A Quality Diversity Approach to Automatically Generate Multi-Agent Path Finding Benchmark Maps

Cheng Qian[1*], Yulun Zhang[1*], Varun Bhatt[2], Matthew C. Fontaine[2], Stefanos Nikolaidis[2], Jiaoyang Li[1]

[1]Robotics Institute, Carnegie Mellon University
[2]Thomas Lord Department of Computer Science, University of Southern California
chengqia@andrew.cmu.edu, yulunzhang@cmu.edu,{mfontain,vsbhatt,nikolaid}@usc.edu, jiaoyangli@cmu.edu

Abstract

We use the Quality Diversity (QD) algorithm with Neural Cellular Automata (NCA) to generate benchmark maps for Multi-Agent Path Finding (MAPF) algorithms. Previously, MAPF algorithms are tested using fixed, human-designed benchmark maps. However, such fixed benchmark maps have several problems. First, these maps may not cover all the potential failure scenarios for the algorithms. Second, when comparing different algorithms, fixed benchmark maps may introduce bias leading to unfair comparisons between algorithms. Third, since researchers test new algorithms on a small set of fixed benchmark maps, the design of the algorithms may overfit to the small set of maps. In this work, we take advantage of the QD algorithm to (1) generate maps with patterns to comprehensively understand the performance of MAPF algorithms, (2) be able to make fair comparisons between two MAPF algorithms, providing further information on the selection between two algorithms and on the design of the algorithms. Empirically, we employ this technique to generate diverse benchmark maps to evaluate and compare the behavior of different types of MAPF algorithms, including search-based, priority-based, rule-based, and learning-based algorithms. Through both single-algorithm experiments and comparisons between algorithms, we identify patterns where each algorithm excels and detect disparities in runtime or success rates between different algorithms.

1 Introduction

We study the problem of generating diverse and targeted benchmark maps for Multi-Agent Path Finding (MAPF) algorithms. Given a map and a group of agents, MAPF is the problem of finding collision-free paths from their start to goal locations. MAPF has wide applications in coordinating hundreds of robots in automated warehouses (Li et al. 2021b; Varambally, Li, and Koenig 2022; Zhang et al. 2023a,b), moving characters in video games (Ma et al. 2017; Jansen and Sturtevant 2008), managing drone traffic (Ho et al. 2022; Choudhury et al. 2022), and controlling multi-robotic arms (Shaoul et al. 2024).

Given the wide applicability of MAPF, many algorithms are proposed to solve MAPF, and benchmarking these algorithms becomes an important task. Stern et al. (2019) have proposed a set of 33 MAPF benchmark maps, covering a diverse spectrum of map sizes, layouts, and difficulties.[1] These human-designed maps are widely used to test newly proposed MAPF algorithms (Li, Ruml, and Koenig 2021; Li et al. 2021a; Okumura et al. 2019; Skrynnik et al. 2024). However, such benchmark maps have several problems. First, they may not fully encompass the failure modes of an algorithm. Second, with a small fixed set of maps, it is challenging to sufficiently understand the pros and cons of different algorithms in different maps. Third, while comparing multiple algorithms, researchers are prone to run experiments on a subset of maps that are biased towards the proposed algorithms, resulting in an unfair comparison.

Meanwhile, Quality Diversity (QD) algorithms have been used to generate a diverse set of high-quality solutions by optimizing a given objective function and a set of diversity measure functions. A recent work (Zhang et al. 2023b) has used QD algorithms to optimize layouts for automated warehouses. To ensure that the optimized layouts possess human-explainable and regularized patterns, a follow-up work (Zhang et al. 2023a) then leverages Neural Cellular Automata (NCA) to generate the layouts, while applying QD algorithms to optimize the parameters of the NCA. Cellular Automata (Gardner 1970) iteratively generates complex tile-based structures from a simple one through local interaction between tiles. Each tile decides its next state based on its neighbors via a fixed rule. Neural Cellular Automata (NCA) represents the rule using a convolutional neural network.

In this paper, we adapt the layout optimization approach based on the QD algorithm and NCA from the previous work (Zhang et al. 2023a) and use it with an alternative goal of generating diverse benchmark maps for MAPF algorithms. To demonstrate that our approach can generate benchmark maps for a broad spectrum of MAPF algorithms, we choose five algorithms, namely CBS (Sharon et al. 2015), EECBS (Li, Ruml, and Koenig 2021), PBS (Ma et al. 2019), PIBT (Okumura et al. 2019), and Learn-to-Follow (LTF) (Skrynnik et al. 2024), representing search-based, priority-based, rule-based, and learning-based algorithms, respectively.

We first generate benchmark maps for each algorithm individually, presenting diverse maps that are easy or challenging for each algorithm to solve. We summarize the design

*These authors contributed equally.

[1]https://movingai.com/benchmarks/mapf/index.html

principles of benchmark maps that target each algorithm. We then generate benchmark maps that automatically compare two MAPF algorithms by generating maps that are easy for one algorithm and hard for the other. We aim to understand in which maps one algorithm outperforms the other in terms of either runtime or success rate.

We make the following contributions: (1) we adapt the layout optimization approach based on QD algorithms, proposing a framework to generate diverse MAPF benchmark maps, (2) we apply our framework on five representative MAPF algorithms of different categories, generating novel benchmark maps and providing new insights about their performance on maps of different patterns, and (3) we give guidelines for future researchers on how to better benchmark MAPF algorithms.

2 Preliminaries

2.1 Multi-Agent Path Finding (MAPF)

Definition 1 (Map) *A map is a four-neighbored 2D grid, where each tile can either be an empty space or an obstacle. A map is valid if all empty spaces are connected.*

Definition 2 (MAPF) *Given a valid map and a set of agents with their start and goal locations, MAPF aims to find collision-free paths from their start to goal locations. Agents can either move to their adjacent vertices or stay at their current vertices at each timestep. Two agents collide if they are at the same vertex or swap vertices at the same timestep. The objective of MAPF is minimizing the sum-of-cost, defined as the sum of path lengths of all agents.*

We present a short summary of existing MAPF algorithms and the chosen algorithms for experiments in this paper.

Search-Based. This category includes exponential-time algorithms that exhaustively explore the solution space of MAPF. They usually have theoretical guarantees such as optimality and suboptimality, but suffer from computational time. Examples include M* (Wagner and Choset 2011, 2015), BCP (Lam et al. 2019), ICTS (Sharon et al. 2013), CBS (Sharon et al. 2015; Gange, Harabor, and Stuckey 2019; Li et al. 2020), ECBS (Barer et al. 2014) and EECBS (Li, Ruml, and Koenig 2021). We choose CBS and EECBS as the representatives of this category. CBS starts by planning a shortest path for each agent, ignoring the collisions, and then resolving the collisions with a two-level search. The high-level search iteratively selects unresolved collisions and adds constraints to tackle them. The low-level search runs single-agent planning to compute new paths that satisfy the constraints. The search continues until all collisions are resolved. EECBS is a variant of CBS that uses Explicit Estimation Search (Thayer and Ruml 2011) on the high level and focal search (Pearl and Kim 1982) on the low level.

Priority-Based. Priority-based algorithms plan paths for each agent in a sequence, forcing agents with low priority to avoid colliding with those with high priority. The priority planning (PP) (Erdmann and Lozano-Pérez 1987) algorithm plans paths for agents in a pre-defined priority order. Monte-Carlo PP (Friedrich et al. 2024) pre-defines a set of randomized priority orders and selects the paths with the best solutions. PBS (Ma et al. 2019) combines CBS with PP to explicitly search for a good priority order with a two-level search. The high-level search of PBS is similar to CBS except that PBS constrains one agent to have a higher priority order than the other. The low-level search then plans paths with the order from the high-level search. We choose PBS as the representative of this category.

Rule-Based. Rule-based algorithms (Wang and Botea 2011; Okumura 2023) leverage pre-defined rules to move agents to their goals. They usually run much faster but produce worse solutions than search-based and priority-based methods. We choose PIBT (Okumura et al. 2019) as the representative of this category. PIBT uses an iterative one-timestep rule to move the agents. At each timestep, each agent plans a single-step action towards its goal. In case of collision, a predefined rule based on priority inheritance and backtracking is applied to resolve the collision.

Learning-Based. Learning-based algorithms usually formulate MAPF as a Multi-Agent Reinforcement Learning (MARL) (Sartoretti et al. 2019; Damani et al. 2021) problem. They train a shared policy for each agent, taking its local field of view as input and deciding its next action. We choose Learn to Follow (LTF) (Skrynnik et al. 2024) as the representative of this category. LTF starts by searching for a guide path for each agent without considering collisions. Then it uses a shared learnable policy to move the agents to their goals along the guide paths while avoiding collisions. LTF is developed for lifelong MAPF, a variant of MAPF that constantly assigns new goals to agents. We modify it for MAPF by asking agents to stop when they reach their first goals.

2.2 MAPF Benchmarks

Early works in MAPF typically use randomly generated benchmark maps (Wagner and Choset 2015) or maps from the single-agent path planning benchmark set (Sturtevant 2012). Stern et al. (2019) established the major set of MAPF benchmark maps with 33 maps of five categories (game, maze, empty, random, and room) and map sizes ranging from 32×32 to 256×256. This benchmark set is used extensively in MAPF research (Okumura et al. 2019; Li et al. 2021a; Okumura 2023; Friedrich et al. 2024). However, while testing novel MAPF algorithms, instead of using all maps, researchers usually select a subset of them of different categories and sizes. This could result in unfair and biased benchmarking results because they can intentionally or unintentionally cherry-pick maps that favor their proposed algorithms, or the design of their algorithms could overfit to the selected maps.

The MARL community usually generates random maps to test the trained policy for MAPF problems (Sartoretti et al. 2019; Damani et al. 2021). Researchers who develop MAPF algorithms for the multi-robot systems for automated warehouses or sortation centers often use a set of warehouse maps (Li et al. 2021b; Varambally, Li, and Koenig 2022; Zhang et al. 2023b,a, 2024).

2.3 Quality Diversity (QD) Algorithms

QD algorithms (Cully et al. 2015; Lehman and Stanley 2011a,b) are inspired by evolutionary algorithms with diversity search to generate a diverse collection of high-quality solutions by optimizing an objective function and diversifying a set of diversity measure functions simultaneously. QD algorithms maintain an *archive*, which is a tessellated measure space defined by the measure functions. The archive stores the best solution in each tessellated cell. QD algorithms then optimize the sum of objective values of all solutions in the archive, defined as QD-score. We choose Covariance Matrix Adaptation MAP-Annealing (CMA-MAE) (Fontaine and Nikolaidis 2023) as the method for generating benchmark maps because it is the state-of-the-art QD algorithm specialized for continuous search domains. CMA-MAE is an extension of MAP-Elites (Mouret and Clune 2015) that incorporates the covariance matrix adaptation mechanism of CMA-ES (Hansen 2016), which is a derivative-free single-objective optimizer. CMA-ES maintains a multi-variate Gaussian distribution and iteratively samples from it for new solutions. It evaluates the solutions and updates the Gaussian toward high-objective regions. CMA-MAE adapts this mechanism to optimize the QD-score.

2.4 Automatic Scenario Generation

Automated methods of generating scenarios to understand, test, or benchmark a given algorithm have been proposed by different fields. In autonomous driving, researchers have generated scenarios to test and evaluate developed autonomous driving systems (Arnold and Alexander 2013; Abeysirigoonawardena, Shkurti, and Dudek 2019; Mullins et al. 2018). In human-robot-interaction, prior works have used QD algorithms to generate shared autonomy scenarios (Fontaine et al. 2021b) or diverse kitchen layouts (Fontaine et al. 2021a) to understand the coordination behavior between humans and robots. A follow-up work (Bhatt et al. 2023) leverages model-based QD methods (Bhatt et al. 2022; Zhang et al. 2022) to improve the sample efficiency of QD algorithms. In reinforcement learning, prior works have generated maps to benchmark (Cobbe et al. 2020) or continuously improve (Wang et al. 2019, 2020) trained agent.

In MAPF, one recent work (Ren et al. 2024) has used QD to generate maps for MAPF. Our work differs in two key aspects: First, they focus on generating maps of fixed difficulties, quantified by an approximate metric based on map connectivity, while we are interested in generating benchmarking maps targeted for specific MAPF algorithms. Second, they directly optimize tile types (i.e., whether it is an empty space or an obstacle) using MAP-Elites, while we optimize a map generator based on NCA, which has been shown to be more effective for generating maps with diverse regularized patterns (Zhang et al. 2023a).

3 Benchmark Generation Approach

We adapt the previous work (Zhang et al. 2023a) to use CMA-MAE and NCA to generate diverse benchmark maps with the objective and measures computed by running MAPF algorithms. Figure 1 shows the overview of the method. Starting by sampling b parameter vectors $\boldsymbol{\theta}$, we form b NCA generators and use them to generate b maps. Maps generated by NCA might not be valid. We then adapt a mixed integer linear programming (MILP) solver (Zhang et al. 2020, 2023a,b) to repair the map such that (1) the map is valid, and (2) the number of obstacles falls in a pre-defined range $[O_{lb}, O_{ub}]$. We then run MAPF algorithms to evaluate the objective and measure values and add the evaluated maps to an archive. For each evaluation, we run the given MAPF algorithm in N_e instances. Finally, we update the Gaussian distribution, starting a new iteration. We repeat this process until we evaluate N_{eval} maps.

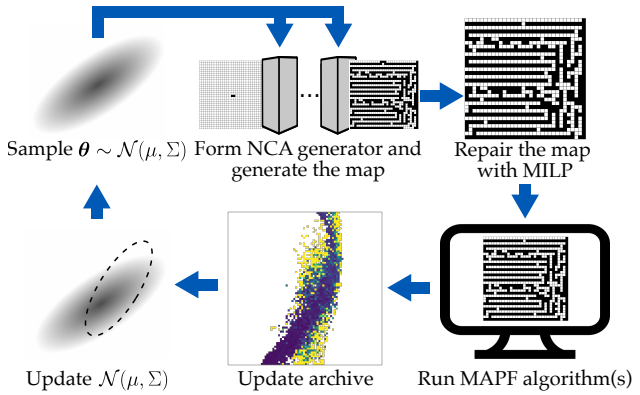


Figure 1: Overview of our approach of using CMA-MAE to optimize NCAs to generate diverse benchmark maps.

MAPF Instance. In this paper, we focus on generating MAPF benchmark *maps* of a pre-defined size and number of agents. To generate a MAPF *instance* based on a map, we follow the bucket method (Stern et al. 2019) to generate evenly distributed start and goal locations of the agents with a distance constraint, which was used for generating the "even" scenarios in the MAPF benchmark. For each map, we generate N_e MAPF instances with different starts and goals, run MAPF algorithms on each instance, and compute the average objective and measures of all instances as the objective and measures of the map.

Framework Realization. Our proposed framework is flexible on a high level. It is capable of generating a diverse set of high-quality MAPF benchmark maps tailored to different MAPF algorithms by carefully designing objective and diversity measures. In this section, we propose two concrete realizations of our framework that are useful for benchmarking MAPF algorithms by designing objectives and measures. We first provide an *one-algorithm* realization, intending to generate diverse and difficult benchmark maps targeted for five representative MAPF algorithms. We then provide a *two-algorithm* realization, aiming to generate maps that are easy for one algorithm and hard for the other. We highlight that other realizations, such as generating easy maps or comparing more than two algorithms, are possible.

3.1 Objective

In all experiments, our objective is a function $f : \boldsymbol{X} \to \mathbb{R}$, where $\boldsymbol{X}$ is the space of all maps. The function f runs one or two MAPF algorithms on N_e different MAPF instances with N_a agents. For CBS, EECBS, and PBS, we set a time limit T for each run. For PIBT and LTF, we set a maximum makespan M.

One-Algorithm Experiments. We intend to generate maps that are challenging for the MAPF algorithms in one-algorithm experiments. The objective quantifies the empirical hardness of a map $\boldsymbol{x} \in \boldsymbol{X}$ to a MAPF algorithm $\phi \in A$, where $A = \{\text{CBS, EECBS, PBS, PIBT, LTF}\}$ is all MAPF algorithms considered in the experiments.

For CBS, EECBS and PBS, we maximize their *CPU runtime* because they are exponential-time algorithms with the major limitation being frequently running out of time. We do not consider the solution quality as the objective because CBS and EECBS are proven to be optimal and bounded-suboptimal algorithms. PBS, while being unbounded suboptimal, empirically finds close-to-optimal solutions. For example, Ma et al. (2019) shows that PBS finds solutions never over 4% worse than optimal in fixed game maps. Therefore, we maximize the *CPU runtime* to find failure cases for CBS, EECBS, and PBS. Concretely, suppose the function $t_\phi : \boldsymbol{X} \to \mathbb{R}_{>0}$ computes the average CPU runtime by running $\phi \in A$ in N_e instances, the objective f for the one-algorithm experiments is $f(\boldsymbol{x}) = t_\phi(\boldsymbol{x}), \phi \in \{\text{CBS, EECBS, PBS}\}$. If the algorithms fail to return solutions within the timelimit T, we set the CPU runtime as T.

PIBT and LTF, on the other hand, suffer mainly from deadlocks, meaning that some agents might never reach their goals. Therefore, to find failure cases, we minimize the *Regularized Success Rate (RSR)*, defined as follows:

$$RSR_\phi(\boldsymbol{x}) = \begin{cases} SR_\phi(\boldsymbol{x}) & \text{if } SR < 1 \\ SR_\phi(\boldsymbol{x}) \cdot C - SoC_\phi(\boldsymbol{x}) & \text{if } SR = 1 \end{cases} \quad (1)$$

$SR_\phi : \boldsymbol{X} \to \mathbb{R}_{>0}$ computes the success rate of running algorithm ϕ on one MAPF instance of a map, which is computed as the percentage of agents that successfully reach their goals within the given makespan M. $SoC_\phi : \boldsymbol{X} \to \mathbb{R}_{>0}$ computes the sum-of-cost of the solution if $SR = 1$. C is a large constant making sure that the RSR score of maps with $SR = 1$ always dominates that of maps with $SR < 1$. Intuitively, if not all agents reach their goals ($SR < 1$), we maximize the success rate. If all agents reach their goals ($SR = 1$), we regularize the success rate with the sum-of-cost to quantify the quality of the solution. Let the function $r_\phi : \boldsymbol{X} \to \mathbb{R}_{>0}$ return the average regularized success rate by running algorithm ϕ in N_e MAPF instances. Concretely, $r_\phi(\boldsymbol{x}) = \sum_{i=1}^{N_e} RSR_\phi^{(i)}(\boldsymbol{x})$, where i denotes the i-th run of algorithm ϕ. Then the objective of PIBT and LTF are $f(\boldsymbol{x}) = r_{\text{PIBT}}(\boldsymbol{x})$ and $f(\boldsymbol{x}) = r_{\text{LTF}}(\boldsymbol{x})$, respectively. We do not consider runtime as the objective of PIBT and LTF because they run very fast, making CPU runtime not a good metric of their performance in a map.

Two-Algorithm Experiments. To compare two given MAPF algorithms in two-algorithm experiments, we aim to generate maps that maximize the performance gap between the two algorithms. We consider two pairs of comparisons: (1) EECBS vs. PBS, and (2) PIBT vs. LTF. We compare EECBS and PBS because (1) both suboptimal algorithms use a two-level search based on collision avoidance, (2) both find near-optimal solutions empirically, and (3) few prior works have compared them thoroughly. We compare PIBT and LTF because (1) both use a pre-defined policy, either rule-based or learned, to move the agents step by step to their goals while avoiding collisions, (2) both run fast but suffer from deadlocks, and (3) while the LTF work (Skrynnik et al. 2024) has compared them on warehouse maps, showing that LTF outperforms PIBT, we want to find cases where PIBT outperforms LTF.

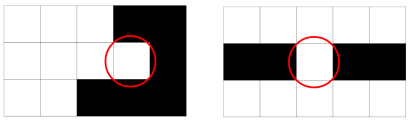
To compare EECBS and PBS, we set the objective as the absolute difference in the average CPU runtime. Concretely, the objective of a generated map $\boldsymbol{x} \in \boldsymbol{X}$ is computed as $f(\boldsymbol{x}) = |t_{\text{EECBS}}(\boldsymbol{x}) - t_{\text{PBS}}(\boldsymbol{x})|$. Similarly, to compare PIBT and LTF, we set the objective as the absolute difference in the regularized success rate, i.e., $f(\boldsymbol{x}) = |r_{\text{PIBT}}(\boldsymbol{x}) - r_{\text{LTF}}(\boldsymbol{x})|$.

3.2 Diversity Measures

We aim to select diversity measures such that the generated maps are diverse in terms of (1) general hardness towards all algorithms to analyze whether the general hardness of a map aligns with the hardness to a specific algorithm, and (2) spatial arrangement of the obstacles to generate maps of different patterns.

General Hardness. To measure the general hardness of a map, the most commonly used metric is the number of obstacles. The number of obstacles in a map affects the space available to resolve collisions as well as the search space of the algorithms. Therefore, diversifying it would give a range of difficulties. We also consider two metrics that are proposed specifically to reflect the hardness of a map: the standard deviation of Betweenness Centrality (std of BC) (Ewing et al. 2022) and λ_2 (Ren et al. 2024). To compute the std of BC, we search for the shortest path between every pair of empty spaces in the map and then calculate the standard deviation of the usage of all empty spaces. The higher the std of BC, the more congestion and the harder the MAPF problem. To compute λ_2 of a map, we view the map as a graph $G(V, E)$, and compute the second smallest eigenvalue of the normalized Laplacian matrix of G. Ren et al. (2024) shows empirically that λ_2 is correlated with the hardness of the map.

Spatial Arrangement. Aside from the hardness of the maps, we want to diversify the spatial arrangement of the maps, making the generated maps stylistically diverse. We first consider the KL divergence of the tile pattern distribution (Fontaine et al. 2021b) between the tile pattern distribution of the generated map and one category of fixed maps selected from the MAPF benchmark (Stern et al. 2019). A *tile pattern* is one possible arrangement of the obstacles and empty spaces in a 3×3 grid. To compute the tile pattern distribution of a map, we count the number of all possible tile patterns in the map. We then compute the KL divergence between the tile pattern distribution of the map and a



(a) One-entry space. (b) One-tile entry.

Figure 2: Examples of one-entry space and one-tile entry, circled in red.

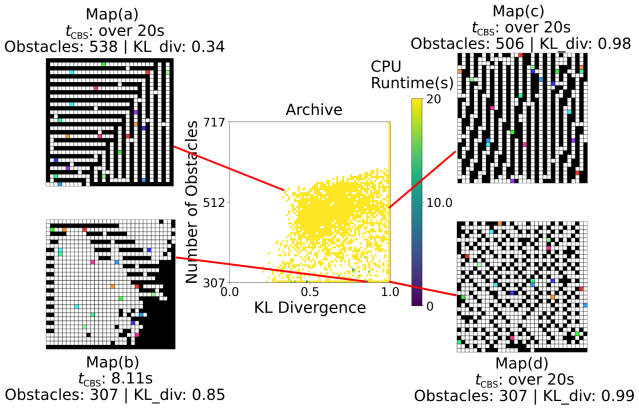


Figure 3: Archive for CBS with representative maps.

fixed distribution computed from all the maze maps[2] of the human-designed MAPF benchmark set (Stern et al. 2019). The smaller the KL divergence of tile pattern distribution, the more similar in local patterns our generated map is to the selected set of benchmark maps. We choose maze maps because they empirically yield the most complex patterns. In addition, we consider the entropy of the tile pattern distribution (Zhang et al. 2023a). The lower the entropy, the more regularized patterns the map possesses. Finally, we consider the KL divergence of the Weisfeiler-Lehman (WL) graph feature (Shervashidze et al. 2011) between the generated map and the maze maps. Similar to the tile pattern distribution, a smaller KL divergence of the WL graph feature implies more similarities between the local patterns of the generated map and the maze maps.

In our experiments, we choose **the number of obstacles** and **KL divergence of tile pattern distribution** as the diversity measures. We empirically observe that, among the measures considered, the number of obstacles is the most correlated with the general hardness of the maps, and the KL divergence of tile pattern distribution yields the most diverse map patterns. We justify our choice with experimental results in Appendix A.

4 Benchmarking One Algorithm

For one-algorithm experiments, we intend to generate a diverse set of difficult benchmark maps targeted for five representative algorithms we choose from Section 2.1, namely CBS, EECBS, PBS, PIBT, and LTF.

[2]https://movingai.com/benchmarks/maze/index.html

4.1 Experiment Setup

Hyperparameters. Defining w as the suboptimality bound away from the optimal solution, we use $w = 1.5$ for EECBS. For CBS, we use the implementation of EECBS with $w = 1$. We set $T = 20$ seconds for CBS, EECBS, and PBS, $M = 1000$ for PIBT, and $M = 512$ for LTF. For all algorithms, we fix the map size as 32×32 and the number of agents at 150. The pre-defined range for the number of obstacles is $[O_{lb} = 307, O_{ub} = 717]$, which corresponds to 30% to 70% of the map size. For each evaluation, we run the given MAPF algorithm in $N_e = 5$ different instances. For all algorithms, we evaluated $N_{eval} = 10,000$ maps. For CBS, EECBS, and PBS, we run evaluations with $N_a = 50$ agents, and for PIBT and LTF, we use $N_a = 150$. We use more agents for PIBT and LTF because they empirically have higher success rates in congested MAPF instances.

Implementation. We implement CMA-MAE with Pyribs (Tjanaka et al. 2023). For MAPF algorithms, we use the open source implementations of CBS and EECBS[3], PBS[4], PIBT[5], and LTF[6].

Compute Resources. We run our experiments on three machines: (1) a local machine equipped with a 64-core AMD Ryzen Threadripper 3990X CPU and 192 GB of RAM, (2) a local machine with a 64-core AMD Ryzen Threadripper 7980X CPU and 128 GB of RAM, and (3) a high-performance cluster featuring multiple 64-core AMD EPYC 7742 CPUs, each with 256 GB of RAM. All CPU runtimes are measured on machine (1).

4.2 Results

We first define *one-entry space* and *one-tile entry* (shown in Figure 2) as features of the benchmark maps. In Figures 3 to 5, the yellow cells indicate maps where the algorithm runs out of time. The dark blue cells indicate maps where the algorithm quickly finds a solution. In Figures 6 and 7, the yellow cells indicate maps where the algorithm fails to plan for any agents. The dark blue cells indicate maps where the algorithm successfully plans for all agents. The colored tiles in the representative maps display 10 out of 150 pairs of start and goal locations. While we are primarily interested in maps that are challenging for the algorithms, we also find maps that are easy, as the by-product of the experiment. We discuss more in Section 6.

CBS. Figure 3 shows the archive of maps for CBS. It is easy to generate challenging maps for CBS. Map (a), Map (c), and Map (d) show the most typical patterns where it is hard for CBS to find optimal solutions. Map (a) and Map (c) contain many long corridors, and Map (d) contains many one-entry spaces with long corridors, where one-entry spaces represent the empty space surrounded by three obstacles. Figure 2a shows an example. By running CBS on Map (a) and Map (b) in 200 MAPF instances, the success rate is below 5% with an average runtime of over 18 seconds. The

[3]https://github.com/Jiaoyang-Li/EECBS
[4]https://github.com/Jiaoyang-Li/PBS
[5]https://github.com/Kei18/pibt2
[6]https://github.com/AIRI-Institute/learn-to-follow

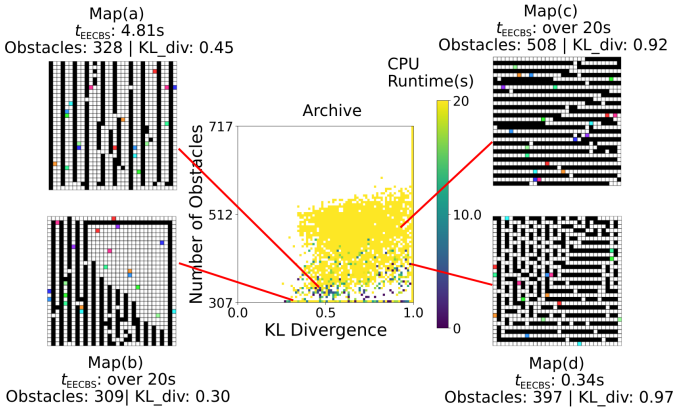


Figure 4: Archive for EECBS with representative maps.

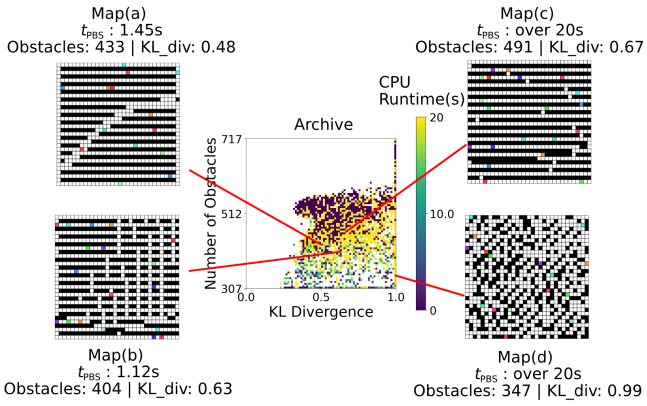


Figure 5: Archive for PBS with representative maps.

result shows that CBS performs poorly on maps with many long corridors and one-entry spaces.

Some maps that CBS can solve are maps with fewer obstacles and a large chunk of empty spaces. Map (b) is one of these typical maps. Map (b) contains a large chunk of empty spaces and short corridor components in between, providing more space for agents to avoid collisions.

EECBS. Figure 4 shows the archive of maps for EECBS. Compared to the archive of CBS, it is harder to generate challenging maps for EECBS. Map (b) and Map (c) are hard maps with typical patterns for EECBS. Map (b) contains a large chunk of empty space, but with many long corridors and one-tile entries. Map (c) contains many long corridors and one-tile entries. EECBS runs out of time on both maps. To further validate the results, we run EECBS with 200 different MAPF instances on these two maps. We get a success rate of only 11% with an average CPU runtime of 16 seconds, verifing that long corridors with one-tile entries are challenging map patterns for EECBS to solve. On the other hand, we observe that EECBS performs well in maps with more empty spaces between each long obstacle component and maps with short obstacle components, where obstacle components refer to clusters of two or more obstacles. Map (a) and Map (d) are two maps with typical patterns.

Map (a) contains many long obstacle components with one-tile entries; however, the map has two or more columns of empty space between each long obstacle component. Map (d) contains many short obstacle components with many entries in between, providing more space for agents to avoid collision. Other maps where EECBS performs well are of similar patterns or have a large chunk of empty spaces. With more entry space, it is easier for EECBS to resolve collisions. We show more maps with similar patterns in Appendix B to support our findings.

PBS. Figure 5 shows the archive of maps for PBS. We empirically discover that PBS returns no solutions in many maps. Therefore, as a side discovery, we show the maps in which PBS returns no solution in at least one out of five MAPF instances during the evaluation in the archive as purple cells.

Map (c) and Map (d) are hard maps with typical patterns

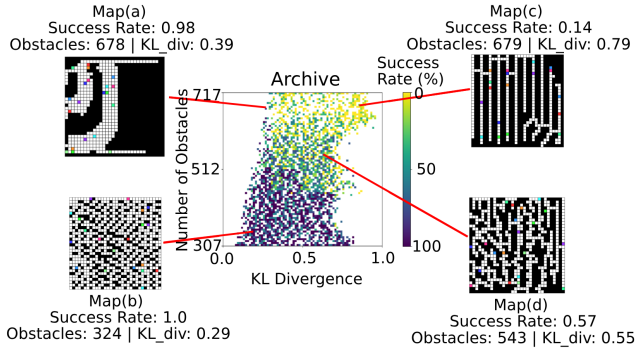


Figure 6: Archive for PIBT with representative maps.

for PBS. Map (c) contains many long corridors with one-tile entries in between, and Map (d) contains many one-entry spaces. PBS always reports no solution on Map (c) and runs out of time on Map (d). Upon running 200 MAPF instances on Map (c) and (d), PBS could hardly solve one instance in Map (c) and always reports no solution. On Map (d), PBS only achieves a success rate of 17.5% and an average runtime of 19.1 seconds for successful instances, which validates our observation.

On the other hand, Map (a) and (b) show two typical patterns where PBS performs well. Map (a) contains long corridors but with more entry spaces in between. Map (b) contains many short obstacle components, providing more space for agents to avoid collisions. To validate our observation with long corridors, we run PBS in 200 MAPF instances on Map (a). The results indicate that PBS achieves an 87% success rate, with an average runtime of 3.4 seconds for successful cases, which indicates that PBS can effectively solve maps featuring long corridors with increased entry spaces in between.

PIBT. Figure 6 shows the archive of maps of PIBT. The success rate of PIBT decreases with an increased number of obstacles and KL divergence. Similar to EECBS and CBS, PIBT performs poorly in maps with long corridors, such as Map (c). Running PIBT 200 in MAPF instances on Map (c) resulted in an average success rate of 12%, validating

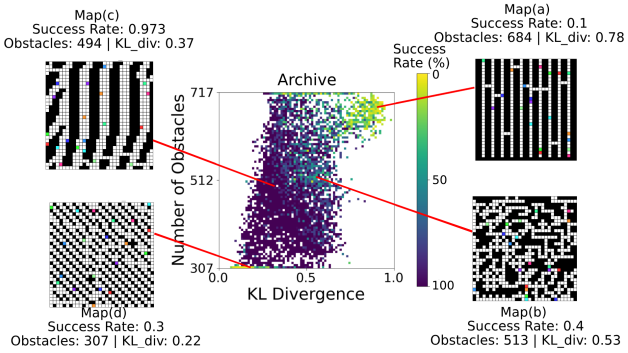


Figure 7: Archive for LTF with representative maps.

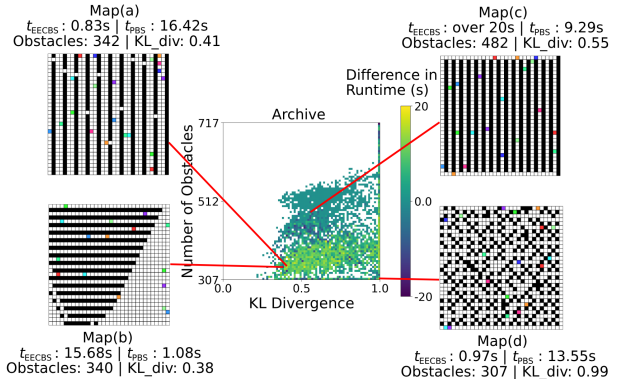


Figure 8: Archive for EECBS vs PBS with representative maps.

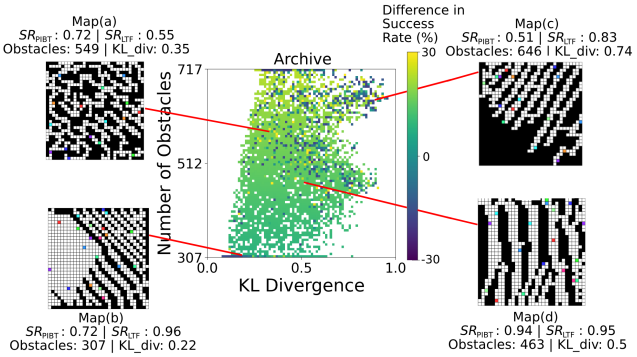


Figure 9: Archive for PIBT vs LTF with representative maps.

its poor performance. However, there are cases with large number of obstacles and low KL divergence in that PIBT has a high success rate. These maps share a common pattern: large chunks of empty spaces with few corridors. Map (a) is one such example. Running PIBT in 200 instances on Map (a) results in an average success rate of 96%. By inspecting the patterns in maps where PIBT has around a 50% success rate, we observe that the most common patterns are a combination of long corridors and one-entry spaces. Map (d) is an example. Map (b), on the other hand, has many one-entry spaces but fewer obstacles, achieving a success rate of 100%. Upon running PIBT in map (b) with 200 MAPF instances, PIBT achieves an average success rate of 95%. This result indicates that PIBT performs well when encountering one-entry spaces with fewer obstacles. Overall, PIBT can efficiently solve maps that include long corridors and one-entry spaces if given sufficient space.

LTF. Figure 7 shows the archive of maps for LTF. Similar to PIBT, LTF performs worse with more obstacles and higher KL divergence, as shown in Map (a). Similar to challenging maps for CBS, EECBS, and PIBT, Map (a) contains long corridors, which increase the chance of congestion at the entries. We use the manually designed maps with long corridors, same as the maps tested with PBS, to further evaluate the performance of LTF. The result shows that with a makespan of 512, the success rate for LTF is below 10%. With fewer obstacles, LTF performs poorly on maps with many one-entry spaces, such as Map (b) and Map (d). These two maps contain different numbers of obstacles, but they have similar patterns of one-entry spaces. Upon running LTF in 200 instances on each map, we observe an average success rate of 35%, revealing poor performance on maps with this pattern. On the other hand, LTF performs better with fewer obstacles. These maps possess large chunks of empty spaces, exemplified in Map (c).

5 Comparing Two Algorithms

Given the different categories and properties of the MAPF algorithms, hard maps for one algorithm might not be hard for the other. Therefore, for two-algorithm experiments, we aim to generate maps that are easy for one algorithm and hard for the other by maximizing the performance gap between them.

5.1 Experiment Setup

We run EECBS with $w = 1.5$. For EECBS and PBS, we use $T = 20$ seconds. For PIBT and LTF, we use $M = 512$, which is used to train the LTF policy (Skrynnik et al. 2024). We use the same O_{lb}, O_{ub}, N_e, N_{eval}, N_a and map sizes as the one-algorithm experiments. The implementation and compute resources are specified in Section 4.1.

5.2 Results

We show the archives and representative benchmark maps in Figures 8 and 9. To better visualize the comparisons of the algorithms, we plot the differences, instead of the absolute differences, between the CPU runtime or success rate of the algorithms in each map. In Figure 8, the yellow cells represent $t_{\text{EECBS}}(\boldsymbol{x}) - t_{\text{PBS}}(\boldsymbol{x}) \geq 5$ seconds, and the dark blue cells represent $t_{\text{PBS}}(\boldsymbol{x}) - t_{\text{EECBS}}(\boldsymbol{x}) \geq 5$ seconds. In Figure 9, the yellow cells represent $SR_{\text{PIBT}}(\boldsymbol{x}) - SR_{\text{LTF}}(\boldsymbol{x}) \geq 30\%$, and the blue cells represent $SR_{\text{LTF}}(\boldsymbol{x}) - SR_{\text{PIBT}}(\boldsymbol{x}) \geq 30\%$.

EECBS vs. PBS. Figure 8 shows the archive comparing EECBS and PBS. In most cases, EECBS and PBS have similar performances. However, we can identify their advantageous ranges clearly at the upper right corner and bottom left corner. Maps with more than 512 obstacles are usually too hard for both algorithms. Therefore, it is hard to generate maps that have a large difference in CPU runtime. With

fewer than 400 obstacles, EECBS generally performs better than PBS in two types of maps, exemplified in Map (a) and (d). Map (a) is similar to Map (a) of Figure 4, with long but wide corridors and one-entry spaces between adjacent corridors. Map (d) shows a relatively random pattern with many one-entry spaces. To validate the comparison, we run 200 MAPF instances in Map (a) and (d). In Map (a), EECBS has a higher success rate (95.5%) than PBS (69.5%), and a lower average runtime (4.3 seconds) than PBS (12.5 seconds). Similarly, in Map (d), EECBS has a higher success rate (100%) than PBS (77.5%), and a lower average runtime (1.13 seconds) than PBS (7.4 seconds).

Map (b) and (c) are examples where PBS outperforms EECBS. Notably, while Map (c) contains narrow long corridors, both ends of the corridors are open, potentially making PBS more easily resolve conflicts. Upon validating our results on 200 MAPF instances, in Map (b), PBS has a higher success rate (96.5%) than EECBS (56.5%), and a lower average runtime (2.0 seconds) than EECBS (9.8 seconds). In Map (c), PBS has a higher success rate (43%) than EECBS (14.5%), and a lower average runtime (13.7) than EECBS (18.3 seconds).

Our results serve as a more comprehensive comparison between EECBS and PBS. Notably, only one work (Chan et al. 2023) has systematically compared EECBS and PBS. They conclude that EECBS outperforms PBS in five out of six maps selected from the MAPF benchmark (Stern et al. 2019). In contrast, our results indicate that EECBS and PBS exhibit similar behavior on most maps, with each having distinct advantages on different map patterns.

PIBT vs. LTF. Figure 9 shows the archive comparing PIBT and LTF. In most cases with less than 50% of obstacles, PIBT and LTF have a similar success rate. In maps with large chunks of empty space such as Map (d), both PIBT and LTF perform well. With the increase of obstacles over 50%, in general, PIBT has a 5% to 10% higher success rate on average than LTF. In most cases, these maps contain many one-entry spaces, such as Map (a). Upon running 200 instances on Map (a) with PIBT and LTF, PIBT outperforms LTF by an average success rate of 13%. The result aligns with our observations in Section 4. PIBT has better performance in maps with many one-entry spaces compared to LTF.

On the other hand, the archive also shows many cases where LTF has a higher success rate than PIBT. These cases are dispersed throughout the archive, with notable concentrations in the upper right and bottom left. Map (b) and (c) in Figure 9 are the representative maps. Upon running 200 instances on both maps, LTF has an average of 20% higher success rate on these maps compared to PIBT. By examining the patterns in these maps, we find that a slight increase in the number of empty spaces between corridors can create a significant gap in success rates between LTF and PIBT.

Our results provide additional comparisons between PIBT and LTF to the experiments conducted by Skrynnik et al. (2024), which conclude that LTF outperforms PIBT by comparing them on a set of human-designed warehouse maps.

6 Discussion and Conclusion

We propose a framework based on QD algorithms to generate diverse benchmark maps for MAPF algorithms. We provide two concrete realizations of the framework, presenting experimental results with diverse benchmark maps of different levels of hardness and patterns. For the one-algorithm experiments, we generate hard benchmark maps for five representative MAPF algorithms by either maximizing the CPU runtime or minimizing the regularized success rate. For the two-algorithm experiments, we generate benchmark maps that are hard for one algorithm and easy for the other.

When and How to Use the Proposed Framework. While developing novel MAPF algorithms, researchers can use our one-algorithm pipeline with their desired objective to find maps their algorithm has difficulty solving. By generating such maps, they can have a better understanding of the weaknesses of their algorithms. In addition, researchers can leverage our two-algorithm pipeline to compare their proposed algorithm with an existing algorithm. They can observe which kinds of maps their algorithm performs better or worse than the existing algorithm. We argue that this is a more systematic way of comparing two algorithms than comparing them on a pre-defined set of fixed benchmark maps because researchers might intentionally or unintentionally cherry-pick maps that favor their algorithm or overfit the design of their algorithms to the selected set of maps.

Nevertheless, the one-algorithm and two-algorithm pipelines are not the only realizations of our framework. Researchers can tailor the framework for alternative purposes by designing the objectives and diversity measures. For example, researchers can minimize, instead of maximize, the CPU runtime of CBS, EECBS, and PBS with a larger number of agents to generate a diverse collection of maps that are *easy*, instead of hard, for these algorithms. While we present some easy maps for each selected algorithm in Section 4, they are by no means an exhaustive set. If easy maps are of interest, researchers should use alternative objectives, such as minimizing the CPU runtime or maximizing the success rate. If researchers are interested in both hard and easy maps, using the CPU runtime or regularized success rate as diversity measures is a better choice.

Researchers can also generalize the two-algorithm pipeline to an arbitrary number of MAPF algorithms to rank the performance of more than two algorithms. In addition, our framework focuses on generating MAPF benchmark *maps*. Future research can explore generating diverse MAPF *instances* by incorporating numbers of agents, map sizes, as well as start and goal locations into the pipeline.

Computational Cost. Our proposed framework relies on QD algorithms, which require a massive number of evaluations in the generated benchmark maps to compute the objective and diversity measure values. Specifically, with our one-algorithm experiment setup of $N_{eval} = 10,000$ evaluations, each with $N_e = 5$ runs in different MAPF instances, it takes 6 to 7 hours to generate benchmark maps for PIBT and LTF, 10 hours for PBS, and more than 12 hours for CBS and EECBS. The runtime is measured in machine (1) specified in Section 4.1. Future works can focus on reducing the computational cost of QD algorithms to make the pipeline

more efficient. For example, prior works (Zhang et al. 2022; Bhatt et al. 2022, 2023) have explored training a data-driven surrogate model to assist the QD search. Meanwhile, we will release our most representative generated benchmark maps online. If the researchers' new algorithm is built on one of the algorithms we have tested, they can directly evaluate new algorithms on our benchmark maps.

Acknowledgments

The research at Carnegie Mellon University was supported by the National Science Foundation (NSF) under grant number #2328671 and a gift from Amazon. The work used Bridge-2 at Pittsburgh Supercomputing Center (PSC) through allocation CIS220115 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by NSF under grant numbers #2138259, #2138286, #2138307, #2137603, and #2138296. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, or the U.S. government. We also thank Daniel Harabor, Peter J. Stuckey, Nathan Sturtevant, as well as reviewers at AAAI 2025 for their valuable feedback on this paper.

References

Abeysirigoonawardena, Y.; Shkurti, F.; and Dudek, G. 2019. Generating Adversarial Driving Scenarios in High-Fidelity Simulators. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 8271–8277.

Arnold, J.; and Alexander, R. 2013. Testing Autonomous Robot Control Software Using Procedural Content Generation. In *Proceedings of International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*, 33–44.

Barer, M.; Sharon, G.; Stern, R.; and Felner, A. 2014. Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem. In *Proceedings of the Annual Symposium on Combinatorial Search (SoCS)*, 19–27.

Bhatt, V.; Nemlekar, H.; Fontaine, M. C.; Tjanaka, B.; Zhang, H.; Hsu, Y.-C.; and Nikolaidis, S. 2023. Surrogate Assisted Generation of Human-Robot Interaction Scenarios. In *Proceedings of the Conference of Robot Learning (CoRL)*, 513–539.

Bhatt, V.; Tjanaka, B.; Fontaine, M.; and Nikolaidis, S. 2022. Deep Surrogate Assisted Generation of Environments. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 37762–37777.

Chan, S.-H.; Stern, R.; Felner, A.; and Koenig, S. 2023. Greedy Priority-Based Search for Suboptimal Multi-Agent Path Finding. In *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 11–19.

Choudhury, S.; Solovey, K.; Kochenderfer, M.; and Pavone, M. 2022. Coordinated Multi-Agent Pathfinding for Drones and Trucks over Road Networks. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 272–280.

Cobbe, K.; Hesse, C.; Hilton, J.; and Schulman, J. 2020. Leveraging procedural generation to benchmark reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2048–2056.

Cully, A.; Clune, J.; Tarapore, D.; and Mouret, J.-B. 2015. Robots that can adapt like animals. *Nature*, 521(7553): 503–507.

Damani, M.; Luo, Z.; Wenzel, E.; and Sartoretti, G. 2021. PRIMAL$_2$: Pathfinding Via Reinforcement and Imitation Multi-Agent Learning - Lifelong. *IEEE Robotics and Automation Letters*, 6(2): 2666–2673.

Erdmann, M. A.; and Lozano-Pérez, T. 1987. On Multiple Moving Objects. *Algorithmica*, 2: 477–521.

Ewing, E.; Ren, J.; Kansara, D.; Sathiyanarayanan, V.; and Ayanian, N. 2022. Betweenness Centrality in Multi-Agent Path Finding. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 400–408.

Fontaine, M.; and Nikolaidis, S. 2023. Covariance Matrix Adaptation MAP-Annealing. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 456–465.

Fontaine, M. C.; Hsu, Y.-C.; Zhang, Y.; Tjanaka, B.; and Nikolaidis, S. 2021a. On the Importance of Environments in Human-Robot Coordination. In *Proceedings of the Robotics: Science and Systems (RSS)*.

Fontaine, M. C.; Liu, R.; Khalifa, A.; Modi, J.; Togelius, J.; Hoover, A. K.; and Nikolaidis, S. 2021b. Illuminating Mario Scenes in the Latent Space of a Generative Adversarial Network. *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 5922–5930.

Friedrich, P.; Zhang, Y.; Curry, M.; Dierks, L.; McAleer, S.; Li, J.; Sandholm, T.; and Seuken, S. 2024. Scalable Mechanism Design for Multi-Agent Path Finding. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 58–66.

Gange, G.; Harabor, D.; and Stuckey, P. J. 2019. Lazy CBS: Implicit Conflict-Based Search Using Lazy Clause Generation. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 155–162.

Gardner, M. 1970. Mathematical Games – The Fantastic Combinations of John Conway's New Solitaire Game "Life". *Scientific American*, 223(4): 120–123.

Hansen, N. 2016. The CMA Evolution Strategy: A Tutorial. *ArXiv*, abs/1604.00772.

Ho, F.; Gonçalves, A.; Rigault, B.; Geraldes, R.; Chicharo, A.; Cavazza, M.; and Prendinger, H. 2022. Multi-Agent Path Finding in Unmanned Aircraft System Traffic Management With Scheduling and Speed Variation. *IEEE Intelligent Transportation Systems Magazine*, 14(5): 8–21.

Jansen, M. R.; and Sturtevant, N. R. 2008. Direction Maps for Cooperative Pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 185–190.

Lam, E.; Le Bodic, P.; Harabor, D.; and Stuckey, P. J. 2019. Branch-and-Cut-and-Price for Multi-Agent Pathfinding. In

Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 1289–1296.

Lehman, J.; and Stanley, K. O. 2011a. Abandoning Objectives: Evolution Through the Search for Novelty Alone. *Evolutionary Computation*, 19(2): 189–223.

Lehman, J.; and Stanley, K. O. 2011b. Evolving a Diversity of Virtual Creatures through Novelty Search and Local Competition. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 211–218.

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

Li, J.; Gange, G.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2020. New Techniques for Pairwise Symmetry Breaking in Multi-Agent Path Finding. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 193–201.

Li, J.; Ruml, W.; and Koenig, S. 2021. EECBS: A Bounded-Suboptimal Search for Multi-Agent Path Finding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 12353–12362.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. K. S.; and Koenig, S. 2021b. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 11272–11281.

Ma, H.; Harabor, D.; Stuckey, P. J.; Li, J.; and Koenig, S. 2019. Searching with Consistent Prioritization for Multi-Agent Path Finding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 7643–7650.

Ma, H.; Yang, J.; Cohen, L.; Kumar, T. K. S.; and Koenig, S. 2017. Feasibility Study: Moving Non-Homogeneous Teams in Congested Video Game Environments. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 270–272.

Mouret, J.-B.; and Clune, J. 2015. Illuminating Search Spaces by Mapping Elites. *ArXiv*, abs/1504.04909.

Mullins, G. E.; Stankiewicz, P. G.; Hawthorne, R. C.; and Gupta, S. K. 2018. Adaptive Generation of Challenging Scenarios for Testing and Evaluation of Autonomous Vehicles. *Journal of Systems and Software*, 137: 197–215.

Okumura, K. 2023. LaCAM: Search-Based Algorithm for Quick Multi-Agent Pathfinding. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 11655–11662.

Okumura, K.; Machida, M.; Défago, X.; and Tamura, Y. 2019. Priority Inheritance with Backtracking for Iterative Multi-agent Path Finding. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 535–542.

Pearl, J.; and Kim, J. H. 1982. Studies in Semi-Admissible Heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4(4): 392–399.

Ren, J.; Ewing, E.; Kumar, T. K. S.; Koenig, S.; and Ayanian, N. 2024. Map Connectivity and Empirical Hardness of Grid-based Multi-Agent Pathfinding Problem. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 484–488.

Sartoretti, G.; Kerr, J.; Shi, Y.; Wagner, G.; Kumar, T. K. S.; Koenig, S.; and Choset, H. 2019. PRIMAL: Pathfinding via Reinforcement and Imitation Multi-Agent Learning. *IEEE Robotics and Automation Letters*, 4(3): 2378–2385.

Shaoul, Y.; Mishani, I.; Likhachev, M.; and Li, J. 2024. Accelerating Search-Based Planning for Multi-Robot Manipulation by Leveraging Online-Generated Experiences. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 523–531.

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.

Shervashidze, N.; Schweitzer, P.; van Leeuwen, E. J.; Mehlhorn, K.; and Borgwardt, K. M. 2011. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research*, 12(77): 2539–2561.

Skrynnik, A.; Andreychuk, A.; Nesterova, M.; Yakovlev, K.; and Panov, A. 2024. Learn to Follow: Decentralized Lifelong Multi-Agent Pathfinding via Planning and Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 17541–17549.

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 International Symposium on Combinatorial Search (SoCS)*, 151–159.

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

Thayer, J. T.; and Ruml, W. 2011. Bounded Suboptimal Search: A Direct Approach Using Inadmissible Estimates. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 674–679.

Tjanaka, B.; Fontaine, M. C.; Lee, D. H.; Zhang, Y.; Balam, N. R.; Dennler, N.; Garlanka, S. S.; Klapsis, N. D.; and Nikolaidis, S. 2023. pyribs: A Bare-Bones Python Library for Quality Diversity Optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 220–229.

Varambally, S.; Li, J.; and Koenig, S. 2022. Which MAPF Model Works Best for Automated Warehousing? In *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 190–198.

Wagner, G.; and Choset, H. 2011. M*: A Complete Multirobot Path Planning Algorithm with Performance Bounds. In *IROS*, 3260–3267.

Wagner, G.; and Choset, H. 2015. Subdimensional Expansion for Multirobot Path Planning. *Artificial Intelligence*, 219: 1–24.

Wang, K. C.; and Botea, A. 2011. MAPP: A Scalable Multi-Agent Path Planning Algorithm with Tractability and Completeness Guarantees. *Journal of Artificial Intelligence Research*, 42: 55–90.

Wang, R.; Lehman, J.; Clune, J.; and Stanley, K. O. 2019. Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions. *ArXiv*, abs/1901.01753.

Wang, R.; Lehman, J.; Rawal, A.; Zhi, J.; Li, Y.; Clune, J.; and Stanley, K. O. 2020. Enhanced POET: Open-Ended Reinforcement Learning through Unbounded Invention of Learning Challenges and their Solutions. In *Proceedings of the International Conference on Machine Learning (ICML)*, 9940–9951.

Zhang, H.; Fontaine, M. C.; Hoover, A. K.; Togelius, J.; Dilkina, B.; and Nikolaidis, S. 2020. Video Game Level Repair via Mixed Integer Linear Programming. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 151–158.

Zhang, Y.; Fontaine, M. C.; Bhatt, V.; Nikolaidis, S.; and Li, J. 2023a. Arbitrarily Scalable Environment Generators via Neural Cellular Automata. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 57212–57225.

Zhang, Y.; Fontaine, M. C.; Bhatt, V.; Nikolaidis, S.; and Li, J. 2023b. Multi-Robot Coordination and Layout Design for Automated Warehousing. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 5503–5511.

Zhang, Y.; Fontaine, M. C.; Hoover, A. K.; and Nikolaidis, S. 2022. Deep Surrogate Assisted MAP-Elites for Automated Hearthstone Deckbuilding. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 158–167.

Zhang, Y.; Jiang, H.; Bhatt, V.; Nikolaidis, S.; and Li, J. 2024. Guidance Graph Optimization for Lifelong Multi-Agent Path Finding. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 311–320.

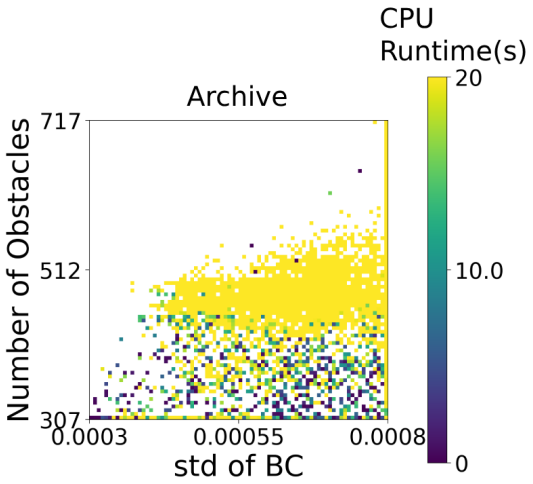


Figure 10: Archive of PBS with the number of obstacles and std of BC as diversity measures.

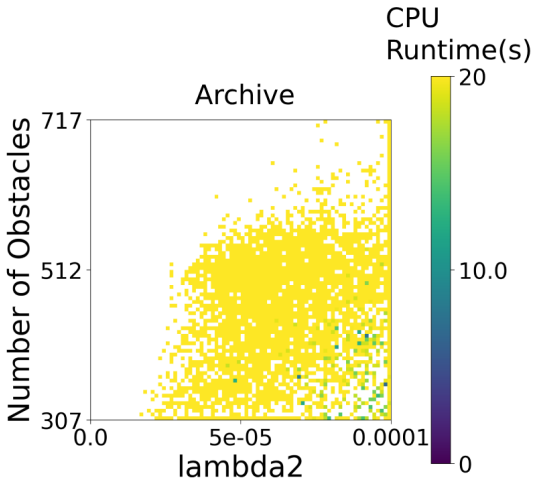


Figure 11: Archive of PBS with the number of obstacles and λ_2 as diversity measures.

A Choosing Diversity Measures

In this section, we justify our choice of diversity measures by setting up the archive with every two pairs of competitive diversity measures and exploring their relationship and behavior. We show the following experiments: (1) we compare the number of obstacles and standard deviation of Betweenness Centrality (std of BC) (Ewing et al. 2022), the number of obstacles and λ_2 (Ren et al. 2024), and show that the number of obstacles could quantify the general hardness of the generated maps better than std of BC and λ_2. (2) We compare the KL divergence of the tile pattern distribution and map entropy, the KL divergence of the tile pattern distribution and the KL divergence of the WL graph feature, and show that the KL divergence of the tile pattern distribution can generate more complex patterns than map entropy and have less computational cost than the KL divergence of WL graph feature.

A.1 General Hardness

Number of Obstacles vs. Std of BC. We run the one-algorithm experiment of PBS with the number of obstacles and std of BC as the diversity measures to explore the relationship between the two measures and their correlation with the general hardness of generated maps. From Figure 10, we observe that in the y-axis, with the increase of obstacles, the CPU runtime of PBS is increasing, which is consistent with the one-algorithm experiment result of PBS in Section 4 of the main text. With fewer variations in the CPU runtime of PBS at a given number of obstacles, we can quantify the general hardness of the maps based on the number of obstacles. From the x-axis, however, there are many maps with the same std of BC but showing different general hardness of the maps for PBS. Therefore, the std of BC is not a good measure to quantify the general hardness of the maps.

Number of obstacles vs. λ_2. We run the one-algorithm experiment of PBS with the number of obstacles and λ_2 as the diversity measures to explore the relationship between the two measures and their correlation with the hardness of the generated maps. From Figure 11, we observe that most of the generated maps are hard for PBS to solve. In this case, the QD search is close to convergence, revealing that applying each diversity measure can efficiently generate challenging maps for PBS. Therefore, both diversity measures can quantify the general hardness of generated maps. We use the number of obstacles in our experiments due to its computational efficiency and intuitive interpretation.

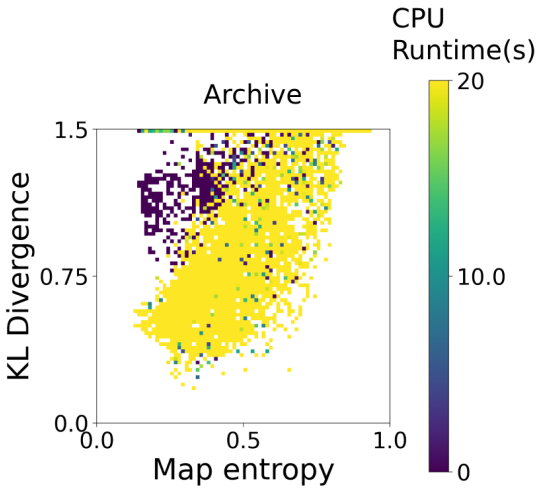


Figure 12: Archive of PBS with map entropy and KL divergence of the tile pattern distribution as diversity measures.

A.2 Spatial Arrangement

KL Divergence of Tile Pattern Distribution vs. Map Entropy. We run a one-algorithm experiment with PBS. We use the same objective in Section 4 and use map entropy and KL divergence of the tile pattern distribution as the diversity measures. Figure 12 shows the resulting archive of maps.

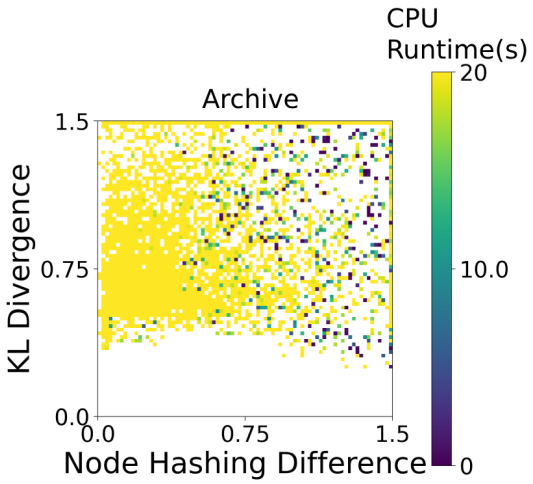


Figure 13: Archive of PBS with KL divergence of the tile pattern distribution and KL divergence of WL graph feature as diversity measure.

We observe that, with map entropy less than 0.5, the generated maps are usually composed of a large chunk of empty spaces with a large component of obstacles on the boundary, revealing a simple structure. Compared to maps generated by the KL divergence of the tile distribution in Figure 5 of Section 4, maps generated by map entropy have less diverse patterns. Therefore, we use the KL divergence of tile pattern distribution as the diversity measure.

KL Divergence of Tile Pattern Distribution vs. KL Divergence of WL Graph Feature. We run the one-algorithm experiment of PBS with diversity measures of the KL divergence of the tile pattern distribution and the KL divergence of the WL graph feature to explore their correlation and the spatial arrangements in generated maps. From Figure 13, we observe that all generated maps are well-distributed across the archive, revealing a similar distribution of generated maps with different pairs of KL divergence of the tile pattern distribution and the KL divergence of WL graph feature. Therefore, these two diversity measures are similar. Both quantify the similarity between the generated maps and the pre-defined set of maze maps (Stern et al. 2019). We use the KL divergence of the tile pattern distribution in our experiments since the WL graph feature needs a graph transformation and several iterations of hash transformation, which is more computationally expensive.

B Maps with similar patterns

Figure 14 shows the easy maps of CBS, similar to the maps shown in Figure 3 in Section 4 in the main text.

Figure 15 shows the challenging and easy maps of EECBS, similar to the maps shown in Figure 4 in Section 4 in the main text.

Figure 16 shows the challenging and easy maps of PBS, similar to the maps shown in Figure 5 in Section 4 in the main text.

Figure 17 shows the challenging and easy maps of PIBT, similar to the maps shown in Figure 6 in Section 4 in the main text.

Figure 18 shows the challenging and easy maps of LTF, similar to the maps shown in Figure 7 in Section 4 in the main text.

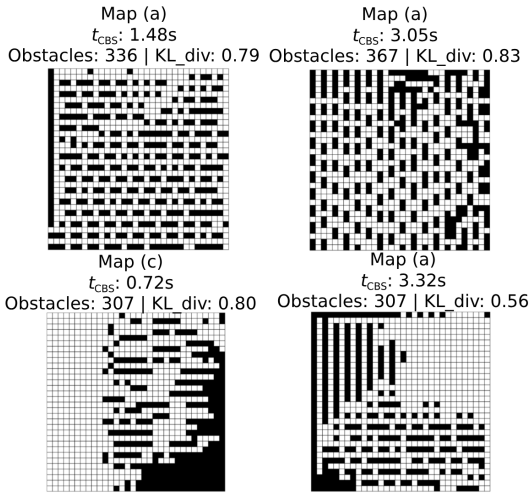


Figure 14: Similar maps with evenly distributed short corridor components or large chunks of empty space on which CBS performs well. Maps (a) and (b) are maps with evenly distributed short corridor components. Map (c) is the map with large chunks of empty space. Map (d) is the map with both evenly distributed short corridor components and large chunks of empty space, but also long corridors on the left.

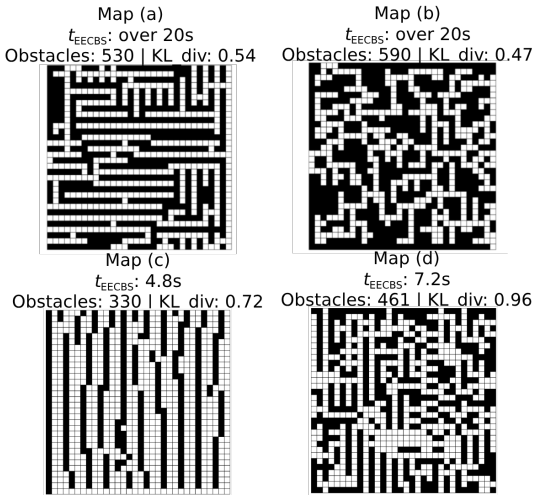


Figure 15: Map (a) and Map (b) are maps similar to representative maps with long corridors and one-tile spaces on which EECBS performs poorly. Map (c) and Map (d) are maps similar to representative maps with two or more columns of empty space between each long obstacle component and short obstacle components on which EECBS performs well.

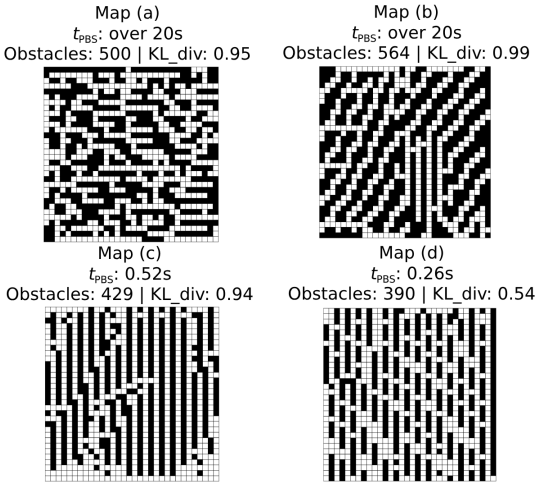


Figure 16: Map (a) and Map (b) are maps similar to representative maps with one-tile entries between corridors on which PBS performs poorly. Map (c) and Map (d) are maps similar to representative maps of long corridors with more entry spaces and short obstacle components on which PBS performs well.

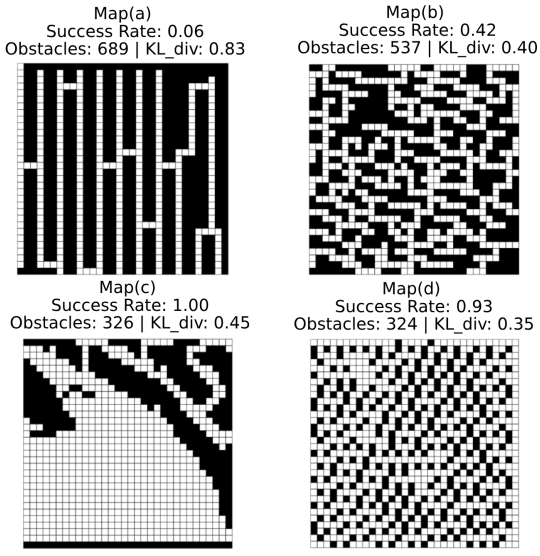


Figure 17: Map (a) and Map (b) are maps similar to representative maps with long corridors and one-entry spaces with many obstacles on which PIBT performs poorly. Map (c) and Map (d) are maps similar to representative maps with large chunks of empty space and one-entry spaces with fewer obstacles on which PIBT performs well.

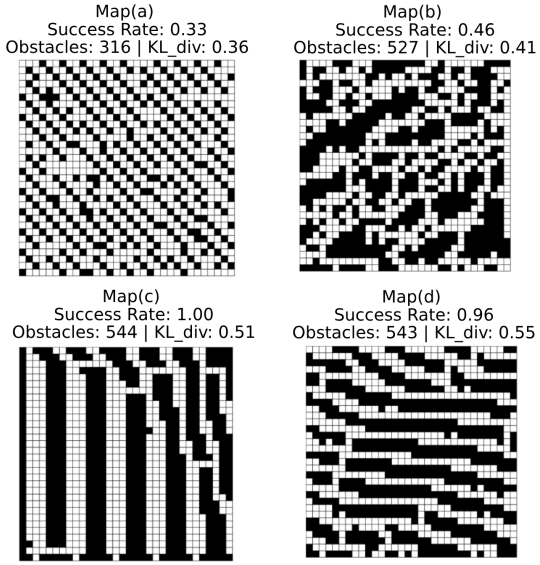


Figure 18: Map (a) and Map (b) are maps similar to representative maps with long corridors and one-entry spaces on which LTF performs poorly. Map (c) and Map (d) are maps similar to representative maps with fewer obstacles on which LTF performs well.