Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks

Roni Stern,[1] **Nathan R. Sturtevant,**[2] **Ariel Felner,**[1] **Sven Koenig,**[3] **Hang Ma,**[3]
Thayne T. Walker,[4] **Jiaoyang Li,**[3] **Dor Atzmon,**[1] **Liron Cohen,**[3] **T. K. Satish Kumar,**[3] **Eli Boyarski,**[1] **Roman Barták**[5]
[1]Ben Gurion University of the Negev, [2]University of Alberta, [3]USC, [4]University of Denver [5]Charles University
sternron@post.bgu.ac.il, nathanst@ualberta.ca, felner@bgu.ac.il, skoenig@usc.edu, hangma@usc.edu, thayne.walker@du.edu,
jiaoyanl@usc.edu, dorat@post.bgu.ac.il, lironcoh@usc.edu, tkskwork@gmail.com, boyarske@post.bgu.ac.il, bartak@ktiml.mff.cuni.cz

Abstract

The Multi-Agent Pathfinding (MAPF) problem is the fundamental problem of planning paths for multiple agents, where the key constraint is that the agents will be able to follow these paths concurrently without colliding with each other. Applications of MAPF include automated warehouses and autonomous vehicles. Research on MAPF has been flourishing in the past couple of years. Different MAPF research papers make different assumptions, e.g., whether agents can traverse the same road at the same time, and have different objective functions, e.g., minimize makespan or sum of agents' actions costs. These assumptions and objectives are sometimes implicitly assumed or described informally. This makes it difficult to establish appropriate baselines for comparison in research papers, as well as making it difficult for practitioners to find the papers relevant to their concrete application. This paper aims to fill this gap and support researchers and practitioners by providing a unifying terminology for describing common MAPF assumptions and objectives. In addition, we also provide pointers to two MAPF benchmarks. In particular, we introduce a new grid-based benchmark for MAPF, and demonstrate experimentally that it poses a challenge to contemporary MAPF algorithms.

1 Introduction

MAPF is an important type of multi-agent planning problem in which the task is to plan paths for multiple agents, where the key constraint is that the agents will be able to follow these paths concurrently without colliding with each other. MAPF has a range of relevant contemporary applications including automated warehouses, autonomous vehicles, and robotics. Consequently, this problem has received attention in recent years from various research groups and academic communities (**?**; **?**; **?**; **?**; **?**; **?**; **?**).

Different MAPF research papers consider different sets of assumptions about the agents and aim for different objectives. These assumptions and objectives are sometimes implicitly assumed or described informally. Even in cases where the assumptions and objective function are described formally, there are still differences in used MAPF terminology. This makes it difficult to navigate through and understand existing literature and to establish appropriate baselines for comparison. In addition, it makes it difficult for practitioners to find papers relevant to their concrete application.

This paper aims to address this growing challenge by introducing a **unified terminology** to describe MAPF problems, and by establishing **common benchmarks and evaluation measures** for evaluating MAPF algorithms. The unified MAPF terminology we present in this paper is our attempt to classify the currently studied MAPF variants. We hope this terminology will serve as a common ground for future researchers, and will be used by them to describe their contributions succinctly and accurately.

In the second part of this paper, we introduce a new grid MAPF benchmark to the community. This benchmark includes a diverse set of maps, as well as generated source and target vertices. We report the performance of a standard MAPF algorithm on this benchmark, to serve as baseline for comparison to future research. This benchmark is intended to help future researchers and enable more scientifically rigorous empirical comparisons of existing and future MAPF algorithms. We do not claim that these benchmarks are perfect, as they may have some biases. But, through their use and study these biases can be discovered and corrected. It is also important to emphasize that this document is not intended to be a survey of state of the art MAPF algorithms. For such a survey, see (**?**; **?**). In addition, the newly created website http://mapf.info contains MAPF-related tutorials and other resources.

2 Classical MAPF

We first describe what we refer to as a *classical MAPF* problem. The input to a classical MAPF problem with k agents is a tuple $\langle G, s, t \rangle$ where $G = (V, E)$ is an undirected graph, $s : [1, \ldots, k] \rightarrow V$ maps an agent to a source vertex, and $t : [1, \ldots, k] \rightarrow V$ maps an agent to a target vertex. Time is assumed to be discretized, and in every time step each agent is situated in one of the graph vertices and can perform a single *action*. An action in classical MAPF is a function $a : V \rightarrow V$ such that $a(v) = v'$ means that if an agent is at vertex v and performs a then it will be in vertex v' in the next time step. Each agent has two types of actions: *wait* and

move. A *wait* action means that the agent stays in its current vertex another time step. A *move* action means that the agent moves from its current vertex v to an adjacent vertex v' in the graph (i.e., $(v, v') \in E$).

For a sequence of actions $\pi = (a_1, \dots a_n)$ and an agent i, we denote by $\pi_i[x]$ the location of the agent after executing the first x actions in π, starting from the agent's source $s(i)$. Formally, $\pi_i[x] = a_x(a_{x-1}(\cdots a_1(s(i))))$. A sequence of actions π is a **single-agent plan** for agent i iff executing this sequence of actions in $s(i)$ results in being at $t(i)$, that is, iff $\pi_i[|\pi|] = t(i)$. A **solution** is a set of k single-agent plans, one for each agent.

2.1 Types of Conflicts in Classical MAPF

The overarching goal of MAPF solvers is to find a solution, i.e., a single-agent plan for each agent, that can be executed without collisions. To achieve this, MAPF solvers use the notion of *conflicts* during planning, where a MAPF solution is called *valid* iff there is no conflict between any two single-agent plans. The definition of what constitutes a conflict depends on the environment, and correspondingly the literature on classical MAPF includes several different definitions of what constitutes a conflict between plans. We list common conflict definitions below. Let π_i and π_j be a pair of single-agent plans.

- **Vertex conflict.** A *vertex conflict* between π_i and π_j occurs iff according to these plans the agents are planned to occupy the same vertex at the same time step. Formally, there is a vertex conflict between π_i and π_j iff there exists a time step x such that $\pi_i[x] = \pi_j[x]$.
- **Edge conflict.** An *edge conflict* between π_i and π_j occurs iff according to these plans the agents are planned to traverse the same edge at the same time step in the same direction. Formally, there is an edge conflict between π_i and π_j iff there exists a time step x such that $\pi_i[x] = \pi_j[x]$ and $\pi_i[x+1] = \pi_j[x+1]$.
- **Following conflict.** A *following* conflict between π_i and π_j occurs iff one agent is planned to occupy a vertex that was occupied by another agent in the previous time step. Formally, there is a following conflict between π_i and π_j iff there exists a time step x such that $\pi_i[x+1] = \pi_j[x]$.
- **Cycle conflict.** A *cycle* conflict between a set of single-agent plans $\pi_i, \pi_{i+1}, \dots \pi_j$ occurs iff in the same time step every agent moves to a vertex that was previously occupied by another agent, forming a "rotating cycle" pattern. Formally, a *cycle conflict* between a set of plans $\pi_i, \pi_{i+1}, \dots \pi_j$ occurs iff there exists a time step x in which $\pi_i(x+1) = \pi_{i+1}(x)$ and $\pi_{i+1}(x+1) = \pi_{i+2}(x)$ $\dots$ and $\pi_{j-1}(x+1) = \pi_j(x)$ and $\pi_j(x+1) = \pi_i(x)$.
- **Swapping conflict.** A *swapping* conflict between π_i and π_j occurs iff the agents are planned to swap locations in a single time step. Formally, there is a swapping conflict between π_i and π_j iff there exists a time step x such that $\pi_i[x+1] = \pi_j[x]$ and $\pi_j[x+1] = \pi_i[x]$. This conflict is sometimes called *edge conflict* in the current MAPF literature.

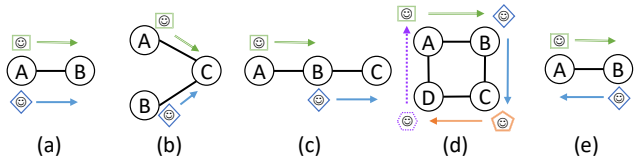


Figure 1: An illustration of common types of conflicts. From left to right: an edge conflict, a vertex conflict, a following conflict, a cycle conflict, and a swapping conflict.

Figure 1 illustrates the different types of conflicts. Note that the above set of conflict definitions is certainly not a complete set of all possible conflicts. Considering the formal definitions of these conflicts, it is clear that there are dominance relation between them: (1) forbidding vertex conflicts implies edge conflicts are also forbidden, (2) forbidding following conflicts implies cycle conflicts and swapping conflicts are also forbidden, (3) forbidding cycle conflicts implies that swapping conflicts are also forbidden. Vice versa, (1) allowing edge conflicts implies vertex conflicts are also allowed, (2) allowing swapping conflicts implies cycle conflicts are also allowed,[1] and (3) allowing cycle conflicts implies following conflicts are also allowed.

To properly define a classical MAPF problem, one needs to specify which types of conflicts are allowed in a solution. The least constrained restriction is to only forbid edge conflicts. However, to the best of our knowledge, all prior work on classical MAPF also forbid vertex conflicts. Some work on MAPF with payload transfers allows swapping conflicts (**?**). Most work on search-based MAPF algorithms (**?**; **?**) forbid swapping conflicts, but allow following conflicts. Some work on compilation-based MAPF algorithms as well as all work that consider MAPF as a pebble motion problem, forbid following conflicts as well (**?**; **?**).

2.2 Agent Behavior at Target in Classical MAPF

In a solution to a classical MAPF problem, agents may reach their targets at different time steps. Therefore, when defining a classical MAPF problem one must define how an agent behaves in the time steps after it has reached its target and before the last agent has reached its target.

There are two common assumptions for how agents behave at their targets.

- **Stay at target** Under this assumption, an agent waits in its target until all agents have reached their targets. This waiting agent will cause a vertex conflict with any plan that passes through its target after it has reached it. Formally, under the stay-at-target assumption, a pair of single-agent plans π_i and π_j will have a vertex conflict if there exists a time step $t \geq |\pi_i|$ such that $\pi_j[t] = \pi_i[|\pi_i|]$.
- **Disappear at target** Under this assumption, when an agent reaches its target it immediately disappears. This means the plan of that agent will not have any conflict after the time step in which the corresponding agent has reached its target.

[1] A swapping conflict is, in fact, a cycle conflict for two agents.

Most prior work on classical MAPF assumed stay-at-target, but recent work also considered the disappear-at-target assumption (**?**).

2.3 Objective Functions in Classical MAPF

It is safe to say that in most real applications of MAPF, some MAPF solutions are better than others. To capture that, work in classical MAPF considers an **objective function** that is used to evaluate MAPF solutions. The two most common functions used for evaluating a solution in classical MAPF are *makespan* and *sum of costs*.

- **Makespan.** The number of time steps required for all agents to reach their target. For a MAPF solution $\pi = \{\pi_1, \dots \pi_k\}$, the makespan of π is defined as $\max_{1 \leq i \leq k} |\pi_i|$.
- **Sum of costs.** The sum of time steps required by each agent to reach its target. The sum of costs of π is defined as $\sum_{1 \leq i \leq k} |\pi_i|$. Sum of costs is also known as *flowtime*.

If the agent-at-target behavior is *stay at target* and the objective function is *sum of costs*, then one needs to specify how staying at a target affects the sum of costs. For example, one can define that, if an agent waits at its target, then it does not increase the sum of costs. The common assumption in most prior work is that an agent staying in its target counts as a wait action unless it is not planning to move away from its target again. For example, assume that agent i reaches its target at time step t, leaves its target at time step t', arrives back at its target at time step t'', and then stays at its target until all agents reach their target. Then, this single-agent plan will contribute t'' to the sum of costs of the corresponding solution.

We do not claim that these are the only possible objective functions for classical MAPF. One may define other objective functions, such as the total non-waiting actions required to reach the target (some refer to this as the sum-of-fuel), and total time spent by the agent not in the target. However, to the best of our knowledge, the above objective functions are the only ones used in prior work on classical MAPF. Makespan has been used extensively by compilation-based MAPF algorithms, while sum of costs has been used by most search-based MAPF algorithms. But, there has also been work on both objective functions by both types of MAPF algorithms (**?**). There has also been work on maximizing the number of agents reaching their targets within a given makespan (i.e., deadline) (**?**).

3 Beyond Classical MAPF

All the above classical MAPF variants share the following assumptions: (1) time is discretized into time steps, (2) every action takes exactly one time step, and (3) in every time step, each agent occupies exactly a single vertex.

Next, we briefly list several MAPF variants that relax these assumptions.

3.1 MAPF on Weighted Graphs

The assumption that each action – move or wait – takes exactly one time step, implicitly assumes a somewhat simplistic motion model for the agents. More complex motion models have been studied in the MAPF literature, in which different actions may have different duration. This means the underlying graph that represents the possible locations agents may occupy (denoted G earlier) is now a weighted graph, where the weight of each edge represents the duration it will take an agent to traverse this edge.[2]

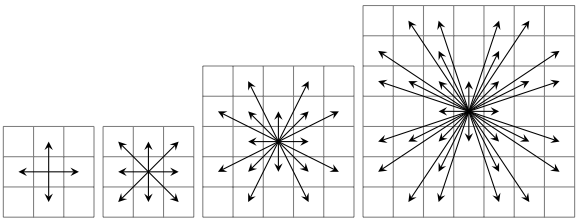


Figure 2: 2^k Neighborhood movement models for k = 2,3,4 and 5.

Bartak et al. (**?**) proposed a scheduling-based approach for MAPF on weighted graphs, and Walker et al. (**?**) proposed a variant of the Increasing Cost Tree Search (ICTS) algorithm. Yakovlev and Andreychuk (**?**) proposed a hybrid of the SIPP algorithm (**?**) and prioritized planning for weighted graphs.

The types of weighted graphs that have been used in MAPF reseach so far include:

- **MAPF in 2^k-neighbor grids.**[3] Such maps are a restricted form of weighted graphs in which every vertex represents a cell in a two-dimensional grid. The move actions of an agent in a cell are all its 2^k neighboring cells, where k is a parameter. Costs are based on Euclidean distance, therefore when $k > 2$, this introduces actions with different costs. For example, in an 8-neighbor grid a diagonal move costs $\sqrt{2}$ while a move in one of the cardinal directions costs 1. Figure 2 shows the possible move actions in 2^k-neighbor grids for $k = 2, 3, 4,$ and 5.
- **MAPF in Euclidean space.** MAPF in Euclidean space is a generalization of MAPF in which every node in G represents a Euclidean point (x, y), and the edges represent allowed move actions. Such settings arise, for example, when the underlying graph is a roadmap generated for a continuous Euclidean environment (**?**; **?**).

3.2 Feasibility Rules

The definition of a valid solution used in classical MAPF – no conflicts – is just one type of solution requirement. We use the term *feasibility rule* to refer to a requirement over a MAPF solution. Other MAPF feasibility rules have been suggested.

- **Robustness rules.** These are rules designed to ensure that a MAPF solution considers inadvertent delays in execution. A *k-robust* MAPF plan builds in a sufficient buffer for agents to be delayed up to k time steps without resulting in a conflict (**?**). When the probability of future delays

[2] One can also differentiate between the time it takes to traverse an edge and the cost it incurs. E.g., it may take one time step to traverse an edge but it may cost more, for example, energy.

[3] Such grids are also referred to as 2^k-connected grids.

is known, robustness rules can require that the probability an agent will conflict during execution is lower than a given bound (**?**) or be combined with execution policies to guarantee a conflict-free execution (**?**).

- **Formation rules.** These are restrictions over the allowed move actions of an agent that depend on the location of the other agents but are not related to collisions. For example, restrictions intended for the agents to maintain a specified formation (**?**), or to maintain a communication link with a set of neighboring agents (**?**; **?**).

3.3 From Pathfinding to Motion Planning

In classical MAPF, agents are assumed to occupy exactly one vertex, in a sense having no volume, no shape, and move at constant speed. By contrast, motion planning algorithms directly consider these properties. There, an agent is situated at each time step in a *configuration* instead of only a vertex, where a configuration specifies the agent's location, orientation, velocity, etc, and an edge between configurations represents kinematic motion. Several notable MAPF variants are steps towards closing this gap between classical MAPF and motion planning.

MAPF with large agents. Some MAPF research considered agents with a specific geometric shape and volume (**?**; **?**; **?**; **?**). The fact that agents have volume raises questions about how they are situated in the underlying graph G and how they move in it. In particular, if an agent is located in one vertex, it may prohibit other agents from occupying nearby vertices. Similarly, if an agent moves along an edge it may prohibit other agents from moving along intersecting edges or staying at vertices that are too close to the edge. This may introduce new types of conflicts, such as vertex-to-vertex, edge-to-edge, and edge-to-vertex conflicts (**?**).

Several approaches for solving MAPF with large agents have appeared in the literature, including a CBS-based approach (**?**), an ICTS-based approach (**?**), and a prioritized planning approach (**?**). A special case of agents with volume is the *convoy* setting, in which agents occupy a string of vertices and their connecting edges (**?**).

MAPF with kinematic constraints. Other MAPF research considered kinematic constraints over agents' move actions (**?**; **?**). That is, the move actions an agent can perform depend not only on its current location, but also on state parameters such as velocity and orientation. A by-product of such constraints is that the underlying graph becomes directed, as there may be edges that can only be passable in one direction due to kinematic constraints of the agent. MAPF-POST, as an example, is a MAPF algorithm that considers these kinematic constraints by post-processing a solution created by a MAPF algorithm. There is also a reduction-based approach that assumes rotation actions as a half way to kinematic constraints (**?**).

3.4 Tasks and Agents

In classical MAPF, each agent has one task - to get it to its target. Several extensions have been made in the MAPF literature in which agents may be assigned more than one target.

Anonymous MAPF. In this MAPF variant, the objective is to move the agents to a set of target vertices, but it does not matter which agent reaches which target (**?**; **?**). Another way to view this MAPF variant is as a MAPF problem in which every agent can be assigned to any target, but it has to be a one-to-one mapping between agents and targets.

Colored MAPF. This MAPF variant is a generalization of anonymous MAPF in which agents are grouped into teams, and every team has a set of targets. The objective is to move the agents in each team to their targets (**?**; **?**). Another way to view this MAPF variant is as a MAPF problem in which every agent can be assigned to targets only from the set of targets designated for its team.

One can generalize colored MAPF even further, assigning a target and an agent to multiple teams.

Online MAPF. In *online MAPF*, a sequence of MAPF problems are solved on the same graph. This setting has also been called "Lifelong MAPF" (**?**; **?**). Online MAPF problems can be classified as follows.

- **Warehouse model.** This is the setting where a fixed set of agents solve a MAPF problem, but after an agent finds a target, it may be tasked to go to a different target (**?**). This setting is inspired by MAPF for autonomous warehouses.
- **Intersection model.** This is the setting where new agents may appear, and each agent has one task – to reach its target (**?**). This setting is inspired by autonomous vehicles entering and exiting intersections (**?**).

Of course, hybrid models in which an agent can receive a new task when it reaches its target and new agents can appear over time is also possible.

4 Benchmarks

In this section, we describe how classical MAPF algorithms have been evaluated, suggest an organized benchmark for this purpose, and point to other relevant benchmark suites.

4.1 Characteristics of a MAPF Benchmark

A MAPF problem is defined by a graph and a set of source and target vertices. As such, a benchmark for MAPF includes a set of graphs, and for every graph a set of sets of source and target vertices.

Graphs for evaluating MAPF algorithms. The types of maps commonly used in prior work include:

- **Dragon Age Origins (DAO) maps.** These are grids taken from the game Dragon Age Origin and are publicly available in Sturtevant's `movingai.com` repository (**?**). These grids are relatively large and open, where some grids are as large as a 1000×1000 and more.
- **Open $N \times N$ grids.** These are $N \times N$ grids, where common values of N are 8, 16, and 32. Such grids allow experiments in which the ratio of agents to space or *agent density* is high, having fewer vertices without an agent in them.

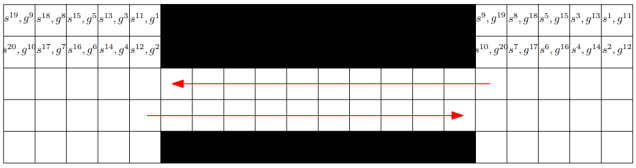


Figure 3: An example of a designated method for setting sources and targets, taken from (**?**). They chose randomly a source from the open space on the left and a target from the open space on the right for 25% of the agents, and chose sources and targets from open spaces on the right and left, respectively, for the rest of the agents.

- $N \times N$ **grids with random obstacles.** These are $N \times N$ grids, where a set of grid cells are randomly selected and are considered to be impassable (obstacles) (**?**).
- **Warehouse grids.** Inspired by real-world autonomous warehouse applications, recent MAPF papers also experimented with grids shaped to be similar to an automated warehouse, with long corridors (**?**; **?**). Figure **??** shows an illustration of a warehouse grid taken from (**?**).

4.2 Sources and targets assignments

After choosing a type of map, one needs to set the agents' source and target vertices. Several methods for setting agents' sources and targets have been used in the literature, including:

- **Random.** Setting the source and target vertices by randomly choosing vertices and making sure there is a path in the graph between them.
- **Clustered.** Setting the first agent's source and target by randomly choosing vertices in the graph. Setting the sources and targets of all others agents to be with distance of at most r from the first agent's source and target, respectively, where r is a parameter.
- **Designated.** Setting the source of each agent by randomly choosing from a designated set of possible source vertices, and setting the target of each agent similarly by choosing from a set designated set of possible target vertices.

Random assignment is probably the most common in the literature. The *clustered* assignment method has been used to make MAPF problems more challenging. The *designated* assignment method has been used in prior work in an effort to simulate automated warehouses (**?**; **?**; **?**) and autonomous vehicles in intersections (**?**). In an automated warehouse, there are often humans situated in specific locations to package the delivered bin and most tasks deliver a package to, or retrieve a package from, a human in these locations. In a setting with autonomous vehicles driving in and out of an intersection, the designated sources and targets are the intersection end points (**?**).

4.3 Publicly Available MAPF Benchmarks

We describe here two publicly available benchmarks for MAPF research, the first of which is a new benchmark set described for the first time in this paper.

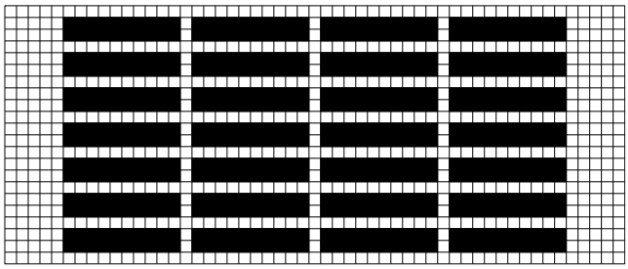
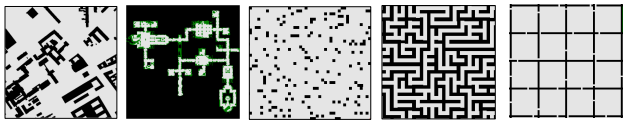


Figure 4: Illustration of a warehouse grid (**?**).



(a) City (b) DAO (c) Open + obstacles (d) Maze (e) Room

Figure 5: An example of a map from each type of maps available in the grid MAPF benchmark.

Grid-based MAPF. This publicly available[4] benchmark consists of 24 maps taken from (1) maps of real cities, (2) the video games Dragon Age Origins and Dragon Age 2, (3) open grids with and without random obstacles, (4) maze-like grids, and (5) room-like grids. All maps were taken from the MovingAI pathfinding repository (**?**).[5] Figure 5 shows an example of a map from each of these types, and Table 1 shows the dimensions of these maps.

Every map has 25 *scenarios*. Every scenario has a list of source and target vertices that were set using a variant of the random method (see Section 4.2) All points in the largest reachable region of each map were randomly paired, and then the first 1000 problems were put into the scenario. Thus, one can create a set of MAPF problems from each scenario by choosing any subset of source and target vertices.

We propose using this benchmark in the following way. For a chosen MAPF algorithm, map type, and scenario, try to solve as many agents as possible in each scenario, adding them in consecutive order. That is, start by creating a MAPF problem of two agents, using the first two source-target pairs associated with the chosen scenario, and run the MAPF algorithm of choice to solve this problem. If the algorithm of choice successfully solves this MAPF problem in reasonable time, create a new MAPF problem with 3 agents by using the first three source-target pairs of that scenario and try to solve it with the MAPF algorithm of choice. This continues iteratively until the algorithm of choice cannot solve the created MAPF problem in reasonable time. An evaluated algorithm can then report, for every scenario, the maximal number of agents it was able to solve in reasonable time.

To provide a baseline for comparison, we performed this evaluation process using ICBS (**?**). Using the terminology

[4] https://movingai.com/benchmarks/mapf.html
[5] https://movingai.com/benchmarks/grids.html

Type	Map	Size	Problems	Solved	Min	Max
City	Berlin_1_256	256 X 256	25000	892	4	52
	Boston_0_256	256 X 256	25000	718	13	47
	Paris_1_256	256 X 256	25000	805	3	47
DAO	brc202d	481 X 530	25000	252	2	22
	den312d	81 X 65	25000	577	7	36
	den520d	257 X 256	25000	661	5	47
	lak303d	194 X 194	25000	377	8	27
	orz900d	656 X 1491	25000	162	2	12
	ost003d	194 X 194	25000	535	7	37
Dragon Age 2	ht_chantry	141 X 162	25000	513	11	35
	ht_mansion_n	270 X 133	25000	795	17	42
	w_woundedcoast	578 X 642	25000	336	6	22
	lt_gallowstemplar_n	180 X 251	25000	493	10	32
Open	empty-8-8	8 X 8	800	528	18	25
	empty-16-16	16 X 16	3200	840	15	52
	empty-32-32	32 X 32	12800	1190	12	81
	empty-48-48	48 X 48	25000	1349	18	118
Open+ obstacles	random-32-32-10	32 X 32	11525	1027	15	68
	random-32-32-20	32 X 32	10225	862	15	46
	random-64-64-10	64 X 64	25000	1450	24	87
	random-64-64-20	64 X 64	25000	1078	10	64
Maze	maze-32-32-2	32 X 32	8325	327	8	17
	maze-32-32-4	32 X 32	9875	317	4	23
	maze-128-128-10	128 X 128	25000	272	5	20
	maze-128-128-2	128 X 128	25000	178	4	15
Room	room-32-32-4	32 X 32	12800	469	10	26
	room-64-64-16	64 X 64	25000	629	12	45
	room-64-64-8	64 X 64	25000	360	8	24

Table 1: Results for running ICBS with a timeout of 30 seconds on the grid MAPF benchmark.

introduced in this paper, our setting was a classical MAPF setting on a 4-neighbor grid where (1) edge, vertex, and swapping conflicts were forbidden, (2) following and cycle conflicts were allowed, (3) the objective was the sum of costs, and (4) the agent behavior at target is stay at target.

Table 1 shows the result of this evaluation. We set 30 seconds as the runtime limit. The different rows correspond to different maps. The "Size" column reports the number of rows and columns in each map. The "Problems" column reports the number of problems available for each map. Note that this number is aggregated over the 25 scenarios, where the number of problems available in a scenario is the number of source-target pairs defined for it. The "Solved" column reports the number of problems solved by ICBS under the specified time limit (30 seconds). As can be seen, while ICBS is able to solve many problems, the problems in this benchmark are complex enough so that there are many problems that cannot be solved by ICBS in the allotted time. Thus, the problems in this grid MAPF benchmark are hard enough to pose a challenge for contemporary MAPF solvers.

Table 1 has two additional columns - "Min" and "Max". These columns report the maximal number of agents solved by ICBS in the scenario in which this number was smallest ("Min") and when it was largest ("Max"). For example, the "Min" value for map `brc202d` is 2 and the "Max" is 22. This means there exists a scenario for this map in which ICBS were able to solve at most 2 agents before reaching a timeout, and there is a different scenario for this map in which ICBS were able to solve up to 22 agents. We report these values to show the diversity in difficulty between the scenarios of the same map.

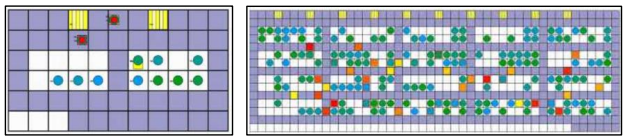


Figure 6: Two scenarios from the Asprilo framework. The left figure is a full warehouse scenario, which includes moving bins from one place to another. The right figure is a movement-only scenario, which corresponds to a classical MAPF problem.

Asprilo. An additional tool that is useful for MAPF research is Asprilo. Asprilo is a publicly available framework for simulating an automated warehouse (**?**). It includes tools for defining and generating standard automated warehouse planning problems, and tools for verifying and visualizing plans that solve these problems.

The type of planning problems supported by Asprilo includes problems in which robots are tasked to pick up and deliver bins in the warehouse from one place to another. These scenarios are grouped into different *domains*, representing different types of problems. Of interest to the MAPF community in particular is domain M, which basically represents MAPF problems. Thus, one can use problems from this domain as a benchmark for MAPF algorithms. Figure 6 shows two scenarios from Asprilo. The scenario depicted on the left side is a full warehouse scenario, where the agents are tasked to move bins from one place to another. The scenario on the right side is a movement-only scenario, i.e., a classical MAPF problem. Details on ASPRILO can be found in (**?**), as well as the project's website.[6]

5 Conclusion

In the first part of this paper, we defined common assumptions in the "classical" Multi-Agent Pathfinding (MAPF) problem and discuss the relationships between them. Then, we defined notable extensions to classical MAPF that were previously published. In the second part of this paper, we introduced a new suite of MAPF benchmark problems and point to another set of MAPF benchmark problems. Both parts of this paper are intended to propose a common language, terminology, and experimental setting for MAPF research. It is our hope that future MAPF researchers will follow our terminology and find these benchmarks useful.

6 Acknowledgments

This research is supported by ISF grants no. 210/17 to Roni Stern and #844/17 to Ariel Felner and Eyal Shimony, by BSF grant #2017692 and by NSF grant #1815660.

Ab fuga veniam numquam quae explicabo temporibus esse repudiandae tempora doloremque, esse rem ab rerum voluptatibus sit odit perferendis, magnam pariatur saepe itaque officia minima?Tempore fugit pariatur at veniam, cum enim vero a consequuntur labore velit, aperiam non sed adipisci ad debitis ratione cumque beatae?Quasi ut adipisci

[6] https://asprilo.github.io

saepe eaque doloremque est voluptate quaerat, quos facilis impedit omnis.