Algorithm Selection for Optimal Multi-Agent Path Finding via Graph Embedding

Anonymous submission

Abstract

Multi-agent path finding (MAPF) is the problem of finding paths for multiple agents such that they do not collide. Finding optimal solutions to MAPF is NP-Hard, yet modern optimal solvers can scale to hundreds of agents and even thousands in some cases. Different solvers employ different approaches, and there is no single state-of-the-art approach for all problems. Furthermore, there are no clear, provable, guidelines for choosing when each optimal MAPF solver to use. Prior work employed Algorithm Selection (AS) techniques to learn such guidelines from past data. A major challenge when employing AS for choosing an optimal MAPF algorithm is how to encode the given MAPF problem. Prior work either used hand-crafted features, graph embedding of the shortest paths, or an image representation of the problem. Each encoding is lossy, in the sense that it does not capture some aspect of the MAPF problem. We propose a graph encoding of the MAPF problem and show how it can be used on-the-fly with a modern graph embedding algorithm called FEATHER. We also show how this encoding can be effectively joined with existing encodings, resulting in a novel AS method we call MAPF Algorithm selection via Graph embedding (MAG). An extensive experimental evaluation of MAG on several MAPF algorithm selection tasks reveals that it outperforms existing methods significantly.

Introduction

Multi-Agent Path Finding (MAPF) is the problem of finding paths for a group of agents, moving each agent from its initial location to a designated target location. The main constraint in MAPF is that the agents must not collide. MAPF has received significant interest recently in the scientific community and in industry, as it has applications in robotics (Veloso et al. 2015) and automated warehouses (Wurman, D'Andrea, and Mountz 2008). Finding an optimal solution to MAPF with respect to various optimization criteria, is known to be NP Hard (Surynek 2010; Yu and LaValle 2013). Nevertheless, a range of practical algorithms that guarantee optimality exists (Daniel Kornhauser 1984; Surynek 2009). It has been shown that these algorithms are able to find optimal solutions to MAPF problems with more than 100 agents with less than one minute of runtime (Li et al. 2021).

Different optimal MAPF algorithms employ different problem-solving techniques. For example, EPEA* (Goldenberg et al. 2014) employs a heuristic search technique, BCP (Lam et al. 2022) uses optimization techniques, and SAT-MDD (Surynek et al. 2016) compiles MAPF to Boolean Satisfiability. Correspondingly, different solvers work best for different MAPF instances, and no algorithm has emerged to dominate all others. The variance in performance can be great, where some algorithms perform poorly on some instances but significantly outperform all others on other instances. On a recently performed extensive comparison of 5 optimal MAPF algorithms (Kaduri, Boyarski, and Stern 2021), it was shown that even the least effective algorithm had some grids in which it was able to solve instances with 4 times more agents than all other evaluated algorithms. Thus, developing methods for selecting the best optimal MAPF search algorithm for a given instance is a worthwhile endeavor.

The problem of determining which algorithm from a given set of algorithms is expected to perform best on a given problem instance is known as the Algorithm Selection (AS) problem (Rice 1976; Kerschke et al. 2019).[1] Several recent works have developed AS techniques for optimal MAPF (Kaduri, Boyarski, and Stern 2020; Ren et al. 2021; Alkazzi et al. 2022). They used supervised learning to train a classifier that chooses the best optimal MAPF algorithm for a given MAPF instance, from a portfolio of optimal MAPF algorithms that include EPEA* (Goldenberg et al. 2014), ICTS (Sharon et al. 2013), SAT-MDD (Surynek et al. 2016), CBSH (Felner et al. 2018), and Lazy CBS (Gange, Harabor, and Stuckey 2019). A key challenge faced by all prior work on AS for optimal MAPF is how to encode a given MAPF instance, as an input to the supervised learner. Prior works proposed hand-crafted MAPF-related features, casting the MAPF instance as an image (Alkazzi et al. 2022), and graph embedding of a subgraph containing the shortest paths from start to target. Neither of these encodings completely captures the encoded

[1] Technically, this problem is called the per-instance AS problem.

MAPF instance.

To this end, we propose several contributions. The first contribution is a different encoding of the MAPF instance called FG2V, that fully utilizes the power of graph embedding algorithms by encoding the entire graph, augmented with artificial edges marking the start and target vertices of every agent. This embedding is done using FEATHER (Rozemberczki and Sarkar 2020), a modern graph embedding algorithm. The resulting embedding yields superior results in most cases, but not always. The second contribution is a simple method for integrating multiple encodings, which enables a more comprehensive encoding of the problem. This method can be used with different embeddings in a seamless manner. The resulting AS method is called MAG. Our third contribution is a comprehensive evaluation of MAG on a standard benchmark over three different AS tasks, which differ in how similar are the train and test instances. Our results show that MAG is superior to baseline methods, demonstrating the first practical AS method for MAPF that utilizes graph embeddings. All our code and datasets will be made publicly available to the community.

Background

For completeness, we provide here a brief background on MAPF, AS for MAPF, and graph embedding.

MAPF

A MAPF problem instance is defined by a tuple $\langle k, G, s, t\rangle$ where k is the number of agents, $G = (V; E)$ is an undirected graph, $s : [1 \ldots, k] \to V$ maps an agent to its source vertex, and $t : [1, \ldots, k] \to V$ maps an agent to its target vertex. Initially, each agent is in its source vertex. In every time step, each agent either waits in its current vertex or moves to one of the vertices adjacent to it. A single-agent plan for agent i is a sequence of move/wait actions that moves i from $s(i)$ to $t(i)$. A solution to a MAPF instance is a set of single-agent plans, one for each agent, such that the agents do not collide with each other. In the classical MAPF problem, the cost of a solution is either the sum of actions in all single-agent plans or the number of actions in the longest single-agent plan. The former cost function is known as sum-of-costs (SOC) and the latter is known as makespan. We measured solution cost in terms of SOC in our experiments, but most of our contributions are agnostic to the chosen cost function. MAPF extensions that consider non-unit action costs, large agents, and continuous time have been explored (Atzmon et al. 2020; Andreychuk et al. 2022; Li et al. 2019). In this work, we focus on classical MAPF.

Optimal classical MAPF algorithms are classical MAPF algorithms that are guaranteed to return optimal solutions according to a predefined solution cost function. Different optimal algorithms have been proposed for solving classical MAPF problems. Prime examples are EPEA* (Goldenberg et al. 2014), SAT-MDD (Surynek 2010), CBS (Sharon et al. 2015; Felner et al. 2018) and its many extensions, BCP (Lam et al. 2022), Lazy CBS (Gange, Harabor, and Stuckey 2019), and ICTS (Sharon et al. 2013). These algorithms apply a range of techniques: some use heuristic search on dedicated state spaces, other compile the problem to Boolean Satisfiability (SAT) and call an off-the-shelf SAT solver, while others borrow ideas from constraint programming. No algorithm fully dominates the other, and different problems are solved best with different algorithms. This raised the need for an automated way to select which optimal MAPF solver to use for a given problem.

AS for MAPF

Previously proposed AS methods for MAPF followed a standard supervised learning paradigm, and differ mainly in the type of features they extract. Sigurdson et al. (Sigurdson et al. 2019) and Ren et al. (Ren et al. 2021) mapped a given MAPF problem to an image and extracted image-based features with a Convolutional Neural Networks (CNN). Kaduri, Boyarski, and Stern (Kaduri, Boyarski, and Stern 2020) proposed a set of hand-crafted, MAPF-specific features, such as the number of agents divided by the number of unblocked cells in the grid. We refer to these features as the KBS features. Ren et al. also explored the potential of using features that are based on mapping the given MAPF problem to a graph and using a graph embedding method. While their exploration showed that graph embedding can provide useful features for MAPF AS, they do not provide a practical method to use it. In fact, they consider their method to be "not a deployable algorithm selector in any reasonable sense" (Ren et al. 2021), since it could not be used on any MAPF problem not observed during training. We overcome this limitation in our work.

Kaduri et al. (Kaduri, Boyarski, and Stern 2021) distinguished between three types of AS problems for MAPF: (1) in-grid AS, (2) in-grid-type, and (3) between-grid-type. In-grid AS means the train and test problems are all from the same underlying graph. In-grid-type means the train and test problems are on different grids, but their grids have similar topologies. Between-grid-type means that graphs in the train and test problems are completely different, e.g., training on maze-like graphs and testing on graphs that represent road map in a city. Most prior work has focused on the in-grid AS problem, which is, of course, significantly easier.

Graph Embedding Algorithms

Node embedding methods are algorithms for encoding a node in a graph into a low-dimensional continuous vector (Goyal and Ferrara 2018). Similarly, graph embedding methods are algorithms for encoding an entire graph G into a low-dimensional continuous vector (Goyal and Ferrara 2018), referred to as the embedding of G. Ideally, graphs with similar structure will have embedding that are close in terms of their Euclidean distance. Graph embeddings have proven to be useful features for various machine learning tasks such as classification (You et al. 2020).

Graph2Vec (Narayanan et al. 2017) is a neural graph embedding algorithm that accepts a set of graphs and outputs an embedding for each graph by analyzing local neighborhood of their nodes. It runs stochastic gradient descent to optimize a loss function that ensures similar graphs in the input set of graphs will have a similar embedding. Notably, it cannot be effectively used on other graphs without re-training, and thus its usefulness for our purposes is limited. FEATHER (Rozemberczki and Sarkar 2020) is a recently proposed algorithm that serves as both a node embedding and a graph embedding algorithm. Its embedding is based on the likelihood of reaching each node in a random walk. Unlike Graph2Vec, it does not require an optimization step, and can reasonably used to embed a single graph. The embeddings created by FEATHER have shown to be effective in node-level and graph-level machine learning tasks, such as classifying fake users in a social network and link prediction. FEATHER has an additional positive property that it describes isomorphic graphs with the same representation and exhibits robustness to data corruption.

Method

In this section, we describe our AS method for choosing optimal classical MAPF algorithms called MAPF Algorithm selection via Graph embedding (MAG). MAG works as follows. First, it encodes the given MAPF problem as a graph. Then, it creates an embedding of the resulting graph with FEATHER. The resulting vector is added to a set of previously proposed MAPF-specific features extracted from the given MAPF problem, and used to solve our AS problem using supervised machine learning. Next, we describe each of these steps in more detail.

Encoding MAPF as a Graph

We consider two ways to encode a MAPF problem $\Pi = \langle k, G, s, t\rangle$ as a graph. The first encoding method, called G2V, was previously proposed by Ren et al. (Ren et al. 2021). G2V encodes Π by extracting from G that induced subgraph that includes only nodes that are on a shortest path between the source and the target of an agent. In more detail, G2V computes for each agent the shortest path from its source to its target while ignoring all other agents. Then, we construct a graph that contains only the nodes on these shortest paths, adding an edge between any pair of nodes on these shortest paths that have an edge between them in the original MAPF problem. Note that the resulting graph may be disconnected.

The benefit of G2V is that the resulting graphs are significantly smaller than G. However, these graphs lose information: they do not consider regions on G that are not on the agents shortest paths. These regions may be important to consider when the corresponding MAPF problem is particularly difficult and agents must move away from their shortest paths. To address this limitation of G2V we propose FullG2V (FG2V), which uses the entire graph G to encode the given MAPF problem. To include details about the MAPF problem beyond G, the graph outputted by FG2V includes an artificial edge for every agent between its source and target. Formally, for a MAPF problem $\langle k, G = (V, E), s, t\rangle$ FG2V outputs the graph $G' = (V', E')$ where $V' = V$ and $E' = E \cup \{(s(i), t(i))\}_{i=1}^{k}$. Figure 1 illustrates G2V and FG2V on a simple MAPF problem on a 4-neighborhood grid.

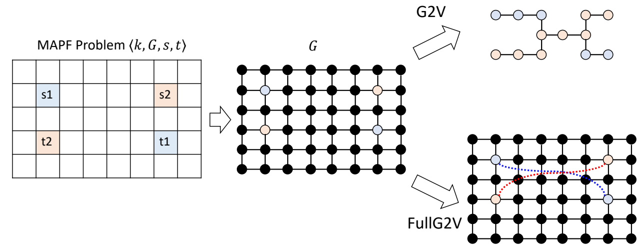

Figure 1: An example of the G2V and FG2V encoding methods.

Embedding the Graph

To embed the encoded graph into a vector space, we used the FEATHER algorithm (Rozemberczki and Sarkar 2020). FEATHER creates a graph embedding by first embedding each of the graph nodes and then pooling the resulting vectors to a single vector of size 500. Unlike other graph embedding techniques, it does not require a-priori training. Consequently, the features extracted using FEATHER can be extracted and used in a meaningful way even for graphs created for MAPF problems that are not in the training set. This is key to allowing graph embedding features to be used for optimal MAPF AS methods.

Note that the default pooling method in FEATHER is "mean". This means the graph embedding is created by taking the mean over its constituent node embeddings. We observed that using mean pooling did not perform well in our context, i.e., led to poor classification results when used as features. The reason for this is that mean pooling diminishes the impact of the artificial edges added between the source and target of each agent. Thus, MAPF problems on the same graph yielded very similar embeddings. To overcome this, we configured FEATHER to use "max" pooling, which emphasizes small differences between graphs created from MAPF problems on the same grid. This yielded significantly better results when training the AS classification model.

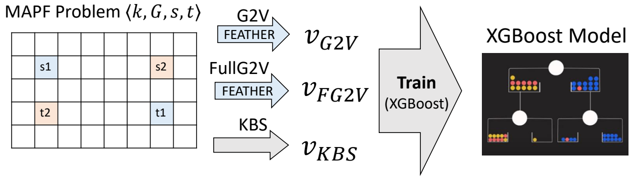

Figure 2: Diagram of the MAG feature extraction and training process.

Feature Extraction and Learning

For a given MAPF problem Π, MAG uses G2V and FG2V to create two graphs G_{G2V} and G_{FG2V} that encoding Π. Then, it creates two graph embeddings v_{G2V} and v_{FG2V} by applying FEATHER on these graphs. It also creates a vector v_{KBS} by extracting all the MAPF-specific features proposed by Kaduri et al. (Kaduri, Boyarski, and Stern 2020). Finally, MAG concatenates v_{G2V}, v_{FG2V}, and v_{KBS}. The resulting vector is the features MAG uses for training. The training process itself is a straightforward multi-class supervised learning process, where every instance is a MAPF problem instance and the label is the fastest algorithm for that instance within our portfolio of algorithms. Figure 2 illustrates the feature extraction and training process. Note that more sophisticated approaches to AS exists in other domains (Kerschke et al. 2019), e.g., methods that involve runtime predictions. This is left for future work.

Experimental Results

In this section, we present an experimental evaluation of MAG on a standard publicly available grid-based MAPF benchmark (Stern et al. 2019). This benchmark contains 33 grids arranged into seven grid types: video games (denoted as "game" grids), city maps ("city"), maze-like grids ("maze"), grids arranged as rooms with narrow doors between them ("room"), open grids ("empty"), open grids with randomly placed obstacles ("random"), and grids that are inspired by the structure of warehouses ("warehouse"). Figure 3 shows an example grid from each type.[2] The benchmark includes scenario files for each grid. Each scenario file contains source and target locations for as many as 1,000 agents.[3] The scenario files of each grid are grouped into two sets. In the first set of scenario files, denoted Random, the agents source and target locations are located purely randomly. In the second set of scenario files, denoted Even, the agents' source and target locations are evenly distributed in buckets of 10 agents according to the distance between each agent's start and target. Only the scenarios from the Even set were used, as they represent a more diverse set of MAPF problems.

In total, the training sets in our experiments included an average 3,000 instances for each map. The entire training process (over all training set instances) required approximately 10 12 hours, including feature creation, training, and hyperparameter tuning. The runtime required for feature extraction of a single instance was a single second on average, mostly devoted to running FEATHER. The prediction (inference) runtime of our XGBoost model is also extremely fast, since it is not a large deep NN. All experiments were run on a server with an Intel 12th Gen i7-12850HX 2.10 GHz CPU, 64.0 GB RAM, and a 64-bit OS. We are fully committed to making the code, datasets, and results publicly available once the paper is accepted.

Experimental Setup

We performed three sets of experiments, one for each of the AS problem setups defined by Kaduri et al. (Kaduri, Boyarski, and Stern 2021): in-grid, in-grid-type, and between-grid-type. To train and evaluate MAG in each setup, we used the publicly available dataset of Kaduri et al. (2021), which includes results for running a set of optimal MAPF solvers of the entire MAPF benchmark mentioned above. Specifically, results for the following optimal MAPF solvers are available in this dataset: ICTS (Sharon et al. 2013), EPEA* (Goldenberg et al. 2014), SAT-MDD (Surynek et al. 2016), CBSH (Felner et al. 2018), and Lazy CBS (Gange, Harabor, and Stuckey 2019).

Baselines We compared MAG against two baselines:

- KBS. The AS method by Kaduri, Boyarski and Stern (Kaduri, Boyarski, and Stern 2020), which uses only their hand crafted features.
- G2V. The AS method described by Ren et al.(Ren et al. 2021), which uses only the G2V encoding. To extract features from the G2V graph, we used the FEATHER graph embedding.[4]

We do not compare against baselines that always choose the same algorithm, as Kaduri et al. (2019) already established that such baselines yield poorer results compared to KBS Prior work already established that AS with the KBS features yielded better results In addition, we performed an ablation study for MAG, and report on results for AS methods that use different subsets of features. Namely, FG2V + G2V KBS + FG2V KBS + G2V and FG2V. Note that KBS + G2V + FG2V is exactly MAG. For training, we used XGBoost (Chen and Guestrin 2016), a well-known supervised learning algorithm. Preliminary experiments with other learning algorithms, such as Logistic Regression and Random Forest, yielded weaker results. The hyper

[2]The images were taken from the MovingAI repository (Sturtevant 2012), which hosts the grid MAPF benchmark we used (Stern et al. 2019).

[3]Every scenario file contains 1,000 pairs of start and target locations, except for grids that are too small to occupy that many agents.

[4]This differs from Ren et al., who used Graph2Vec. As explained earlier, Graph2Vec is not a practical method for our problem, since it requires knowing a-priori the graphs to embed.

Figure 3: An example grid from each of the grid types in our benchmark, From left to right: empty, random, warehouse, game, city, maze, and room.

parameters of XGBoost were tuned by performing a 4-fold cross validation over the training set, for each AS setup.

Metrics The main metrics used in prior work on AS for MAPF are:

- Accuracy (Acc). Ratio of instances where the AS method returned the fastest MAPF algorithm.
- Coverage (Cov). Ratio of MAPF instances solved under a time limit of 5 minutes.[5]
- Runtime (RT). Average run-time in minutes to solve a single MAPF instance with the selected MAPF solver.[6]

Note that since all solvers are optimal MAPF solvers, all solvers return solutions of exactly the same cost. Thus, comparing solution quality is redundant.

To provide context for our results, we also report on the results of an Oracle, which always selects the fastest algorithm for every MAPF instance. No practical AS method can perform better than Oracle, which has accuracy and coverage of 1.0, and the smallest possible runtime. A final metric, we report for every AS method the average percentage of Oracle runtime required for the selected MAPF solver beyond the runtime required by the algorithm selected by Oracle. We call this metric the average regret, or briefly $\%Rg$. The average regret of Oracle is by definition zero, and better AS methods will have lower average regret values.

In-Grid Results

Table 1 presents the results for the in-grid AS setup experiments. The rows correspond to different AS methods, and the columns are the metrics defined earlier. The column groups "All" and "Avg" provide a slightly different way to aggregate the results over all test problems. The results under the "All" columns are averages over all test problems, regardless of their grids and grid types. This is how most prior work on AS for MAPF aggregated their results. The limitation of this aggregation is that some grids in the benchmark are smaller than others, and has fewer MAPF problems defined for them. The results under the "Avg" columns are averages of averages, where the results of each grid type are averaged separately and only the resulting averages are averaged. This mitigates unwanted to bias stemming from the number of problems in each grid type.

	All			Avg			
Metric	Acc	Cov	RT	Acc	Cov	RT	%Rg
KBS	0.83	0.98	0.549	0.88	0.99	0.41	12.4
G2V	0.81	0.97	0.589	0.87	0.98	0.45	25.4
MAG	0.85	0.99	0.514	0.89	0.99	0.40	9.0
Oracle	1.00	1.00	0.436	1.00	1.00	0.36	0.0
FG2V +G2V	0.84	0.98	0.531	0.89	0.99	0.41	11.5
KBS +G2V	0.84	0.98	0.532	0.89	0.99	0.40	9.5
KBS +FG2V	0.85	0.99	0.513	0.89	0.99	0.39	8.1
FG2V	0.84	0.98	0.534	0.88	0.99	0.41	11.8

Table 1: Results for the In-Grid AS setup, averaged over all test problems.

Consider first the results for MAG and our two main baselines, KBS and G2V. For each metric, we highlighted the best results among these AS methods in bold. As the results clearly show, MAG is either on par or better than these baselines on all metrics. For example, the accuracy of MAG is 0.85 in "All" while it is 0.83 and 0.81 for KBS and G2V respectively. The advantage of MAG over G2V is more significant, and more modest compared to KBS Still advantage is significant, especially in terms of the average regret, which is approximately 25% smaller than KBS (9.0 vs. 12.4).

The results of our ablation study are given in the shaded rows of Table 1. The results are very similar to MAG where there is a slight advantage for using the KBS hand-crafted MAPF-specific features together with graph embedding features (either KBS + G2V or KBS + FG2V). This is expected, as more diverse set of features is expected to be more beneficial. That being said, in some cases some of the algorithm configurations in our ablation study performed as well and even slightly better than the full MAG Automated methods for algorithm configuration can potentially be used to identify the optimal configuration for a given problem instance. This is beyond the scope of this work.

[5]This time limit is common in the Optimal MAPF literature.

[6]We considered cases where the selected MAPF solver could not solve the problem within our 5-minute time limit as having a runtime of 5 minutes. The same has been done in prior work on MAPF AS (Kaduri, Boyarski, and Stern 2020; Ren et al. 2021).

Metric	In-Grid-Type Acc	Cov	RT	% Rg	Between-Grid Acc	Cov	RT	% Rg
KBS	0.69	0.93	0.86	78.6	0.63	0.87	0.99	223.1
G2V	0.67	0.92	0.91	91.8	0.61	0.86	1.04	220.7
MAG	0.71	0.94	0.80	67.9	0.63	0.89	0.93	180.0
Oracle	1.00	1.00	0.48	0.00	1.00	1.00	0.36	0.00
FG2V +G2V	0.69	0.93	0.84	75.8	0.62	0.88	0.97	195.8
KBS +FG2V	0.70	0.94	0.81	70.5	0.64	0.89	0.93	192.5
KBS +G2V	0.70	0.94	0.82	71.6	0.65	0.88	0.94	181.8
FG2V	0.66	0.91	0.90	87.9	0.62	0.88	0.95	176.0

Table 2: In-grid-type (left) and Between-grid-type (right) results.

In-Grid-Type and Between-Grid-Type Results

The first 5 columns (left-to-right) in Table 2 presents the results for the in-grid-type AS setup experiments. Similar to Table 1, the rows are different AS methods and the columns are different metrics, corresponding to the "Avg" column family. We highlighted in bold the AS method, among MAG and our two baselines, that yielded the best results in each metric. As in the in-grid experiments, the advantage of MAG over the baselines is clear in all metrics. For example, its average regret is 67.9 while it is 78.60 and 91.80 for KBS and G2V respectively.

The ablation study results show that here too, combining the hand-crafted features of KBS with either type of graph embedding provides the biggest performance improvement. For example, KBS with either G2V or FG2V yields 0.70 accuracy and runtime of 0.82 or less while FG2V alone or even with G2V yielded lower accuracy and a higher runtime. It is worth comparing the average regret results here and in the in-grid experiments. While the regret of MAG here is 67.90 it is only 9.0 in the in-grid results. This highlights that in-grid-type AS is a significantly harder task the in-grid AS, since it requires generalizing from different grids (although from the same type).

The rightmost 4 columns in Table 2 presents the results for the between-grid-type AS setup experiments. The first trend we observe is that the overall results for all algorithms is significantly worse compared to all other AS setups (in-grid and in-grid-type). For example, the average regret of MAG in the in-grid-type results is 67.9 but it is 180 in between-grid results. Similarly, MAG accuracy dropped from 0.71 to 0.63. Recall that the regret and accuracy of MAG in the in-grid was 9.0 and 0.89, respectively. These differences are expected, since in the between-grid experiments, the training set did not include any grid of the tested type, which makes the classification problem significantly harder.

In terms of the comparison with our baselines, the general trend we observed so far continues in the between-grid setup: MAG is either on par or better than the baselines in all metrics. For example, its average runtime and regret are 0.93 and 180 while it is 0.99 and 220.7 for the next best baseline, respectively. The ablation study results are less conclusive in this setup. In terms of accuracy, we still see the benefit in combining KBS features with graph embedding features over only using graph embedding features. However, the lowest average regret is achieved when only using MAG. In fact, some subsets of MAG features actually outperform MAG on some metrics. For example, using only FG2V yields lower accuracy than the full MAG but a slightly lower average regret (176 vs. 180). However, these differences are relatively small.

Grid Types Analysis

Table 3 provides a deeper insight into the results of our in-grid, in-grid-type, and between-grid experiments. Here, the results — accuracy, coverage, and regret — are grouped by grid types. For example, the results in the row "City" show the average results over test problems that are on grids of type "City". The rows correspond to the test grid type, and the columns correspond to the AS setup and evaluated algorithm. While MAG is still, in general, the best-performing algorithm, in some grid types and metrics it is not, especially in the between grid setup. This is most evident in the results for Maze grids, where the accuracy, coverage, and regret of KBS— 0.56, 0.71, and 398.8, respectively — are better than the corresponding results of MAG which were 0.47, 0.66, and 483.6. A possible explanation for this is that MAPF solutions in Maze grids are significantly different from MAPF solutions in grids from other grid types. Maze grids have many narrow corridors, and consequently avoiding conflicts between agents often requires one agent to make a large detour.

Interestingly, MAG performs well on City grids in the in-grid AS setup. This suggests it is able to learn how to act in such grids, if they are given to it for training. The in-grid results for Maze grids provide another interesting insight when comparing the results of MAG and G2V. As can be seen, the difference in this grid type between MAG and G2V is most pronounced: the accuracy, coverage, and regret of G2V are 0.79, 0.93, and 119.3, respectively, while they are 0.86, 0.99, and 20.6 for MAG. The poor performance of G2V in Maze grids is understandable: in such grids agents often follow paths that are significantly different from their shortest paths to avoid collisions. Such paths are not encoded by G2V. In contrast, we observed FG2V works quite well on these grids, almost as well as MAG.[7]

Feature Importance

Figure 4 plots a feature importance analysis performed on the prediction models created by MAG for each of the AS setups. The importance of the different features was measured by the coefficient learned for it by the learning algorithm. The features are listed on the x-axis, where the first 20 features are the KBS features, the next 500 features are the G2V features, and the last 500 features are the FG2V features. Within each feature family (KBS,G2V, and FG2V), the features are

[7]These results are not shown in the paper.

Grid-type	AS Setup Metric	In-Grid KBS	G2V	MAG	In-Grid-Type KBS	G2V	MAG	Between-Grid KBS	G2V	MAG
Empty	Acc	0.88	0.89	0.90	0.67	0.68	0.71	0.81	0.78	0.82
	Cov	1.00	1.00	1.00	0.97	1.00	0.99	1.00	0.99	1.00
	%Rg	0.9	0.1	0.8	3.5	1.0	1.8	9.5	17.7	2.5
Random	Acc	0.91	0.90	0.91	0.84	0.79	0.82	0.78	0.71	0.73
	Cov	1.00	1.00	1.00	1.00	0.99	1.00	1.00	0.99	1.00
	%Rg	3.1	4.1	2.4	1.4	2.3	1.1	7.2	54.8	4.0
Warehouse	Acc	0.84	0.84	0.86	0.64	0.64	0.67	0.67	0.62	0.68
	Cov	0.99	0.99	1.00	0.94	0.94	0.96	0.96	0.88	0.98
	%Rg	7.9	9.4	4.7	1.4	1.6	1.3	32.9	78.1	27.3
Game	Acc	0.91	0.91	0.92	0.77	0.72	0.78	0.74	0.72	0.65
	Cov	0.98	0.98	0.99	0.92	0.86	0.92	0.88	0.89	0.89
	%Rg	19.2	21.0	18.0	1.4	2.3	1.7	122.0	113.1	122.0
City	Acc	0.90	0.89	0.92	0.58	0.57	0.61	0.53	0.53	0.58
	Cov	0.99	0.99	0.99	0.81	0.83	0.83	0.90	0.86	0.88
	%Rg	10.1	12.4	7.1	3.1	3.0	2.9	136.2	155.6	146.3
Maze	Acc	0.85	0.79	0.86	0.40	0.44	0.50	0.56	0.45	0.47
	Cov	0.98	0.93	0.99	0.66	0.62	0.66	0.71	0.64	0.66
	%Rg	31.9	119.3	20.6	8.2	9.6	8.4	398.8	511.4	483.6
Room	Acc	0.93	0.90	0.92	0.63	0.50	0.66	0.32	0.43	0.49
	Cov	1.00	1.00	1.00	0.91	0.83	0.94	0.65	0.75	0.81
	%Rg	0.2	2.1	0.1	2.9	5.0	2.2	855.6	614.5	475.0

Table 3: Results for all AS setups, grouped by test grid type.

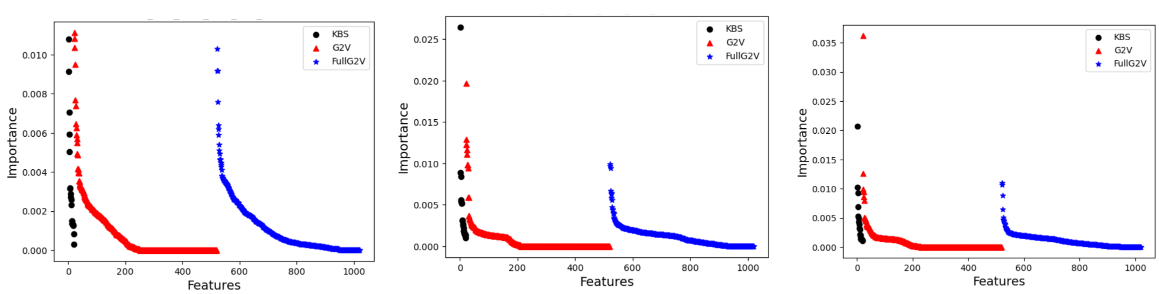

Figure 4: Feature importance for the XGBoost model created for MAG in the in-grid (left), in-grid-type (middle), and between-grid (right) setups.

which corresponds to the successful results of MAG. Another observation is that each feature family includes features applying feature selection method based on weights may be effective. When comparing the different AS setups, it may suggest that additional types of features may be

We proposed MAG, the first practical approach to optiding. MAG uses two encodings of the MAPF problem diate vicinity. To work efficiently on new MAPF problems, MAG utilizes a modern graph embedding algorithm that does not need a-priori training. MAG also uses hand-crafted MAPF-specific features, as suggested by prior work. The combination of graph-embedding features and hand-crafted features leads to strong state of the art AS for optimal MAPF. In an extensive set that MAG significantly outperforms existing baselines almost always. Our results also highlight that and can be the focus of future work. Another impor MAPFASTER (Alkazzi et al. 2022).

References

Alkazzi, J.-M.; Rizk, A.; Salomon, M.; and Makhoul, A. 2022. MAPFASTER: A Faster and Simpler take on Multi-Agent Path Finding Algorithm Selection. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 10088–10093.

Andreychuk, A.; Yakovlev, K.; Surynek, P.; Atzmon, D.; and Stern, R. 2022. Multi-agent pathfinding with continuous time. Artificial Intelligence.

Atzmon, D.; Zax, Y.; Kivity, E.; Avitan, L.; Morag, J.; and Felner, A. 2020. Generalizing multi-agent path finding for heterogeneous agents. In International Symposium on Combinatorial Search (SoCS), volume 11, 101–105.

Chen, T.; and Guestrin, C. 2016. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, 785–794.

Daniel Kornhauser, P. S., Gary Miller. 1984. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In FOCS, 241–250.

Felner, A.; Li, J.; Boyarski, E.; Ma, H.; Cohen, L.; Kumar, T. S.; and Koenig, S. 2018. Adding heuristics to conflict-based search for multi-agent path finding. In International Conference on Automated Planning and Scheduling, volume 28, 83–87.

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

Goldenberg, M.; Felner, A.; Stern, R.; Sharon, G.; Sturtevant, N. R.; Holte, R. C.; and Schaeffer, J. 2014. Enhanced Partial Expansion A*. J. Artif. Intell. Res., 50: 141–187.

Goyal, P.; and Ferrara, E. 2018. Graph embedding techniques, applications, and performance: A survey. Knowledge-Based Systems, 151: 78–94.

Kaduri, O.; Boyarski, E.; and Stern, R. 2020. Algorithm selection for optimal multi-agent pathfinding. In Proceedings of the International Conference on Automated Planning and Scheduling, volume 30, 161–165.

Kaduri, O.; Boyarski, E.; and Stern, R. 2021. Experimental evaluation of classical multi agent path finding algorithms. In Proceedings of the International Symposium on Combinatorial Search, volume 12, 126–130.

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

Lam, E.; Le Bodic, P.; Harabor, D.; and Stuckey, P. J. 2022. Branch-and-cut-and-price for multi-agent path finding. Computers & Operations Research, 144: 105809.

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; Gange, G.; and Koenig, S. 2021. Pairwise symmetry reasoning for multi-agent path finding search. Artificial Intelligence, 301: 103574.

Li, J.; Surynek, P.; Felner, A.; Ma, H.; Kumar, T. S.; and Koenig, S. 2019. Multi-agent path finding for large agents. In AAAI Conference on Artificial Intelligence, volume 33, 7627–7634.

Narayanan, A.; Chandramohan, M.; Venkatesan, R.; Chen, L.; Liu, Y.; and Jaiswal, S. 2017. Graph2Vec: Learning distributed representations of graphs. arXiv:1707.05005.

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 Dignum, F.; Lomuscio, A.; Endriss, U.; and Nowé, A., eds., International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 1055–1063.

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

Rozemberczki, B.; and Sarkar, R. 2020. Characteristic functions on graphs: Birds of a feather, from statistical descriptors to parametric models. In ACM international conference on information & knowledge management, 1325–1334.

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

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

Sigurdson, D.; Bulitko, V.; Koenig, S.; Hernandez, C.; and Yeoh, W. 2019. Automatic algorithm selection in multi-agent pathfinding. arXiv:1906.03992.

Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T.; et al. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In International Symposium on Combinatorial Search, volume 10, 151–158.

Sturtevant, N. 2012. Benchmarks for Grid-Based Pathfinding. Transactions on Computational Intelligence and AI in Games, 4(2): 144 – 148.

Surynek, P. 2009. A novel approach to path planning for multiple robots in bi-connected graphs. In IEEE International Conference on Robotics and Automation, 3613–3619.

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

Surynek, P.; Felner, A.; Stern, R.; and Boyarski, E. 2016. Efficient SAT approach to multi-agent path finding under the sum of costs objective. In European conference on artificial intelligence, 810–818.

Veloso, M. M.; Biswas, J.; Coltin, B.; and Rosenthal, S. 2015. CoBots: Robust Symbiotic Autonomous Mobile Service Robots. In International Joint Conference on Artificial Intelligence (IJCAI).

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

You, R.; Guo, Z.; Cui, L.; Long, X.; Bao, Y.; and Wen, S. 2020. Cross-modality attention with semantic graph embedding for multi-label classification. In AAAI conference on artificial intelligence, volume 34, 12709–12716.

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