Betweenness Centrality in Multi-Agent Path Finding

Eric Ewing
University of Southern California
Los Angeles, CA, United States
ericewin@usc.edu

Jingyao Ren
University of Southern California
Los Angeles, CA, United States
jingyaor@usc.edu

Dhvani Kansara
University of Southern California
Los Angeles, CA, United States
dkansara@usc.edu

Vikraman Sathiyanarayanan
University of Southern California
Los Angeles, CA, United States
sathiyan@usc.edu

Nora Ayanian
University of Southern California
Los Angeles, CA, United States
ayanian@usc.edu

ABSTRACT

Multi-Agent Path Finding (MAPF) is a well studied problem with many existing optimal algorithms capable of solving a wide variety of instances, each with its own strengths and weaknesses. While for some instances the fastest algorithm can be easily determined, not enough is known about their performance to predict the fastest algorithm for every MAPF instance, or what makes some instances more difficult than others. There is no clear answer for which features dominate the hardness of MAPF instances. In this work, we study how betweenness centrality affects the empirical difficulty of MAPF instances. To that end, we benchmark the largest and most complete optimal MAPF algorithm portfolio to date. We analyze the algorithms' performance independently and as part of the portfolio, and discuss how betweenness centrality can be used to improve estimations of algorithm performance on a given instance of MAPF.

KEYWORDS

Multi-Agent Path Finding; Algorithm Portfolio; Betweenness Centrality

ACM Reference Format:
Eric Ewing, Jingyao Ren, Dhvani Kansara, Vikraman Sathiyanarayanan, and Nora Ayanian. 2022. Betweenness Centrality in Multi-Agent Path Finding. In *Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), Online, May 9–13, 2022*, IFAAMAS, 9 pages.

1 INTRODUCTION

Path finding, the problem of finding a feasible path from a start position to a goal position, is a fundamental problem in many real world applications, including many applications in robotic navigation and autonomy. Path finding becomes significantly more challenging when multiple agents must plan paths together and cooperate with other agents in their environment. In autonomous warehouses, for instance, where hundreds of robots may be navigating the same space, finding collision-free paths becomes a significant challenge. In this work we study the discrete Multi-Agent Path Finding (MAPF) problem, which is the computational foundation of many coordination tasks. Formally, an instance of MAPF is a 3-tuple (G, S, T), where $G = (V, E)$ is a connected graph, $S = \{s_1, ..., s_n\}$ is a set of start vertices, and $T = \{t_1, ..., t_n\}$ is a set of goal vertices for n agents. When G is a 2-D grid graph it will generally be referred to as a map. At every discrete timestep, an agent may traverse an edge or wait at its current position. A solution to a MAPF instance is a set of paths in G from s_i to t_i for every agent i. A MAPF solution requires that no two paths collide with each other, meaning agents do not occupy the same node nor traverse the same edge at the same timestep. MAPF algorithms typically optimize either for makespan, the time until the final agent reaches its goal, or sum-of-costs, the sum of the number of time steps for each agent to reach its goal. In this work we focus on algorithms that find optimal paths for the sum-of-costs objective, which is NP-Hard on general graphs as well as 2-D grids [1, 32].

Since MAPF is such a fundamental problem for many real world applications, the research community has produced a number of optimal algorithms that are able to efficiently solve many MAPF instances. When new algorithms are introduced, they are typically compared against existing algorithms on a dataset of maps and random instances. Algorithms are primarily compared based on their overall performance, measured by the completion rate within some cutoff time for every algorithm. While these comparisons give us some idea of how these algorithms perform as standalone solvers, they miss the forest for the trees: they overlook how the algorithms might work and perform differently on the dataset. For this, it is best to consider how the individual algorithms (the trees) affect the solvable space of MAPF problems (the forest). To effectively evaluate a new MAPF algorithm, its performance should be evaluated within a portfolio of existing MAPF algorithms rather than by its overall individual performance.

In this work, we study the performance of a portfolio of algorithms: how the strengths and weaknesses of algorithms complement each other, and what properties of MAPF instances affect the performance of our portfolio. We use a portfolio of six state-of-the-art algorithms, benchmarked on a dataset of over 20,000 diverse MAPF instances.

One of the major factors contributing to MAPF instance difficulty for algorithms in our portfolio is the number of potential collisions found between agents. Instances with no potential conflicts between agents are as simple as generating shortest paths for each agent to its goal. However, when a large number of agents are all trying to traverse the same set of nodes, planning collision-free paths is challenging. The graph centrality measure *betweenness centrality* can help us predict the number of inter-agent conflicts. The betweenness centrality of a node $u \in V$ is the fraction of all

Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), P. Faliszewski, V. Mascardi, C. Pelachaud, M.E. Taylor (eds.), May 9–13, 2022, Online. © 2022 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

possible shortest paths in G that pass through u. Intuitively, betweenness centrality is high in choke points that separate large components of G. Betweenness centrality has previously been applied to a variety of network applications, including identifying key individuals in social networks and measuring the interdisciplinary level of academic journals using citation networks [7, 17]. Betweenness centrality can be used to estimate the difficulty of MAPF maps since it measures the probability that a shortest path will traverse a node. Betweenness centrality can therefore be used to calculate the probability that any pair of agents in a randomly generated MAPF instance will traverse a node and potentially collide at that point. Since it is computed using only G, it does not depend on a particular instance of agent start and goal locations, therefore it can be reused for multiple instances on the same map.

The main contribution of this work is the theoretical and empirical analysis of the relationship between betweenness centrality and difficulty of a MAPF instance. We find that as the average betweenness centrality of a graph increases, the difficulty of a random instance on the graph also increases for our portfolio. To measure this, we first benchmark six of the strongest optimal MAPF algorithms on an extensive and diverse set of MAPF instances. We provide metrics for the performance of each algorithm as an individual and the impact it had on the performance of our overall portfolio. Importantly, we show that MAPF algorithms need not be the overall best algorithm to contribute to portfolio performance. We then compare the performance of our portfolio on maps using betweenness centrality, and show that maps with higher betweenness centrality are harder for our portfolio overall. By investigating what instance properties and map properties affect the performance of our algorithms, we may be able to design MAPF environments that are better suited for the algorithms in our portfolio. Environments with lower betweenness centrality will, in general, be better suited for MAPF.

2 RELATED WORK

Previous work on betweenness centrality has identified the connection between betweenness centrality and congestion in networks. Much attention has been paid to the connection between betweenness centrality and road network congestion [8, 13, 14]. However, results have also shown that betweenness centrality is actually not a good predictor of congestion within road networks [6]. Optimal MAPF differs from road network congestion, in part, because agents in road networks make decisions distributively, while centralized algorithms are used for optimal MAPF. Humans can make boundedly-rational decisions, whereas agents in optimal MAPF problems always attempt to take the shortest path unless other agents prohibit it. We seek not only to measure congestion in our MAPF instances, but also how that congestion affects the performance of MAPF algorithms.

There is currently no simple set of rules to predict which MAPF algorithm will be most effective on a given instance or whether an algorithm will even complete that instance. Several algorithm selectors have been developed using machine learning techniques to accurately predict the fastest algorithm from a pre-defined algorithm portfolio given a MAPF instance. The first efforts at algorithm selection for MAPF were made by Sigurdson et al. [24] by treating it as an image classification task using a version of AlexNet [15]. The MAPF instance is encoded as an RGB image with agents, goals, and obstacles being different colors. Kaduri et al. [11] improved performance with two new models: one based on VGGNet [25] and a tree-based learning model using XGBoost [4]. The XGBoost-based approach is trained on hand-crafted MAPF features, which is considered to be more informative than RGB images, and outperforms the deep learning based approach. Ren et al.'s MAPFAST [22] solved some of the earlier problems of deep learning approaches by including optimal paths in the instance encoding. This provided an efficient link between agents and their goals and outperformed both the previous XGBoost and CNN based methods.

However, there is no clear answer for which features dominate the empirical hardness of MAPF instances. Although the total number of agents plays an important role, the distribution of agents and the map topology also affect the actual instance hardness dramatically [22]. An effective way to compare the expected hardness of different maps is needed to provide guidance for various applications such as warehouse design and MAPF in re-configurable environments. Previous research has shown it is possible to use instance features to reliably estimate the difficulty of hard combinatorial optimization problems, including boolean satasfiability and the travelling salesman problem [9]. In this work, we demonstrate that betweenness centrality is a useful metric to compare maps and can provide a way to compare the hardness of different maps. Additionally, we show that including a measure of betweenness centrality in an existing algorithm selector can boost performance.

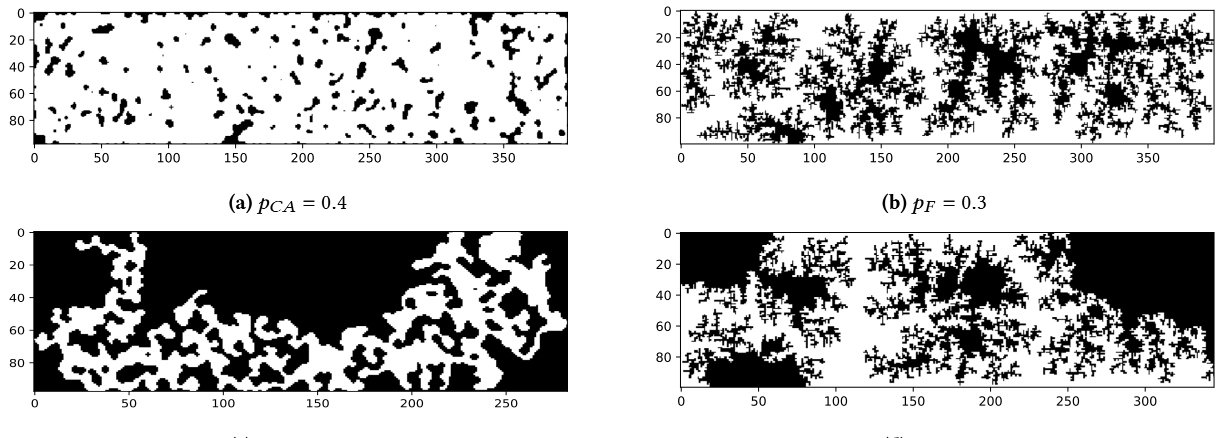
3 MAPF DATASET

Existing MAPF benchmarks have tended towards fewer maps, with more instances on each map. In this work, we seek to characterize and compare performance on a large number of maps. The larger and more diverse dataset of maps helps to find inter-map trends in performance. We therefore generated MAPF instances for our benchmarking with the goal of testing algorithms on as wide a variety of maps as possible, in terms of shape, size, and obstacle distributions.

3.1 Maps

We include new procedurally generated random maps not found in Stern et al.'s standard MAPF benchmark dataset [27] to have more diversity in the size, shape, and obstacle densities of our maps. We introduce two new methods of generating random maps for MAPF instances: a cellular automata method, and a fractal method based on diffusion-limited aggregation. Compared with Stern et al.'s dataset, we include more *medium* sized maps, as well as more oblong maps. Oblong shaped maps typically create more congestion than similarly sized square counterparts without increasing the total number of vertices in the environment. For example, a 4×25 map will likely cause more inter-agent conflicts than a 10×10 map.

The first new map generation method is based on a cellular automata (CA) technique [10] used in games to procedurally generate random caves. This method initially seeds the environment with obstacles uniformly at random with density p_{CA}. Then, a cellular automata rule is applied as follows: if a cell is adjacent to five obstacles, it becomes or remains an obstacle, otherwise it becomes



(a) $p_{CA} = 0.4$ **(b)** $p_F = 0.3$ **(c)** $p_{CA} = 0.5$ **(d)** $p_F = 0.5$

Figure 1: Selected examples of procedurally generated maps in our dataset. Fig. 1a and 1b are maps generated using CA, and seeded with initial densities $p_{CA} = 0.4$ and 0.5 respectively to create different final map characteristics. Fig. 1b and 1d are maps generated using fractal deposition methods until obstacle densities of $p_F = 0.3$ and 0.5 were reached respectively.

free space. We repeat this process for five iterations, with each iteration "smoothing" out the map. These environments are typically characterized as open *cave-like* and can range from largely open environments with smooth walls to maps with small chambers connected by small tunnels, depending on the initial seeding. Figures 1a and 1c show two CA maps resulting from different initial seed densities p_{CA}.

The second new map generation method uses the fractal modeling technique named diffusion limited aggregation [31]. This technique simulates physical processes common in crystal formation and the growth of dendritic structures. The method first places one or more initial seed obstacles randomly on an empty map. Then, a random walk is performed from a random initial open cell until the walk reaches a cell adjacent to an obstacle; this obstacle-adjacent cell is then turned into an obstacle and a new random walk is started. This process is repeated until a set fraction of the map, $0 \leq p_F < 1$, is filled with obstacles. Fractal maps have many more potential choke points with small hallways. Figures 1b and 1d show two different examples of fractal maps in our dataset.

In addition to these new styles of maps, we also include 43 maps from Stern et al.'s benchmarking maps [27]. We have included maps modeling automated warehouse environments, maps of real cities, maps from video games, and maps with uniform random obstacles. In total, 128 maps are included in our dataset. For every map, we perform two postprocessing steps. First, we find all strongly connected components (SCCs) in the map and make every SCC other than the largest into obstacles. This is to ensure that agents have a path to their goal and because a MAPF problem across multiple SCCs can be solved as separate MAPF problems in each SCC. For example, in Fig. 1c, the top middle section had additional freespace that was disjoint from the largest SCC and therefore removed from the map. The second postprocessing step is to remove any full rows or columns of obstacles, thereby trimming the map. Removing excess obstacles is important for accurately calculating map metrics, such as obstacle density.

3.2 Agent and Goal Distributions

The two most common means of generating start and goal locations for MAPF benchmarking instances are uniform random and clustered distributions. Clustered distributions broadly refer to distributions that group start and goal locations together or in other ways attempt to make instances more difficult for algorithms. Stern et al. [27] give a more thorough list of methods to cluster start and goals. For our dataset, we use uniform random, Gaussian, and multi-modal Gaussian distributions of start and goal vertices. For each map, we generate ten instances for each "agent tier", $n = 5, 10, \ldots, 50$ and $n = 60, 70, \ldots, 400$. To generate each instance, we randomly select separate distributions for start and goal vertices. We begin benchmarking with the fewest number of agents, running each algorithm on every instance with a 5 minute cutoff for each algorithm. We terminate benchmarking early on a map if all algorithms fail to solve an instance for two consecutive agent tiers. For example, if every algorithm fails to solve the ten instances for 100 and 110 agents on a given map, we halt execution on that map and do not benchmark for 120 agents. The primary reason for stopping execution is the diminishing returns and high costs of continuing. For instance, a single algorithm that completes none of the instances on a map for an agent tier takes 50 minutes. For an algorithm portfolio with six algorithms, a single tier of failed instances takes 3 hours to complete.

4 ALGORITHM PORTFOLIO

We include six state-of-the-art MAPF algorithms across a variety of approaches, including search based algorithms, SAT reduction algorithms, and constraint and mixed-integer programming algorithms. We now give a brief overview of each algorithm in our portfolio.

We include two popular search algorithms in our portfolio. The first search algorithm is Conflict-Based Search (CBS) [23]. CBS has been a standard of comparison among MAPF algorithms and has become the base of many optimal and suboptimal algorithms. CBS is a two-level search algorithm. A low level search, such as A^*, finds

Table 1: Performance of our portfolio algorithms and a hypothetical Oracle algorithm that selects the fastest algorithm for each individual instance. All values are for instances where at least one algorithm completed.

Algorithm	Completion Rate	Proportion Fastest	Total Runtime (hrs)	Marginal Contribution (hrs)
CBS	0.31	0.0	816	0.0
CBSH	0.68	**0.458**	391	15.24
BCP	0.66	0.082	428	31.14
ID-SAT	0.43	0.002	712	1.96
SMT-CBS	0.47	0.001	657	0.01
LazyCBS	**0.95**	**0.458**	**115**	**243.41**
Oracle	1.0	-	55	-

paths for agents to their goals. The high level search builds a conflict tree, where each node in the tree is a conflict between agents found by the low level planner. Each conflict node is expanded and generates constraints for the low level planner that prevent the previous collision. CBS has been improved by many new advances, several of which are included in our portfolio. The first such algorithm in our portfolio is CBSH [18], which included heuristics in the high level search of CBS to significantly improve performance. We use the version of CBSH with rectangular reasoning [19].

We include two SAT reduction based algorithms in our portfolio. We include an updated version of multi-decision diagram based satisfiability reduction, MDD-SAT [29], that incorporates Independence Detection (ID) [26], which we call ID-SAT [30]. The second SAT based algorithm we include is SMT-CBS [28], which converts CBS into a satisfiability modulo theory (SMT) style SAT algorithm. We chose not to include the original boolean satasfiability algorithm or an Answer Set Programming approach [2, 29]; while these compilation methods are flexible and can be used with many off the shelf solvers, their performance on MAPF instances is slower than algorithms tailored specifically to MAPF.

The final two algorithms in our portfolio are Lam et al.'s branch-cut-and-price (BCP) algorithm [16] and Gange et al.'s lazy-constraint programming algorithm [5]. BCP formulates the instance as a Mixed-Integer Program and solves it using a MAPF-specific version of branch-cut-and-price. LazyCBS combines constraint programming with CBS. CBS can often encounter the same conflict twice at different points in the conflict tree, which requires fixing these conflicts multiple times. LazyCBS fixes this issue by replacing the standard high level search of CBS with a constraint programming model.

For all algorithm implementations, we used original author code or a standard public implementation. All experiments were conducted on a Linux machine with an AMD Ryzen 9 3950X 16-Core Processor and 64 GB of RAM. Algorithms were terminated after 300 seconds if they had not produced a solution.

5 MAPF ALGORITHM PERFORMANCE

The results generated by our benchmarking help us understand how the strengths of certain algorithms overlap with the weaknesses of others. We include results of an Oracle algorithm, which is the hypothetical algorithm that always selects and runs the best algorithm in the portfolio for any given instance. The gap between Oracle and the other algorithms in the portfolio is the benefit gained by using an algorithm portfolio rather than any single algorithm. We measure algorithm performance along four metrics: completion rate, proportion of instances where the algorithm is the fastest algorithm, total runtime (hours), and marginal contribution (hours). Marginal contribution for an algorithm a is the difference in performance of an Oracle including algorithm a and an Oracle that does not include a. Completion rate and the proportion fastest are both measured in terms of completed instances, rather than attempted instances. LazyCBS has a completion rate of 0.62 over all instances attempted. Of these attempted instances, 35% were not solved by any algorithm. On solved instances where at least one algorithm completed, LazyCBS has a completion rate of 0.95. Table 1 shows performance for these algorithms on the set of solved instances. When algorithms do not complete an instance, 300 seconds is added to their total runtime, the lower bound of their possible runtime on that instance.

Our results show that LazyCBS is clearly the strongest algorithm in our portfolio, solving the most instances and taking the least cumulative time overall. However, CBSH is also the fastest algorithm on 45.8% of all completed instances, the same as LazyCBS. This is in large part due to CBSH's speed on instances with fewer agents. However, despite a much lower *Faster* metric, BCP scores higher in marginal contribution than CBSH, meaning it actually makes a larger difference to the performance of the portfolio than CBSH. For most instances where CBSH is fastest, another algorithm finishes marginally slower. However, for many instances where BCP is fastest, no other algorithm is able to solve the instance under the cutoff time.

When trying to select an algorithm to add to an algorithm portfolio, the instances the algorithm is able to solve that were not previously solvable by the portfolio are more important than the individual runtime and completion rate of that algorithm. However, previous evaluations of new MAPF algorithms typically consider only completion rate. For new MAPF algorithms, it is important to consider how the instances they solve complement existing algorithms. The MAPF community may be filtering out innovative, useful approaches that are not as strong when considered independently, but are diverse from existing algorithms.

Figure 2 demonstrates other performance differences between algorithms. This sorted-runtimes plot shows the number of instances solved within a certain time on the x-axis for each algorithm. In general, as the time given to an algorithm increases, the rate at which additional new instances are solved decreases exponentially (n.b. the log-scaled x-axis). While BCP gets off to slow start due to higher overhead than the search-based algorithms, the number of instances it solves is nearly that of CBSH as the cutoff time of 300 seconds is reached. It is possible that if a cutoff time larger than 300 seconds were used, BCP would have solved more instances than CBSH. The choice of cutoff time when comparing algorithms is important and can lead to different relative performances of algorithms. If a cutoff time of 1 second was chosen, CBSH and LazyCBS would have solved roughly the same number of instances. As the cutoff time is increased, the performance of LazyCBS relative to the other algorithms only increases. If a cutoff time of more than five minutes were chosen, we would expect the performance of LazyCBS relative to the rest of the portfolio to increase as well.

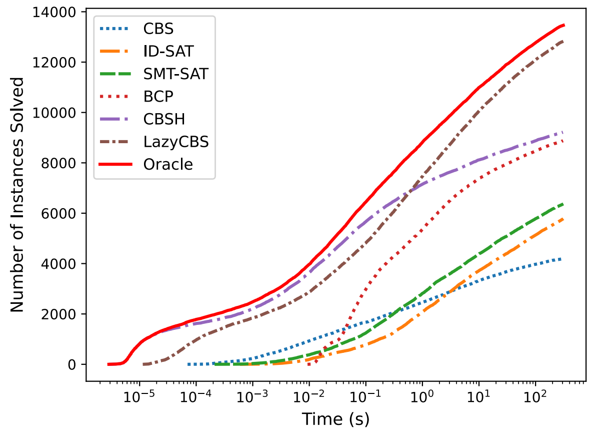


Figure 2: The number of instances solved within certain time for each algorithm. A point on the graph (x, y) indicates that for a cutoff time of x, y instances were completed.

It is important to note that the relative strength of LazyCBS is biased by our benchmarking procedure. As the number of agents increases, the performance of LazyCBS increases *relative* to the other algorithms in our portfolio. If we had chosen to stop all maps at 100 agents and perform more benchmarking with fewer agents, the relative performance of other algorithms would improve. In fact, the performance of LazyCBS in our results appears significantly stronger than it does in Gange et al.'s original paper, where experiments were on only four maps and experiments stopped at 120 agents [5].

6 BETWEENNESS CENTRALITY IN MAPF

We now turn our attention to characterizing the effect that betweenness centrality has on the performance of MAPF algorithms. We seek to characterize the empirical hardness of a given map $G = (V, E)$ for MAPF. The empirical hardness of G is the empirical performance of algorithms in our portfolio. We make no claims on the complexity of MAPF for any single map, as it could be possible to highly tailor an algorithm to a single map if known ahead of time. However, for the algorithms in our portfolio and other existing MAPF algorithms, performance can vary widely between different maps. On a specific warehouse environment, our portfolio completed instances up to 400 agents. On other similarly sized game and city maps they only completed up to 100 agents. Additionally, the difficulty of instances on a map can vary widely even for instances with the same number of agents. We focus on the expected difficulty of a random MAPF instance drawn from the space of possible MAPF instances on G. A random instance on an easier map is more likely to be completed than a random instance on a hard map. We connect the graph metric betweenness centrality to the hardness of G. Betweenness centrality is a measure of centrality for each node $u \in V$, defined as the fraction of all possible shortest paths that u lies on. We will provide a formal justification for this trend and identify the effects of betweenness centrality on the empirical hardness of a map.

The primary driving force of runtime for algorithms in our portfolio is the number of conflicts resolved when solving an instance. The CBS family of algorithms depends on building a conflict-tree data structure and searching this tree until a conflict-free path is found. The algorithms primarily differ in how they search this tree, with the newer algorithms identifying branches which cannot lead to feasible solutions thus reducing the search space. The runtime of these algorithms generally grows exponentially with the depth of the conflict tree, which itself grows with the number of conflicts resolved between agents. BCP also scales with the number of conflicts resolved, adding constraints to a master problem and re-solving when a conflict is found. The ID-SAT reduction algorithm does not maintain a conflict tree, but still scales with conflicts between agents both in the Independence Detection portion and the SAT reduction portion. Maps with choke points, where many agents traverse the same set of nodes, should generally be more difficult for these algorithms to solve. We will use the betweenness centrality graph metric to analyze the presence of choke points and their effect on average difficulty of a map.

We begin by analyzing the expected number of shortest paths traversing a single node $u \in V$. For a start and goal node $s, t \in V$ and a node $u \in V$, let $\sigma_{s,t}(u)$ be the number of shortest paths that traverse from s to t and pass through u. Let the total number of shortest paths from s to t be $\sigma_{s,t}$. The probability that a shortest path chosen uniformly at random from all possible shortest paths from s to t passes through u is given by:

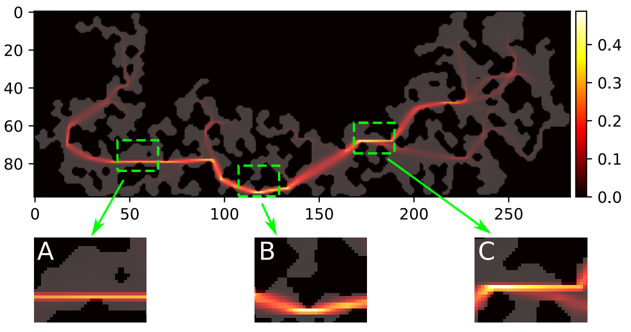
$$Pr[x_u|s,t] = \frac{\sigma_{s,t}(u)}{\sigma_{s,t}}. \qquad (1)$$

If s, t are chosen uniformly randomly from V, the normalized betweenness centrality of node u, denoted $BC(u)$, is the cumulative probability over all possible combinations of $s \neq t \neq u$:

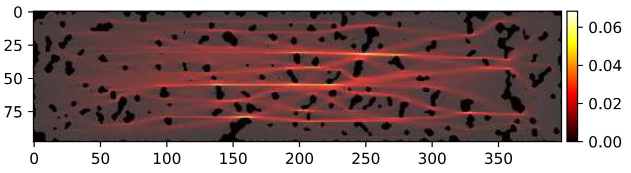
$$BC(u) = Pr[x_u] = \frac{1}{(|V|-1)(|V|-2)} \sum_{s\neq t\neq u\in V} \frac{\sigma_{s,t}(u)}{\sigma_{s,t}}. \qquad (2)$$

The betweenness centrality of node u is the probability that a shortest path chosen at random will traverse node u. Calculating BC for every node in the graph can be done in $O(|V||E|)$ time for unweighted graphs using Brandes' algorithm [3]. Betweenness centrality is often used as a measure of influence of nodes in a graph, often measuring the influence of individuals in social networks [7]. Computation of BC values can be further sped up by parallelizing shortest path computations with a GPU [21]. We used these speed ups and for the largest maps in our dataset used sample based approximations of betweenness centrality. In most cases the calculations could be done within two minutes and in some cases took up to five minutes. This is unreasonable to calculate for every MAPF instance, but only needs to be computed once per map and can be reused for multiple instances on that map.

Now we consider n agents with starts and goals chosen uniformly randomly from V. We first investigate the probability of traversing node u at any point in time on multiple agents' paths. For simplicity of analysis, we allow multiple agents to start from the same location and multiple agents to end at the same location if chosen to do so. When $n \ll |V|$, as is most often the case in MAPF, the impact of sampling start and goal nodes with replacement is small compared to sampling without replacement. The probability of traversing u is always $BC(u)$ for each of the n agents, making $Pr(x_u|n)$ a Bernoulli distribution with $p = BC(u)$. Let X_u be the number of agents to



(a) CA_0.5



(b) CA_0.4

Figure 3: Heatmaps of betweenness centrality on two selected maps. Three areas of interest with high betweenness centrality are highlighted in Fig. 3a.

Table 2: Performance information for CA_0.4 and CA_0.5

Algorithm	Completion Rate		Max # Agents	
	CA_0.5	CA_0.4	CA_0.5	CA_0.4
CBS	0.163	0.262	30	90
CBSH	0.513	0.556	80	200
BCP	**0.543**	0.396	**90**	170
ID-SAT	0.275	0.282	45	80
SMT-CBS	0.313	0.302	45	90
LazyCBS	0.506	**0.809**	80	**210**

traverse u. Then the expected number of agents to traverse u for an instance with n agents is:

$$E[X_u] = n \cdot BC(u).$$

The expected number of agents traversing node u grows linearly with both the total number of agents and the BC of node u. As the congestion on node u rises, so does the probability of collisions. Computing the exact probability of two agents colliding at a node requires knowledge of the underlying graph structure, specifically the distribution of nodes d distance away from u for all distances d from u. Two paths will collide at u if they both start d distance from u. If the distribution of nodes d distance from u is known, the probability of collision can be computed along the lines of the birthday-paradox problem or a hash-collision problem.

6.1 Relationship to Map Topology

To help better explain the relationship of betweenness centrality and map topology, we provide two examples in Fig. 3. Intuitively speaking, betweenness centrality is high in choke points. Fig. 3a contains a single distinct path of high betweenness centrality, averaging around $BC(u) = 0.25$ and peaking around $BC(u) = 0.4$ for nodes on that path. Fig. 3b shows a more open map with maximum betweenness centrality an order of magnitude lower than Fig. 3a. The narrow corridors of Fig. 3a constrict the nodes that shortest paths can travel on. However, it is possible to have nodes with high $BC(u)$ despite not being in a narrow corridor. The highlighted regions of Fig. 3a show different situations that can lead to high betweenness centrality values. Consider the three highlighted regions of Fig. 3a: A, B, and C. Region B is easily recognizable as a location of high betweenness centrality. It serves as the shortest connection between two halves of a map and is only a few cells tall. Regions A and C on the other hand, are less obvious locations for high betweenness centralities. It is possible to have high betweenness centrality even in relatively open locations. Shortest paths in A and C are constricted to the high BC path not by immediately adjacent obstacles, but by farther away structures. These different types of locations with high betweenness centrality could lead to different types of collisions between agents. If the optimality of the paths used for calculating $BC(u)$ were to be relaxed slightly, B would still have a high $BC(u)$, as the obstacles force the paths to travel on a small set of nodes. However A and C would have less concentrated $BC(u)$ values. Resolving conflicts in A and C may be easier for some algorithms than resolving conflicts in B. In B, agents must wait for the choke point to be vacated before traversing through. In A and C, agents may simply take a step out of their way to avoid an oncoming agent and continue on their way.

Table 2 presents the performance of our portfolio on the two maps in Fig. 3. In the more open and less constrained map CA_0.4, LazyCBS performs best in terms of completion rate and the highest number of agents solved for. However, in the highly constrained map CA_0.5, LazyCBS is outperformed by BCP. Additionally, the total number of instances solved by any algorithm was significantly lower for CA_0.5, with BCP topping out at 90 agents. Some of the performance difference is due to CA_0.5 having less open space than CA_0.4, as the number of nodes in a map affect the difficulty of MAPF instances, since agents tend to be more spread out on larger maps. However, that does not explain the entire difference in performance. In other maps with lower average betweenness centrality and a similar number of nodes to CA_0.5, algorithms were able to solve out to 200 agents, twice as many as for CA_0.5.

6.2 Empirical Relationship to Difficulty

We now turn our attention to measuring the empirical hardness for all maps in our dataset. To empirically measure the hardness of a map in our dataset, we use the number of instances solved for each map. Overall completion rate between maps is not comparable for our specific dataset because we terminate some maps early if our portfolio fails too frequently. Ideally, we would run all maps for the same number of instances and directly compare maps based on the success rate of our algorithms. However, the cost of doing so is too prohibitive, given a single instance takes on the order of half an hour to complete with a 5 minute cutoff for each algorithm. We can be fairly confident, given that maps are run until no instances complete for two sequential agent tiers, that attempting the map with more agents will lead to very few additional solved instances. Therefore,

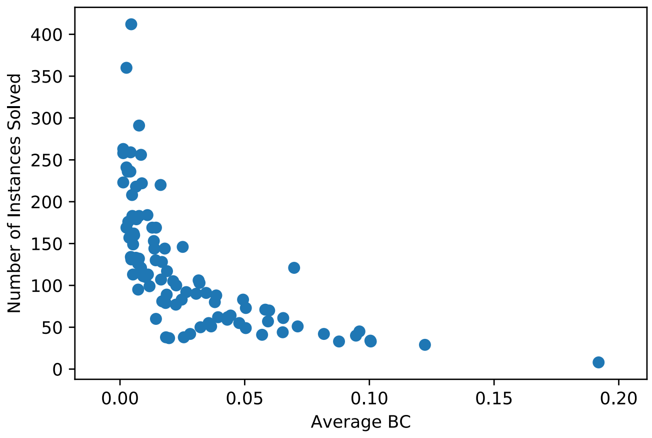


Figure 4: Average betweenness centrality of nodes in a graph compared to the number of instances solvable by our portfolio. As the betweenness centrality increases, the difficulty of an average instance increases and fewer instances are solved.

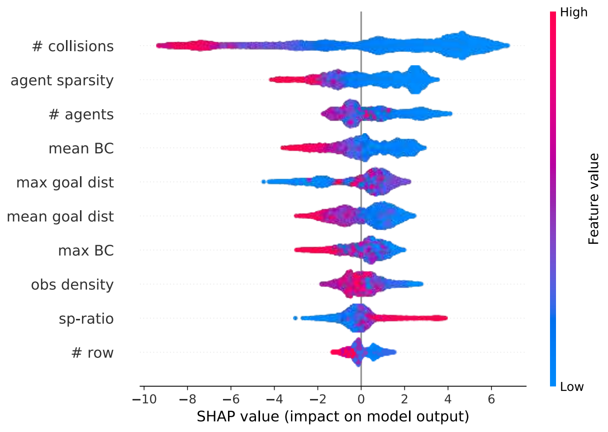


Figure 5: SHAP values of features for an XGBoost model predicting whether our algorithm portfolio will complete an instance or not.

the number of instances attempted is at least an approximation to the overall completion rate for running each algorithm out to 400 agents for every map. Fig. 4 shows the relationship of average BC for a map and the empirical difficulty for our algorithm portfolio on that map. As BC increases, the number of instances solved tends to decrease, supporting the claim that the difficulty of an average instance depends on the BC values of that map.

7 HARDNESS ESTIMATION

For certain applications, it is useful to know how long it will take for a MAPF algorithm to run. Autonomous warehouse operators may prefer to plan optimal paths, but if they predict a certain instance will take too long to solve optimally, they can switch to a suboptimal algorithm. We train a model to predict whether or not any algorithm in our portfolio will finish before the cutoff time. Our model builds on the feature-based classification and runtime estimation XGBoost models introduced by Kaduri et al. [11]. XGBoost [4] is a gradient-boosting tree algorithm capable of strong performance in classification and regression tasks. Kaduri et al. used a set of 12 features to describe a single MAPF instance. The set of features includes information on the number of agents and agent density, the size of the environment, and the distribution of start and goal locations. To these we add three new features: average betweenness centrality, max betweenness centrality, and the number of collisions between single-agent shortest paths for each agent to its goal.

We used the full dataset for our portfolio described in Section 5. The dataset is made up of 65% instances where at least one algorithm in the portfolio successfully solved the instance and 35% instances where none of our algorithms finished. We trained two versions of the XGBoost model, one with our centrality and conflict features, and one that used only the original features introduced by Kaduri et al. Data was divided into train, test, and validation sets and a grid search over hyperparameters was performed for both models to optimize performance. The XGBoost model with the additional features predicted whether an instance would be completed by an algorithm in our portfolio with 92.1% accuracy. The XGBoost model with only the features of Kaduri et al. reached a peak accuracy of 91.0%.

To further explore the effect of the additional features, we perform a SHapley value Additive exPlanation (SHAP) value analysis [20]. SHAP values measure the impact of each feature on the final output value for a single instance. SHAP values are calculated for every feature and every instance. SHAP values for our model with additional betweenness centrality and conflict features are presented in Fig. 5. Each point is the SHAP value of a given feature on one specific instance. For binary classification tasks, negative SHAP values for a specific feature indicate that feature pushed the final output closer to 0, and a positive SHAP value means the feature pushed the final output closer to 1. The farther that a point is from the center, the larger the impact that feature has. Additionally, each point is also colored with the relative value of the feature for that given instance. When the number of collisions feature is high, the model is more likely to predict that the instance will not be completed. Features are ordered by average absolute SHAP values. On average, features towards the top have a higher impact than features toward the bottom. Only the most impactful ten features are shown, the remaining five features are on average less impactful than the number of rows in the environment.

The features can be separated into two categories, instance-specific and map-specific features. Instance specific features, like the number of agents, collisions between agents, and information about start and goal locations, tend to be more important than map-specific features, like the size of the environment and obstacle density. The mean betweenness centrality for a map is on average the most impactful map-specific feature for our model. Figure 5 shows a clear trend of high average betweenness centrality having a negative impact on SHAP values and vice versa. By measuring the impact of map-specific features, we may be able to design maps that are empirically easier for MAPF algorithms, for instance by creating maps with lower mean betweenness centrality.

8 CONCLUSION AND FUTURE WORK

In this work we have explored the relationship between betweenness centrality and the difficulty of MAPF instances. Betweenness centrality is correlated with the number of collisions expected in a random MAPF instance. Maps with high betweenness centrality are likely to lead to more inter-agent conflicts and result in MAPF instances that are harder to solve optimally with existing algorithms. We introduced two ways to procedurally generate new maps for MAPF problems and performed an extensive benchmarking on an algorithm portfolio on a large dataset of maps.

Measuring algorithm performance with respect to a portfolio of algorithms will be important for finding new and useful algorithms for MAPF. If algorithms are only compared directly to existing algorithms using metrics such as completion rate, the community may be filtering out *specialist* algorithms that would make a strong addition to an algorithm portfolio. Specialist algorithms may be able to excel on certain types of instances, but not be able to compete with existing algorithms over a larger diverse set of instances. These specialist algorithms would make a solid contribution to an algorithm portfolio approach, as their weaknesses could be compensated for by other algorithms in the portfolio. We therefore believe that MAPF algorithms should not be compared only by the number of total instances solved, but also the number of instances they are able to solve that no other algorithm is capable of.

There are a number of directions for future research to build off of our initial results. First, as discussed above concerning Fig. 3a, not all nodes with high betweenness centrality will lead to the same type of conflict; some conflicts may be easier to resolve than others (i.e. open space conflicts vs. narrow corridor conflicts). Benchmarking portfolios of algorithms has so far focused on optimal MAPF algorithms, but we also need to gather more information on suboptimal algorithms and measure how their performance is affected by map topological features such as betweenness centrality.

Moving forward, we will seek to combine the benchmarking results of our own dataset with Ren et al.'s [22], Kaduri et al.'s [12], and Sigurdson et al.'s [24] datasets. While the datasets are difficult to merge for their original intent of algorithm selection, as each dataset contains different benchmarked algorithms, the data can still provide information on the relative hardness of MAPF instances.

9 ACKNOWLEDGEMENTS

This work was supported by NSF grants 1553726, 1724392, and 1837779.

REFERENCES

[1] Jacopo Banfi, Nicola Basilico, and Francesco Amigoni. 2017. Intractability of time-optimal multirobot path planning on 2D grid graphs with holes. *IEEE Robotics and Automation Letters* 2, 4 (2017), 1941–1947.
[2] Roman Barták, Neng-Fa Zhou, Roni Stern, Eli Boyarski, and Pavel Surynek. 2017. Modeling and solving the multi-agent pathfinding problem in picat. In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 959–966.
[3] Ulrik Brandes. 2001. A faster algorithm for betweenness centrality. *Journal of mathematical sociology* 25, 2 (2001), 163–177.
[4] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proc. of the 22nd ACM SIGKDD Intl Conf on Knowledge Discovery and Data Mining*. ACM, New York, 785–794.
[5] Graeme Gange, Daniel Harabor, and Peter J Stuckey. 2019. Lazy CBS: implicit conflict-based search using lazy clause generation. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 29. ICAPS, 155–162.
[6] Song Gao, Yaoli Wang, Yong Gao, and Yu Liu. 2013. Understanding urban traffic-flow characteristics: a rethinking of betweenness centrality. *Environment and Planning B: Planning and design* 40, 1 (2013), 135–153.
[7] K-I Goh, Eulsik Oh, Byungnam Kahng, and Doochul Kim. 2003. Betweenness centrality correlation in social networks. *Physical Review E* 67, 1 (2003), 017101.
[8] Petter Holme. 2003. Congestion and centrality in traffic flow on complex networks. *Advances in Complex Systems* 6, 02 (2003), 163–176.
[9] Frank Hutter, Lin Xu, Holger H Hoos, and Kevin Leyton-Brown. 2014. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence* 206 (2014), 79–111.
[10] Lawrence Johnson, Georgios N Yannakakis, and Julian Togelius. 2010. Cellular automata for real-time generation of infinite cave levels. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. 1–4.
[11] Omri Kaduri, Eli Boyarski, and Roni Stern. 2020. Algorithm Selection for Optimal Multi-Agent Pathfinding. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 30. 161–165.
[12] Omri Kaduri, Eli Boyarski, and Roni Stern. 2021. Experimental Evaluation of Classical Multi Agent Path Finding Algorithms. In *Proceedings of the International Symposium on Combinatorial Search*, Vol. 12. 126–130.
[13] Aisan Kazerani and Stephan Winter. 2009. Can betweenness centrality explain traffic flow. In *12th AGILE international conference on geographic information science*. 1–9.
[14] Alec Kirkley, Hugo Barbosa, Marc Barthelemy, and Gourab Ghoshal. 2018. From the betweenness centrality in street networks to structural invariants in random planar graphs. *Nature communications* 9, 1 (2018), 1–12.
[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*. 1097–1105.
[16] Edward Lam, Pierre Le Bodic, Daniel Damir Harabor, and Peter J Stuckey. 2019. Branch-and-Cut-and-Price for Multi-Agent Pathfinding. In *Proc. of the Intl Joint Conf on Artificial Intelligence*. 1289–1296.
[17] Loet Leydesdorff. 2007. Betweenness centrality as an indicator of the interdisciplinarity of scientific journals. *Journal of the American Society for Information Science and Technology* 58, 9 (2007), 1303–1319.
[18] Jiaoyang Li, Ariel Felner, Eli Boyarski, Hang Ma, and Sven Koenig. 2019. Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search.. In *IJCAI*. 442–449.
[19] Jiaoyang Li, Daniel Harabor, Peter J Stuckey, Hang Ma, and Sven Koenig. 2019. Symmetry-breaking constraints for grid-based multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 6087–6095.
[20] Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *Proceedings of the 31st international conference on neural information processing systems*. 4768–4777.
[21] Adam McLaughlin and David A Bader. 2014. Scalable and high performance betweenness centrality on the GPU. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 572–583.
[22] Jingyao Ren, Vikraman Sathiyanarayanan, Eric Ewing, Baskin Senbaslar, and Nora Ayanian. 2021. MAPFAST: A Deep Algorithm Selector for Multi Agent Path Finding using Shortest Path Embeddings. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*. 1055–1063.
[23] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219 (2015), 40–66.
[24] Devon Sigurdson, Vadim Bulitko, Sven Koenig, Carlos Hernandez, and William Yeoh. 2019. Automatic algorithm selection in multi-agent pathfinding. *arXiv preprint arXiv:1906.03992* (2019).
[25] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1409.1556

[26] Trevor Standley. 2010. Finding optimal solutions to cooperative pathfinding problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 24.

[27] Roni Stern, Nathan R Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Satish Kumar, et al. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Twelfth Annual Symposium on Combinatorial Search*.

[28] Pavel Surynek. 2019. Unifying search-based and compilation-based approaches to multi-agent path finding through satisfiability modulo theories. In *Twelfth Annual Symposium on Combinatorial Search*.

[29] Pavel Surynek, Ariel Felner, Roni Stern, and Eli Boyarski. 2016. Efficient SAT approach to multi-agent path finding under the sum of costs objective. In *ECAI*. 810–818.

[30] Pavel Surynek, Jiří Švancara, Ariel Felner, and Eli Boyarski. 2017. Integration of independence detection into sat-based optimal multi-agent path finding-A novel sat-based optimal MAPF solver. In *International Conference on Agents and Artificial Intelligence*, Vol. 2. 85–95.

[31] Thomas A Witten and Leonard M Sander. 1983. Diffusion-limited aggregation. *Physical review B* 27, 9 (1983), 5686.

[32] Jingjin Yu and Steven M LaValle. 2013. Structure and intractability of optimal multi-robot path planning on graphs. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*.