in MAPF right now, we first describe how we build our own datasets before we describe our learning tasks.

Candidate Solvers In this paper, we build two separate datasets for the purpose of doing standard algorithm selection between different algorithms, and for the purpose of doing solver selection between different solvers that are from the same algorithm.

For the standard algorithm selection part, we enumerate some of the most commonly used MAPF solvers nowadays. Our candidate algorithms are: CBS (Sharon et al. 2015), EECBS (Li et al. 2021), PP (Silver 2005), PPS (Sajid, Luna, and Bekris 2012), and PIBT+ (Okumura et al. 2022) . We did not include some recent research like large neighborhood search (LNS) (Li et al. 2021) which can use different MAPF solvers as subsolvers. LNS-based solvers can solve any scenarios that are solvable by other solvers by using them as the initial solver with just a small overhead and can solve more scenarios when the later neighborhood search actually happens. So comparing them to more standard MAPF solvers is generally not fair. Theoretically, LNS users can also use our model to choose which algorithm they want to use as their subsolvers.

For the hyperparameter selection, we want to select the optimality bound for EECBS. The optimality gap w in EECBS guarantees the solution found to be within w times compared to an optimal solution, and more time will be spent during the search almost for sure if a small and tight bound is given. But a large w can also find near-optimal or even optimal solutions in specific scenarios, so choosing the optimality gap can make the runtime much faster in those scenarios where giving a small value is not helpful for EECBS to find a better solution. Specifically, we aim to choose the optimality gap from $w = 1.05, 1.1, 1.15, 1.2$.

Features Features have long been known as a very important factor in machine learning research related to MAPF. Most previous works on algorithm selection for MAPF used the scenario information without any pre-processing (Sigurdson et al. 2019; Kaduri, Boyarski, and Stern 2020). One exception (Ren et al. 2021) considered only a given shortest path without specifying a clear way to determine which shortest path to use when an agent has multiple shortest

paths. It also over-compressed different kinds of information into each channel, which can largely reduce the potential performance. So, in this work, we use the image representation of the MAPF scenarios from the MAPF benchmark and propose a new set of features, each one of which is a separate channel fed into the model.

The features we used are listed here in order:

1. Whether this specific cell is an obstacle. This is always the same in scenarios generated from the same map.
2. Whether this specific cell is a start cell. Different start cells do not have any further differences.
3. Similar to start cells, an indicator of whether the specific cell is a goal cell for any agent.
4. If everyone follows its shortest path selected by (Han and Yu 2020) without considering any conflict, how many times will the cell be visited.
5. Total number of conflicts on the cell between all pairs of shortest paths from different agents.
6. Total number of conflicts on the cell between all pairs of shortest paths and 1-suboptimal paths from different agents that happens on the cell.
7. How many times will the cell be visited if every agent tries every possible shortest path on its own without considering any conflict.

Here, we define a 1-suboptimal path as a path whose length is the length of the shortest path plus 1. In this paper, we make the input a rectangle image, and then we normalize all the input features to 0 to 100 by dividing every channel with the maximum value of all maps for a specific channel.

We realize that although cells with obstacles can never be occupied or visited by any agents, treating those cells with a value of 0 is not always helping the learning. So, we change all obstacle cells to a value of 200 for the (4) and (7) channels listed above, which is the heatmap that sums up the shortest path from a given set. We choose to just change these two channels both after some empirical results on a not complete enumeration, and from the intuition, deep learning models may want to learn the total density from the heatmap so having an extra obstacle in the map should make the map more crowded than make them less crowded, which should be related to the average value of these two channels.

After defining the feature, we need to pre-process the data to make the input fit into the same neural network so we do not need to train separate models for different maps. While previous works (Sigurdson et al. 2019) all use a default resize [1], which is also known as interpolation in many fields, to make all image to be of the same size without even mentioning that in the paper, we realize that this is not always the correct thing to do. Although we formulate our problem as an image-based input problem like computer vision, this resizing way works properly in computer vision because the pictures have a good property of zooming invariance, i.e., a hand is still a hand no matter how many times it is zoomed

[1]Researchers in MAPFAST (Ren et al. 2021) said they use padding to formalize the input but in fact, they are not using it according to their public code on Github.

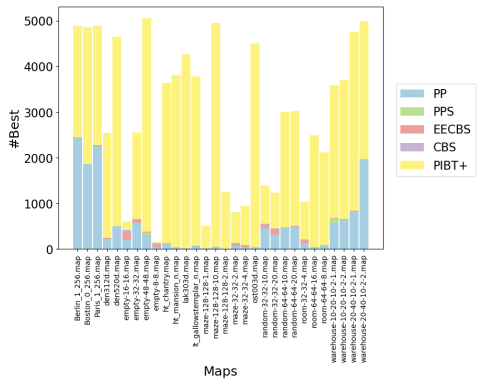

Figure 1: The frequency of each algorithm being the best one on a given map when $w = 0.001$ in Eq. 3. Different maps have different numbers of total scenarios because different maps are different in size and obstacles so the total capacity is naturally different.

in or zoomed out. However, in MAPF, each agent can only move one cell at one timestep, so zooming invariance is not held. So using resize as the way to rescale images of different sizes to the same sizes is not a proper way. So we give another option for pre-processing the data, which is for each given image, we make the original image in the center, and pad the image to a fixed size of 384×384 with the number we get from an obstacle. Specifically, the value padded is not a fixed number in different channels because obstacles are not necessarily getting the same value in different channels, and giving the padded part a different value compared to obstacles will lead to some inconsistency.

Labels With all the candidate solvers, we run all solvers on all maps in the MAPF benchmark (Stern et al. 2019). For each solver, we enumerate different numbers of agents with a step size of 10 agents until there are no more than 2 solvers that can solve any larger scenario within the time limit of 2 minutes on the map. With all these generations together with the features we have just defined, we get an algorithm selection dataset of 89,940 data points and a hyperparameter selection dataset of 53,691 data points. It is noteworthy that the algorithm selection dataset is a much larger dataset compared to any previously used dataset in (Kaduri, Boyarski, and Stern 2020; Ren et al. 2021), and a reasonable size for trying modern deep learning models that have millions of parameters without immediately overfitting.

Optimization Objectives

Previous works on algorithm selection for MAPF typically consider the runtime or success rate (at a fixed runtime threshold) of an solver as the only objective and never take solution quality into account. In this way, they can hopefully always deliver a solution to the deployed scenario. However, the actual solution quality can vary largely across different solvers in different maps when we consider suboptimal solvers. So we need to consider both runtime and cost (solution quality) at the same time. While there are many ways

to do this, we use this section to give two intuitive ways of defining the optimization objective.

We first normalize the time and the cost to a similar scale, in a way that is independent of the set of candidate solvers. Suppose the time limit is $time_{limit}$, and the sum of costs for shortest paths for every single agent is $cost_{bound}$ in the scenario, the time and cost used in Eq. 3 are calculated by:

$$time' = \frac{time}{time_{limit}} \tag{1}$$

$$cost' = \frac{cost}{cost_{bound}}, \tag{2}$$

where $time'$ and $cost'$ are the actual values used in Eq. 3. Furthermore, if in any scenario, a specific solver cannot be finished in the time limit $time_{limit}$, the time used for calculation is defined as $5 \times time_{limit}$, and the cost used for calculation is $5 \times cost_{min}$, where $cost_{min}$ are the minimum sum of costs in all success candidate solvers (i.e., all candidate solvers that find solutions within the time limit).

The first, and the most common way of doing so is to use a weighted sum of different objectives as the score (Bischl et al. 2016; Heins et al. 2021; Seiler et al. 2020). Specifically, we calculate the following objective $score$:

$$score(a) = time'_a + w * cost'_a, \tag{3}$$

where w is the hyperparameter that users can control to represent their preference, and $time'$ and $cost'$ are the normalized metrics we calculated above. Then we will want to find the solver with the best score:

$$\min_a \quad score(a) \tag{4}$$

The second group of objectives is to choose a solver that gives a solution within a given cost bound the fastest. Specifically, we find the solver a so that:

$$\begin{aligned} \min_a \quad & time_a \\ s.t. \quad & cost_a \leq bound * cost_{min}, \end{aligned} \tag{5}$$

where $cost_{min}$ is still the minimum sum of costs in all success candidate solvers. This group of objectives is useful in the case that the users just need some guarantee on the solution quality, but as long as the solution is a relatively good one the runtime becomes the only consideration.

With different objectives, we can have the same input but different labels from one data point. In Fig. 1, we show the frequency of different solvers being the best in different scenarios. We can see that while trading off the runtime and cost at different weights, the best solvers are changing a lot. We further provide the frequency plots of all unique tasks in the appendix. We can find that the single best solver that solves most cases is changing while we change the weight and the tasks. It is also different between maps in how large the difference between single best solvers and other solvers is even if the single best solvers are the same.

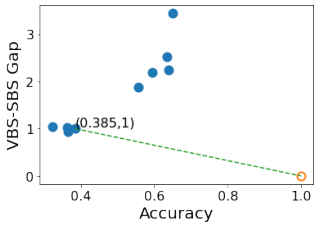

Figure 2: Performance of Algorithm Selection Models in terms of accuracy (higher is better) and VBS-SBS gap (lower is better). Blue points are actual samples we collected in different models. The green dotted line is an imagined line of how the model can be better given the known optimal point (1,0) marked as orange circle.

Metrics for Learning

After defining the optimization objectives, we briefly talk about the two metrics we use in this paper.

The first metric is the standard accuracy, which is used both in popular machine learning research, and in earlier papers in algorithm selection for MAPF (Ren et al. 2021).

Accuracy is normally a good metric in ML, but it only takes right or wrong into account, not how wrong the model is. On the other hand, if we really want to use an algorithm selector, we definitely want to know how bad the wrong choices can lead to as long as the selector is not perfect. So the second group of objectives is a well-known metric in algorithm selection, the VBS-SBS gap (Xu et al. 2012). Given a pre-calculated virtual best solver (VBS) that always outputs the best solver and a single best solver (SBS) that is the posterior best possible solver that always outputs the same solver, the gap for the current solver a is calculated by:

$$gap(a) = \frac{\overline{score(a)} - \overline{score(VBS)}}{\overline{score(SBS)} - \overline{score(VBS)}}, \tag{6}$$

where $\overline{score}$ is the average score function over all data points, which can be the score function we defined in Eq. 3, or just runtime if we use the second optimization objective. This objective is commonly known as a metric for a prediction model, with 0 as the theoretical best and 1 as the same performance as the prediction model that always outputs the single best solver. A good model should have a gap smaller than 1, and the smaller the gap is, the better the model is.

Although there are some people already working with this metric, this metric does not get better in the same direction with accuracy. Intuitionally, if a user chooses to optimize accuracy, the model can easily converge to a group of weights that mostly choose the common scenarios. However, that result can also be the worst one when it is wrong, thus leading to bad results in any metrics that give a penalty for outputting a wrong answer depending on how wrong the answer is. As shown in Fig. 2, a large accuracy does not guarantee a small VBS-SBS gap, and, in most parts where a model can reach nowadays, a small gap requires a drop in accuracy. So we believe the previous workflow of training a model and then evaluating it with both metrics is not the proper way.

Learning Tasks

There are typically three ways to do algorithm selection:

1. Classification: The most standard way is to treat the problem as a classification problem. The model predicts the probability of choosing each solver and is trained with typical classification loss. For inference, the solver with the largest probability is selected.
2. Regression on Expected Score: This method still predicts the probability of each candidate being the best, but it now uses a regression-based loss instead of a classification-based loss during training. Every time, the probability got from the output is used to calculate the expected score by using the model following the probability, and the loss is a function related to how much difference this score is compared to the VBS.
3. Regression on Score Prediction: Instead of giving the probability of choosing each solver, models can be trained to predict the score of a given solver, and every time for inference, it predicts the expected score for every model and choose the model with the best score.

In this paper, based on the time and actual tasks we have, we only use the first two ways of doing algorithm selection to build our learning task, specifically, classification and regression for the expected scores. We choose to use two variants of classification and one way of regression.

The first variant of classification is the most popular version, which is to use a cross-entropy loss for the classification problem. We call this method CE (cross-entropy).

The second variant of classification is by simply changing the cross-entropy loss to a binary cross-entropy loss, which we call BCE (binary cross-entropy).

The third way of learning is the regression on the expected scores we described above. With the possibility get from the model, we calculate the expected score and compute Huber loss (Huber 1992) between our expected score and the score of VBS. We call this learning way Reg (Regression).

A learning *task* is the combination of optimization objectives and a specific learning way. We use Score-w-y to refer to the optimization task in Eq. 4, where w is w from Eq. 4 and y is the learning way to create a learning task. We use Bound-b-y to refer to tasks optimizing as Eq. 5, where b is the value of bound in the equation. For example, Score-0.001-CE is the task that we set $w = 0.001$ in Eq. 3 for score definition, and we use cross entropy to do the training. Bound-1.1-BCE means the optimization tasks with $bound = 1.1$ in Eq. 5, and optimized by the binary cross-entropy loss.

Experiment

Experiment Settings

In this paper, we develop our machine learning model based on 4 different computer vision models: VGG16 (VGGNet) (Simonyan and Zisserman 2015), ViT-Tiny (ViT) (Dosovitskiy et al. 2021; Wightman 2019), MAPFAST (Ren et al. 2021), ResNet-18 (He et al. 2016), based on the standard Timm library (Wightman 2019). Surprisingly, we found that the auxiliary tasks used in MAPFAST are not helping the result for any tasks in our dataset, so all numbers of MAPFAST are from models trained with only the first output channel that directly outputs the probability of each solver, and optimized as a standard classification problem. Remark that, unlike previous papers, we do not train any results on decision trees because previous papers have already shown that neural networks can be better than decision tree models in most cases. It needs to be addressed that ViT is very different from any other model because it is not a convolutional neural network. We hope that ViT can perform differently because it does not have the limit of the small kernel size, nor the shifted invariance property that CNN generally has.

While we primarily set our baseline as getting a better number than SBS, our SBS is selected separately for accuracy (SBS_{Acc}) and VBS-SBS gap (SBS_{Gap}). From intuition, SBS_{Acc} is the solver that is the most common in Fig. 1, while SBS_{Gap} is the one that can solve the most instance, therefore it does not get any large failing penalty. Because of the dominance of each solver in our setting, the SBS for the entire dataset is the same as the SBS per grid or per map type.

For every model, we use some data augmentation methods to prevent the model from overfitting, which include random flip, random rotation, and random erasing with a probability of 0.5. We decide to use resize only in the 4th to the 6th channels of the feature and padding in other channels because they are empirically the best as we will later show in Sec. .

The running results of all MAPF solvers are collected on the same AWS EC2 m4 server, while the training and testing of all machine learning models are conducted on a xeon-6130 server with a single NVIDIA-V100 and 184GB RAM. All parameters are selected by grid-search, and the full hyperparameter table is provided in the appendix.

Results on Different Models

We show our results in Table. 1. Each horizontal block shows one group of tasks that have similar optimization objectives, but with different ways of creating the learning tasks metrics by different learning metrics. From the table, we can know that there is no learning task that can be good in both accuracy and VBS-gap. Regression is the most promising way to build a learning task when using the VBS-SBS metrics, while CE and BCE are better when using the accuracy metrics.

In all groups of tasks, there are always some learning tasks with some machine learning models that get a VBS-SBS gap smaller than 1, and better accuracy compared to SBS. This means that algorithm selection in suboptimal solvers and hyperparameter selection between different solvers from the same algorithm are both doable.

Compared between different neural networks, ViT performs the best in most VBS-SBS gap metrics and some of the accuracy metrics. Other models have their own advantage and disadvantage in different learning tasks, and no model has won the majority of tasks that is not won by ViT.

Ablation Study on Feature Rescale Method

In earlier papers using computer vision based models (Kaduri, Boyarski, and Stern 2020; Sigurdson et al. 2019;

Dataset	Task	MAPFAST$_{cl}$		ViT		VGGNet		ResNet		SBS$_{Acc}$	SBS$_{Gap}$
		Acc	Gap	Acc	Gap	Acc	Gap	Acc	Gap	Acc	Gap
Standard	Score-0.001-CE	0.81	21.07	0.81	19.03	0.84	20.49	**0.91**	2.60	0.80	1
	Score-0.001-BCE	**0.85**	4.48	**0.83**	18.67	**0.85**	22.37	0.86	50.94	0.80	1
	Score-0.001-Reg	0.80	**0.89**	0.79	**0.87**	0.80	**1.00**	0.80	**0.94**	0.80	1
	Score-0.1-CE	0.61	16.73	0.61	14.83	**0.69**	12.90	**0.61**	1.11	0.57	1
	Score-0.1-BCE	**0.65**	10.74	**0.62**	14.79	0.67	9.20	0.56	2.55	0.57	1
	Score-0.1-Reg	0.57	**0.75**	0.57	**0.71**	0.57	**1.00**	0.57	**1.00**	0.57	1
	Score-1-CE	**0.62**	1.82	**0.69**	1.95	**0.66**	2.09	0.60	3.52	0.62	1
	Score-1-BCE	0.57	4.42	**0.68**	1.89	0.64	1.96	**0.62**	2.53	0.62	1
	Score-1-Reg	0.56	**0.71**	0.30	**0.92**	0.26	**1.00**	0.47	**0.75**	0.62	1
	Bound-1.1-CE	0.56	2.99	0.65	2.66	0.69	2.81	**0.61**	4.24	0.51	1
	Bound-1.2-CE	**0.72**	1.64	**0.74**	2.07	**0.73**	4.59	0.58	2.20	0.51	1
EECBS	Score-0.001-CE	**0.70**	1.11	**0.71**	1.12	**0.70**	0.97	**0.69**	1.00	0.69	1
	Score-0.001-BCE	0.69	1.05	0.69	**1.00**	**0.70**	**0.93**	**0.69**	1.00	0.69	1
	Score-0.001-Reg	0.69	**1.00**	0.69	**1.00**	0.69	1.00	**0.69**	**0.93**	0.69	1

Table 1: The accuracy (Acc) and VBS-SBS gap (Gap) results for different models and different learning tasks in terms of accuracy and VBS-SBS gap. The names of the tasks are shown in the Task column, following the naming described in the experiment setting section. The best results for each metric on the same optimization objectives are marked with bold (Bound tasks do not have bold in Gap metrics because in bound tasks accuracy is normally the primary focus).

Padding(p), Resize(r)	Gap	Acc
ppp pppp	1.000	0.262
ppp prrp	1.100	0.298
ppp rppp	1.000	0.262
ppp rrrp	**0.911**	0.307
ppp rrrr	1.265	0.305
rrr rrrp	1.118	0.297
rrr rrrr	0.945	0.279

Table 2: Part of results on different rescale method on different features. 'p' denote using padding, while 'r' denote using the default resize in the corresponding channel of features, in the order used in Sec. . The table is generated with results trained with ViT on Score-1-Reg task. Full table can be found in the appendix.

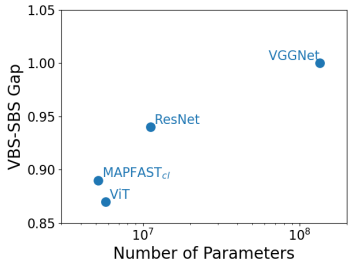

Figure 3: Comparison for VBS-SBS gap of different neural networks for Score-0.001-Reg task regards numbers of parameters.

Ren et al. 2021), researchers are always using interpolation resize (known as resize in ML libraries like Pytorch (Paszke et al. 2019)) to make the images to a fixed size of 227×227. However, resizing the pictures by interpolating the values on every pixel from the original input is losing many underlying assumptions in planning. For example, one of the most important ones is that we can only move one cell at a time, so a cell of 1×1 in a 10×10 map is very different from a group of cells of 10×10 in a 100×100 map. So in this part, we further examine how the performance of a model will be if we change only one channel from interpolation to padding.

Because enumerating all possible combinations of padding and resizing in different channels needs $2^7 = 128$ experiments, which is too large for us to test them all, we assume that channels that have similar meanings should be treated the same, and thus we can change the rescaling method in groups. We show the results of different settings of interpolation (resize) and padding in Table. 2. We found that choosing to use which rescaling method can make a different, and change one channel to another is not independent with what other channels currently are. In our setting, we found that the best rescaling method is to use padding in the first three channels that describe the MAPF instance as start locations, goal locations and obstacles, and the last channels that is the heatmap provide the sum of all possible shortest path. All other channels should use resize. While we do find this really helpful in our experiment, we encourage later user to double check is the same hold in their dataset given that the advantage is not very significant in our experiment.

Discussion on Choosing a Neural Network for Algorithm Selection

While we have found that ViT is the neural network that wins the most time in the experiment section, it is also interesting to see how results change according to the number of parameters it has, which affect both the training time and the inference time. We conclude our findings in Fig. 3. We can find that while we have used the smallest variants of each group of neural networks, ViT is the one that not only has a small number of parameters but also has a good performance. Other modern models can also be successful in tasks while small in the number of parameters, and VGGNet is the only one that is both slow and performing badly.

Conclusion

In this paper, we study algorithm selection for suboptimal MAPF solvers. We formulate algorithm selection as a prediction problem with multiple potential formulations to trade off between runtime and solution cost differently. Deep learning models are trained with many combinations of optimization objectives and loss functions to make predictions. We observed that utilizing changing a combination can greatly enhance performance when training for specific metrics, particularly when training working with modern learning models. We show that different ways of rescaling can benefit different features, and discuss how to choose neural networks for algorithm selection. In addition, we show that hyperparameter selection can be successfully done with the same framework.

References

Bischl, B.; Kerschke, P.; Kotthoff, L.; Lindauer, M.; Malitsky, Y.; Fréchette, A.; Hoos, H. H.; Hutter, F.; Leyton-Brown, K.; Tierney, K.; and Vanschoren, J. 2016. ASlib: A benchmark library for algorithm selection. *Artif. Intell.*, 41–58.

Chen, T.; and Guestrin, C. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the Twenty-second ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 785–794. ACM.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; Uszkoreit, J.; and Houlsby, N. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *Proceedings of the Ninth International Conference on Learning Representations (ICLR)*.

Han, S. D.; and Yu, J. 2020. DDM: Fast Near-Optimal Multi-Robot Path Planning Using Diversified-Path and Optimal Sub-Problem Solution Database Heuristics. *IEEE Robotics Autom. Lett.*, (2): 1350–1357.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778. IEEE Computer Society.

Heins, J.; Bossek, J.; Pohl, J.; Seiler, M.; Trautmann, H.; and Kerschke, P. 2021. On the potential of normalized TSP features for automated algorithm selection. In *Proceedings of the Sixteenth Conference on Foundations of Genetic Algorithms (FOGA)*, 7:1–7:15. ACM.

Huber, P. J. 1992. *Robust Estimation of a Location Parameter*, 492–518. New York, NY: Springer New York. ISBN 978-1-4612-4380-9.

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

Kaduri, O.; Boyarski, E.; and Stern, R. 2021. Experimental Evaluation of Classical Multi Agent Path Finding Algorithms. In *Proceedings of the Fourteenth International Symposium on Combinatorial Search (SOCS)*, 126–130. AAAI Press.

Kerschke, P.; Kotthoff, L.; Bossek, J.; Hoos, H. H.; and Trautmann, H. 2018. Leveraging TSP Solver Complementarity through Machine Learning. *Evol. Comput.*, (4).

Krizhevsky, A.; and Hinton, G. 2009. Learning multiple layers of features from tiny images. Technical report, University of Toronto, Toronto, Ontario.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, 1106–1114.

Laurent, F.; Schneider, M.; Scheller, C.; Watson, J. D.; Li, J.; Chen, Z.; Zheng, Y.; Chan, S.; Makhnev, K.; Svidchenko, O.; Egorov, V.; Ivanov, D.; Shpilman, A.; Spirovska, E.; Tanevski, O.; Nikov, A.; Grunder, R.; Galevski, D.; Mitrovski, J.; Sartoretti, G.; Luo, Z.; Damani, M.; Bhattacharya, N.; Agarwal, S.; Egli, A.; Nygren, E.; and Mohanty, S. P. 2020. Flatland Competition 2020: MAPF and MARL for Efficient Train Coordination on a Grid World. In *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*, 275–301. PMLR.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2021. Anytime Multi-Agent Path Finding via Large Neighborhood Search. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI)*, 4127–4135. ijcai.org.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2022. MAPF-LNS2: Fast Repairing for Multi-Agent Path Finding via Large Neighborhood Search. In *Proceedings of the Thirty-Sixth Conference on Artificial Intelligence (AAAI)*, 10256–10265. AAAI Press.

Ma, H.; Tovey, C. A.; Sharon, G.; Kumar, T. K. S.; and Koenig, S. 2016. Multi-Agent Path Finding with Payload Transfers and the Package-Exchange Robot-Routing Problem. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI)*, 3166–3173. AAAI Press.

Nebel, B. 2020. On the Computational Complexity of Multi-Agent Pathfinding on Directed Graphs. In *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS)*, 212–216. AAAI Press.

Okumura, K.; Machida, M.; Défago, X.; and Tamura, Y. 2022. Priority inheritance with backtracking for iterative multi-agent path finding. *Artif. Intell.*, 103752.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Köpf, A.; Yang, E. Z.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach, H. M.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E. B.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, 8024–8035.

Ren, J.; Sathiyanarayanan, V.; Ewing, E.; Senbaslar, B.; and Ayanian, N. 2021. MAPFAST: A Deep Algorithm Selector for Multi Agent Path Finding using Shortest Path Embeddings. In *Proceedings of the Twentieth International Con-*

ference on Autonomous Agents and Multiagent Systems (AAMAS), 1055–1063. ACM.

Sajid, Q.; Luna, R.; and Bekris, K. E. 2012. Multi-Agent Pathfinding with Simultaneous Execution of Single-Agent Primitives. In *Proceedings of the Fifth International Symposium on Combinatorial Search (SOCS)*, 88–96. AAAI Press.

Seiler, M.; Pohl, J.; Bossek, J.; Kerschke, P.; and Trautmann, H. 2020. Deep Learning as a Competitive Feature-Free Approach for Automated Algorithm Selection on the Traveling Salesperson Problem. In *Proceedings of the Sixteenth International Conference of Parallel Problem Solving from Nature (PPSN)*, Lecture Notes in Computer Science, 48–64. Springer.

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

Sigurdson, D.; Bulitko, V.; Koenig, S.; Hernández, C.; and Yeoh, W. 2019. Automatic Algorithm Selection In Multi-agent Pathfinding. *CoRR*.

Silver, D. 2005. Cooperative Pathfinding. In *Proceedings of the Conference on First Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 117–122. AAAI Press.

Simonyan, K.; and Zisserman, A. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *Proceedings of the Third International Conference on Learning Representations (ICLR)*.

Smith-Miles, K. 2008. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Comput. Surv.*, (1): 6:1–6:25.

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 *Proceedings of the Twelfth Annual Symposium on Combinatorial Search (SoCS)*, 151–159. AAAI Press.

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 *ECAI 2016 - 22nd European Conference on Artificial Intelligence*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, 810–818. IOS Press.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S. E.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1–9. IEEE Computer Society.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 5998–6008.

Wightman, R. 2019. PyTorch Image Models. https://github.com/rwightman/pytorch-image-models.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2012. Evaluating Component Solver Contributions to Portfolio-Based Algorithm Selectors. In *Proceedings of the Fifteenth International Conference of Theory and Applications of Satisfiability Testing (SAT)*, Lecture Notes in Computer Science, 228–241. Springer.

Yang, K.; Qinami, K.; Fei-Fei, L.; Deng, J.; and Russakovsky, O. 2020. Towards fairer datasets: filtering and balancing the distribution of the people subtree in the ImageNet hierarchy. In *Proceedings of ACM Conference on Fairness, Accountability, and Transparency (ACM FAccT)*, 547–558. ACM.

Yu, J.; and LaValle, S. M. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI)*, 1, 1443–1449. AAAI Press.