Multi-Agent Corridor Generating Algorithm

Arseniy Pertzovsky , **Roni Stern** , **Roie Zivan** and **Ariel Felner**
Ben-Gurion University of the Negev
arsenip@post.bgu.ac.il, roni.stern@gmail.com, {zivanr, felner}@bgu.ac.il

Abstract

In this paper, we propose the Multi-Agent Corridor Generating Algorithm (MACGA) for solving the Multi-agent Pathfinding (MAPF) problem, where a group of agents need to find non-colliding paths to their target locations. Existing approaches struggle to solve dense MAPF instances. In MACGA, the agents build *corridors*, which are sequences of connected vertices, from current locations towards agents' goals, and evacuate other agents out of the corridors to avoid collisions and deadlocks. We also present the MACGA+PIBT algorithm, which integrates the well-known rule-based PIBT algorithm into MACGA to improve runtime and solution quality. The proposed algorithms run in polynomial time and have a reachability property, i.e., every agent is guaranteed to reach its goal location at some point. We demonstrate experimentally that MACGA and MACGA+PIBT outperform baseline algorithms in terms of success rate, runtime, and makespan across diverse MAPF benchmark grids.

1 Introduction

Multi-agent Pathfinding (MAPF) is the problem of finding a set of non-colliding paths for a group of agents to their goal locations [Stern *et al.*, 2019]. Instances of MAPF exist in robotics [Barták *et al.*, 2019], automated warehouses [Wurman *et al.*, 2008; Salzman and Stern, 2020], digital entertainment [Ma *et al.*, 2017] and many more [Morris *et al.*, 2016]. Many MAPF algorithms have been proposed in the past decade [Felner *et al.*, 2017; Li *et al.*, 2021; Okumura, 2023b]. Some MAPF algorithms are *complete* in the sense that they are guaranteed to return a solution if one exists, and some are also *optimal*, i.e., they are guaranteed to have the lowest cost according to some cost function.

Optimal and complete MAPF solvers struggle to solve to large-scale MAPF instances. By contrast, MAPF algorithms such as Prioritized Planning (PrP) [Silver, 2005], PIBT [Okumura *et al.*, 2022], MAPF-LNS2 [Li *et al.*, 2022], and LaCAM* [Okumura, 2023a] often scale to very large problems with many agents by trading optimality and sometimes even completeness for faster running time. Yet these algorithms often fail to solve small and dense MAPF problems.

In this work, we propose a novel suboptimal and incomplete MAPF algorithm called the Multi-Agent Corridor Generating Algorithm (MACGA), which outperforms existing algorithms on several benchmark MAPF domains, including dense small maps. MACGA builds on the Corridor Generating Algorithm (CGA) algorithm, which is a recently introduced algorithm for the Single-Agent Corridor Generating (SACG) [Pertzovsky *et al.*, 2024] problem. In SACG, an agent denoted as the *main agent*, needs to arrive at its goal and it is allowed to push away other agents out of its way. MAPF can be viewed as a generalization of SACG, in which each agent has a goal, and they all need to arrive at their destination. MACGA aims to capture this intuition by iterating over the agents one at a time.

MACGA has several attractive properties. Unlike many MAPF algorithms, which require heavy computation (e.g., LaCAM [Okumura, 2023b], LaCAM* [Okumura, 2023a] or CBS [Sharon *et al.*, 2015]), MACGA is tractable, running in polynomial time. Similar to PIBT [Okumura *et al.*, 2022], it guarantees *reachability*, which means that every agent will eventually reach its goal (but not simultaneously with others). In terms of performance, MACGA outperforms PIBT in dense problems in some cases, while in others, it does not. To enjoy the complementary benefits of both algorithms, we propose MACGA+PIBT, a MAPF algorithm that integrates PIBT and MACGA into a single, efficient procedure.

Lastly, we conducted a large set of experiments on standard MAPF benchmarks [Stern *et al.*, 2019] comparing MACGA and MACGA+PIBT to other standard and state-of-the-art suboptimal algorithms, namely PrP, PIBT, LNS2, LaCAM, and LaCAM*. The results show that MACGA and MACGA+PIBT can generate outstanding results in terms of success rate in many MAPF benchmarks. For example, in a challenging *maze-32-32-2* grid, the proposed algorithms solved the majority of instances with 450 agents, while the baseline algorithms were not able to solve any of them.

2 Background

A *MAPF* problem is defined by a tuple $\langle G, n, s, t\rangle$ where $G \doteq (V, E)$ represents an undirected graph, n is the number of agents, $s : [1, ..., n] \to V$ maps an agent to a start vertex and $t : [1, ..., n] \to V$ maps an agent to a target/goal vertex. Time is discretized, and in every time step, each agent occupies a single vertex and performs one action. There are two types

of actions: $wait$ and $move$. A $wait$ action means that the agent will stay at the same vertex v at the next time step. A $move$ action means that the agent will move to an adjacent vertex v' in the graph (i.e. $(v, v') \in E$). A *single-agent plan* for agent i, denoted π_i, is a sequence of actions π_i that is applicable starting from $s(i)$ and ending in $t(i)$. A solution to a MAPF is a set of single-agent plans $\pi = \{\pi_1, \ldots, \pi_n\}$, one for each agent, that do not have any *conflicts*. We consider two types of conflicts: *vertex conflict* and *swapping conflict*. Two single-agent plans have a vertex conflict if they occupy the same vertex at the same time, and a swapping conflict if they traverse the same edge at the same time from opposing directions. The objective is to find a solution that minimizes the *Sum-of-Costs* (SoC), that is the sum over the lengths of π's constituent single-agent paths, or the *makespan*, that is the maximum length of all π's single-agent paths.

MAPF algorithms Many suboptimal MAPF algorithms have been proposed, including PrP, Large Neighborhood Search (LNS2), LaCAM, and others. The algorithm that is mostly related to our work is PIBT [Okumura *et al.*, 2022]. PIBT searches for valid paths in the *configuration* space, where a configuration is a vector representing the agents' locations in some time-step. PIBT searches this space in a greedy and myopic manner. It starts from the initial configuration of the agents and in every iteration generates a configuration for the next time-step until reaching a configuration in which all agents are at their goals. PIBT generates configurations recursively, moving every agent toward its goal while avoiding conflicts with previously planned agents. To avoid deadlocks, PIBT utilizes Priority Inheritance and Backtracking techniques. PIBT is very efficient computationally but is incomplete since it searches greedily in the configuration space.

The Corridor Generating Algorithm (CGA) CGA is an algorithm for solving the Single-Agent Corridor Generating (SACG) problem [Pertzovsky *et al.*, 2024]. SACG is defined on an environment with multiple agents. The objective is to move a single main agent to its target as fast as possible while avoiding conflicts with all other agents. The other agents have no goals of their own, but they may be required to move to evacuate the path of the main agent.

CGA is a complete algorithm for solving SAGC [Pertzovsky *et al.*, 2024]. It works by identifying *separating vertices* and *corridors*, which are defined as follows.

Definition 1 (Separating Vertex)**.** *A vertex v in a graph G is called a separating vertex (SV) if removing v from G results in a graph G' that includes more connected components than the number of connected components in G.*

Definition 2 (Corridor)**.** *A corridor in G is a path $(v_1, \ldots, v_n)$ in G such that all the vertices $v_2, \ldots, v_{n-1}$ are SVs. That is, a corridor is a path in which all vertices except the first and last must be SVs.*

A corridor is called *trivial* if it consists of a pair of neighboring vertices (v_1, v_2) in G such that both of the vertices are not SVs. This is the shortest possible corridor, as there are no intermediate SVs between v_1 and v_2 A vertex that is not an SV is denoted as *non-SV* and the set of all SVs for a given graph is denoted as *SVS*. Fig. 1 shows examples of SVSs for several grids and highlights in blue some of the corridors in these graphs. Any subsequent two green cells are an example of a *trivial corridor*.

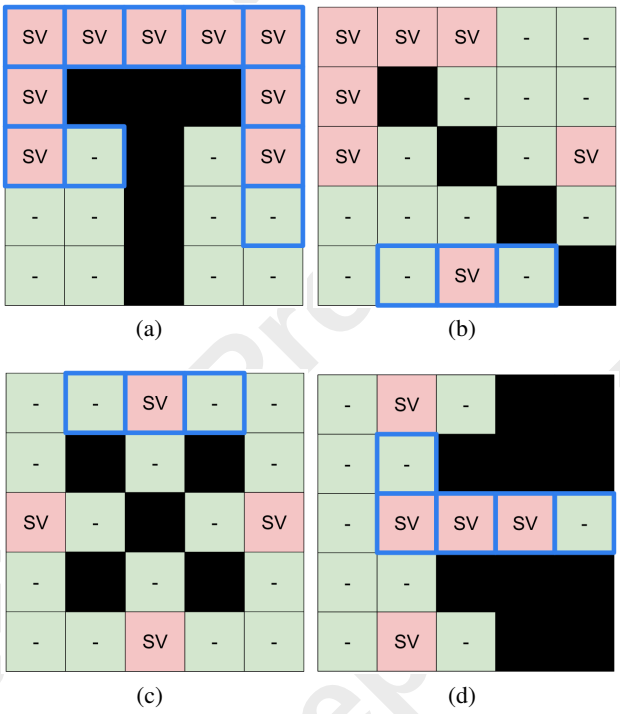


Figure 1: SVSs for different graphs. Black cells are obstacles; red cells marked by *"SV"* are the SVs; green cells are non-SVs.

CGA first finds an optimal path π for the main agent to the goal assuming no other agents exist. Path π can be decomposed to a sequence of corridors. CGA iteratively moves the main agent along these corridors one at a time. While moving through a corridor, CGA ensures that the other agents are evacuated from that corridor to allow the main agent to pass through it. The evacuation of other agents can be performed before or in parallel to the moves of the main agent as long as the main agent is guaranteed to be able to pass through the corridor. See Pertzovsky et al. [2024] for more details. CGA was proved to be a complete algorithm for solving SACG problems, i.e., it guarantees that the main agent eventually reaches its target, given that the following two conditions hold. The first condition is that the number of non-SVs is larger than the length of the longest corridor in a grid. This condition holds true in most grids that are commonly used in MAPF benchmarks. The second condition is that the initial location of the main agent is a non-SV.

3 Multi-Agent CGA (MACGA)

The *Multi-Agent CGA* algorithm solves the classical MAPF problem where all agents need to eventually be at their goals simultaneously. It iterates over all agents and applies CGA for each of them. MACGA works in steps. At each step, it associates agents with sequences of one or more actions to

perform. We refer to such a sequence as the *active plan* of an agent. Then, each agent performs the first step of its active plan, and the next step begins. [1]

Initially, all agents do not have an active plan. Then, MACGA alternates between a *planning phase* and an *execution phase*. In the planning phase, MACGA iterates over all agents. For every agent that does not yet have an active plan and attempts to create one for it. Creating an active plan for agent a is done as follows. First, a *corridor* is identified from the current location of agent a' along an optimal path to its goal. If that corridor is not occupied, a is assigned an active plan to go through it. Otherwise, if the corridor is occupied by agents with no active plan, a dedicated `Evacuate` procedure is invoked to try to set active plans for these agents who move them out of the corridor. If the `Evacuate` procedure succeeds, MACGA assigns a with an active plan to go through the chosen corridor. Otherwise, agent a and the agents that block its corridor are not assigned an active plan. At the end of the planning phase, agents without an active plan are assigned a default active plan of staying in their place for one time step. In the execution phase, all agents perform a single step from their active plans and update their active plans accordingly. MACGA halts when all the agents reach their goals. Next, we describe in more detail the key components of the planning phase, namely (1) how to choose a corridor for a given agent a, (2) how to try to evacuate the other agents from the corridor, and (3) how to assign a with an active plan that passes through the corridor afterward. A detailed pseudo-code of MACGA is presented further in this paper.

Choosing a Corridor This procedure receives agent a's current location and its goal, denoted $a.curr$ and $a.goal$, respectively. The procedure, denoted as `CreateCorridor`, outputs a corridor $a.corridor$ that starts from $a.curr$ and ends in either $a.goal$ or a non-SV (as done in CGA). Specifically, it chooses an optimal path from $a.curr$ to $a.goal$ (ignoring all other agents). Then, it iterates over the vertices in this path starting from $a.curr$ and appending them to $a.corridor$ until reaching either $a.goal$ or a non-SV. To identify if a vertex is an SV or non-SV, we compute the SVS for every agent as described by [Pertzovsky *et al.*, 2024] in a pre-processing stage while considering the agent's goal vertex as an obstacle. This can be done in polynomial time. Fig. 2.a shows an example where the agent needs to go from the orange circle to the orange square. Fig. 2.b shows a corridor for agent 1.

Evacuating the Corridor This procedure, referred to as `Evacuate`, receives an agent a; its corridor, $a.corridor$; the locations of the other agents; and their active plans. `Evacuate` attempts to find an active plan for the agents currently occupying $a.corridor$ such that they eventually exit the corridor, allowing a to pass through it. If `Evacuate` returns $False$, it means that it has not been able to free $a.corridor$ from other agents, and a should not be assigned an active plan that passes through it at this point.

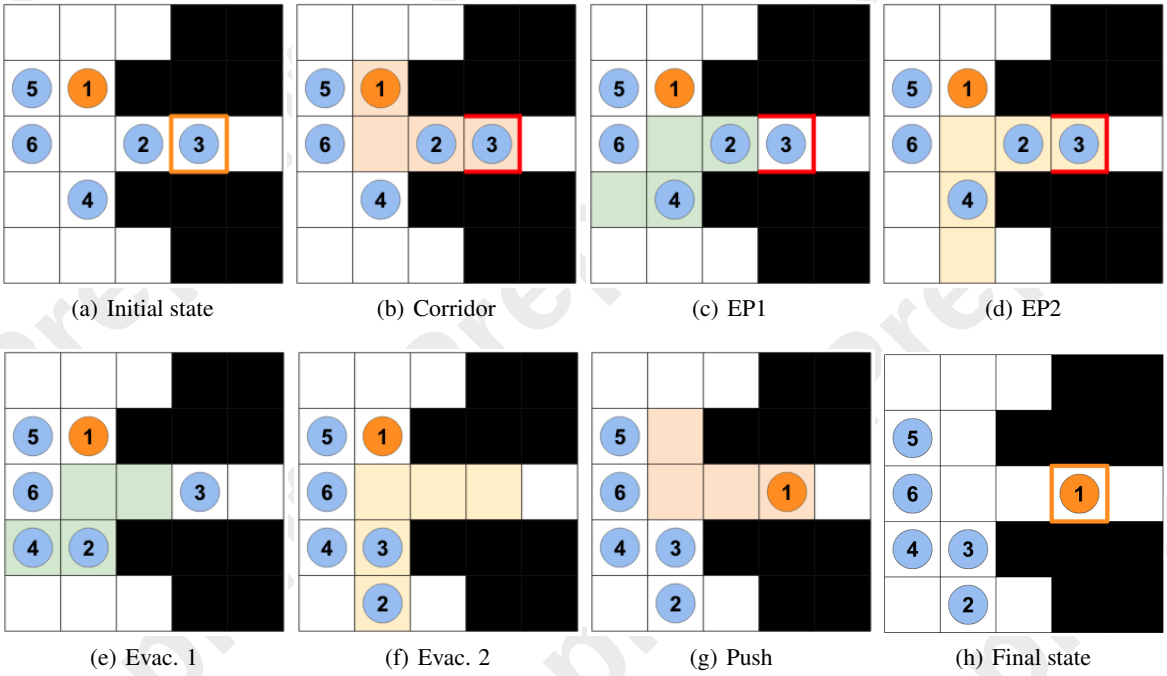
[1] MACGA can be activated online where agents physically perform their moves. Alternatively, it can be activated offline, and these moves are added to a future intended plan that is returned at the end of the execution.

Let A_{in} denote the set of agents whose current location resides in $a.corridor$. If A_{in} includes an agent that already has an active plan, `Evacuate` immediately returns $False$. Otherwise, it iterates over the agents in A_{in}, and for each agent $a' \in A_{in}$, it searches for a path from its current location to an unoccupied vertex outside of $a.corridor$. To avoid deadlocks and livelocks, this search is not allowed to pass through any vertex planned to be occupied by an active plan of another agent or to pass through edges connected to a's target vertex that lead outside of $a.corridor$. We refer to such a path as an *evacuation path* (EP). Importantly, `Evacuate` ensures that each EP ends at a different vertex. Finding EPs that satisfy these requirements is done by running a Breadth-First Search for each agent from its current location, ignoring conflicts with other agents in A_{in}, and avoiding the prohibited vertices and edges mentioned above. If the search for an EP fails for some agent $a' \in A_{in}$, `Evacuate` returns $False$. Otherwise, we obtain a set of EPs, one for every agent in A_{in}.

Note that an EP is a path in G, while an active plan is a sequence of actions that may include wait and move actions. Based on the EPs found, `Evacuate` creates an active plan for every agent a' in A_{in}, denoted $\pi(a')$, as follows. Initially, these active plans are empty. Then, `Evacuate` iterates over the EPs in some arbitrary order, performing the following steps for every EP e.

For each agent $a' \neq a$, we use $loc(a')$ to denote the last location that a' is expected to reach after following its active plan. We iterate through the vertices of e from the closest to the current location of a to the furthest away from it, and for every vertex e_i, we check if there is an agent a' for which $loc(a')$ equals e_i. If such an agent exists, we add it to an ordered list $e(A)$, which is sorted according to the distance of $loc(a')$ from the current location of the agent a. Once the procedure is completed and all relevant agents have been added to $e(A)$ (assume that there are m agents in $e(A)$ and denote the last among them by a_m) we append to $\pi(a_m)$ a sequence of actions moving a_m from $loc(a_m)$ to the last vertex in e. Following this procedure ensures that the vertex $loc(a_m)$ is now unoccupied by any agent. Next, we iterate over the other agents in $e(A)$ in reverse order, appending to the active plan of agent a_i a sequence of actions that move it from $loc(a_i)$ to the previous location of a_{i+1}. Whenever we append an action to an active plan of an agent a_i, we check if an active plan of another agent intends to occupy the same vertex in the future. This may occur when EPs overlap. In such a case, we append wait actions to the active plan of a_i until it can safely move to that vertex. Note that the above process is repeated for every EP, potentially appending actions to agents in A_{in} multiple times. Fig. 2(c-d) shows examples of finding EPs and Fig. 2(e-f) shows the execution of the corresponding active plans.

Passing through the Corridor In the case where `Evacuate` was able to successfully create active plans, the next step involves assigning the agent a with an active plan that moves it through the chosen corridor. Similar to how actions are added to agents in `Evacuate`, we only add an action to a's active plan to move to a vertex v if v is not occupied in the future by any active plan. Otherwise, we append a wait action to a's active plan until v is free. We denote this



(a) Initial state (b) Corridor (c) EP1 (d) EP2

(e) Evac. 1 (f) Evac. 2 (g) Push (h) Final state

Figure 2: An example of MACGA execution.

procedure as `Push`.

Temporary Targets As described above, `Evacuate` may fail (return $False$) to evacuate a corridor $a.corridor$ if it cannot find an EP for some agent a' who resides in the corridor. If this failure is not due to an active plan of some other agent, there is a risk of reaching a deadlock situation. To mitigate this to some extent, if the search for an EP fails but the corresponding search did not attempt to generate any vertex used by an active plan, then `Evacuate` also assigns a new, temporary target for a to avoid deadlocks. In our implementation, the temporary target was the unoccupied non-SV, which is closest to a. When an agent reaches a temporary target, it is reassigned its original target.

An example of such an unsolvable instance and how a temporary goal helps to resolve it is depicted in Fig. 3. Here, agent 1 has a goal vertex (orange square), but it is impossible to evacuate agents 2 and 3 out of the corridor, so the agent moves to the temporary goal vertex (Fig. 3 (a)-(d)). At this stage, we reach the same state as depicted in Fig. 2, which MACGA can solve.

Pseudocode The high-level pseudocode of MACGA (excluding the blue text) and MACGA+PIBT (including the blue text) is illustrated in Algorithm 1. The algorithm starts by creating SVS (line 2). The algorithm halts only when all the agents are at their goal locations (line 3). In every time step, it loops through all agents (lines 5-22). Algorithm 1 continues to the next agent if agent a (line 5) already has an active plan (lines 6-8). If a is at its goal but the goal is temporary, a sets back its goal to be the initial one (lines 9-11). In MACGA+PIBT, if the next vertex is non-SV, the algorithm executes PIBT on agent a and continues to the next agent (lines 12-15). Then, MACGA creates a corridor for a by the `CreateCorridor` procedure (line 16) and attempts to evacuate the corridor with `Evacuate` procedure. If `Evacuate` fails, the algorithm continues to the next agent (lines 18-20). After that, the `Push` procedure is executed (line 21). Lastly, all agents execute their plan for time-step i (the agents without an active plan remain in their current locations)(line 23). The `UpdateOrder` function sends all finished agents to the end of the order (line 24). Due to the reachability property, a first agent in order is guaranteed to eventually reach its goal. The `UpdateOrder` ensures that every agent at some point will be first in the order. If succeeds, the algorithm returns a set of paths for every agent (line 26).

4 Theoretical Properties

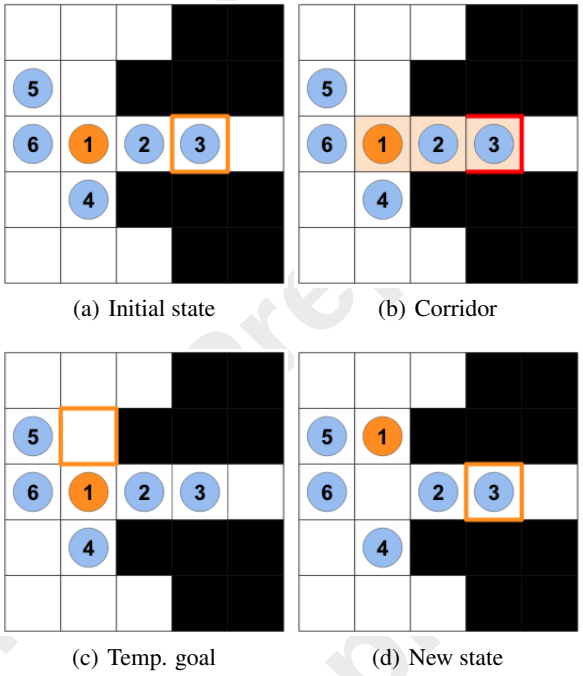
First, we analyze the runtime of the MACGA procedures. The runtime complexity of the `CreateCorridor` procedure is $O(|V|+|E|)$ as it simply runs a Breadth-First Search. Similarly, running `Evacuate` for a single agent requires $O(|V| + |E|)$. `Evacuate` searches for EVs at most $|A|$ times, and thus its runtime is $O(|A|(|V|+|E|))$. The runtime of `Push` is at most $O(|A||V|)$, pushing the agents across already calculated corridors. Unfortunately, there are no guarantees on the global runtime of MACGA, as it cannot identify the unsolvable instances.

While MACGA is incomplete, it does satisfy the *reachability* property as PIBT. That is, each agent is guaranteed to eventually reach its goal, albeit possibly not at the same time.

Next, we gradually prove the reachability of MACGA.

Algorithm 1 MACGA +PIBT

Input: $\langle A, G := (V, E) \rangle$
$SVS \leftarrow CreateSVS(G)$
while not all agents at their goals **do**
  $i \leftarrow$ current time step
  for every $a \in A$ **do**
    if $a.path[i] \neq \emptyset$ **then**
      Continue
    end if
    if $a.curr = a.goal \wedge a.tempGoal$ **then**
      $a.tempGoal \leftarrow False$; $a.goal \leftarrow a.initGoal$
    end if
    if next node is *non-SV* **then**
      `PIBT(`a`, `A`)`
      Continue
    end if
    `CreateCorridor(`a`)`
    `found_bool` $\leftarrow$ `Evacuate(`a`, `A`)`
    if $\neg$ `found_bool` **then**
      Continue
    end if
    `Push(`a`, `A`)`
  end for
  `Execute(`i`, `A`)`
  `UpdateOrder(`A`)`
end while
Return π



(a) Initial state (b) Corridor

(c) Temp. goal (d) New state

Figure 3: An example of assigning a temporary goal.

Lemma 1. *In MACGA, if the first agent in order a_1 is occupying a non-SV and the number of unoccupied vertices in a graph G is greater than or equal to the length of the longest corridor in G, then the* FindEVs *procedure will successfully find EVs for all agents from any corridor for agent a_1.*

Proof outline. Since the main agent is not occupying a SV, there exists a path from every vertex in the next corridor to any vertex in G that does not go through the main agent's location. As there are more unoccupied vertices than vertices in the corridor, there exists an unoccupied vertex in G for every vertex in this corridor. Thus, *FindEVs* will find evacuation routes for every vertex in the corridor, as required. □

Theorem 1 (Completeness for a_1)**.** *If the first agent in order a_1 is not occupying an SV and the number of unoccupied vertices in a graph is equal to or greater than the length of the longest corridor in G, then MACGA is guaranteed to bring a_1 agent to its goal.*

Proof outline. The *CretateCorridor* procedure in CGA ensures that the main agent moves from one non-SV vertex to another along an optimal path to the goal. Due to Lemma 1, *FindEVs* together with *EvacuateAndPush* will successfully evacuate the corridor connecting these two non-SV vertices. Consequently, after a finite number of steps, the a_1 agent will reach its goal. □

Theorem 2 (Reachability of MACGA)**.** *In MACGA, if the number of unoccupied vertices is larger than the longest corridor and the* UpdateOrder *function ensures that every agent will eventually be first in order, then every agent is guaranteed to reach its next goal location in a finite amount of time.*

Proof: Following Theorem 1, the agent with the highest priority will reach its goal location in a finite amount of steps, as it applies MACGA without any restrictions. *UpdateOrder* function assigns the lowest priority to agents that have reached their goals. Thus, eventually, every agent will be the highest priority agent and reach its goal [2]. □

Discussion MACGA can be viewed as a rule-based MAPF algorithm such as Push & Swap (PS) [Luna and Bekris, 2011] and Push & Rotate (PR) [De Wilde *et al.*, 2013], since its `Evacuate` procedure is somewhat reminiscent of their "swapping" and "rotating" procedures. A key advantage of MACGA over these rule-based MAPF solvers is that it pushes all agents to act concurrently as much as possible, mitigating to some extent the poor solution quality often exhibited by rule-based MAPF solvers. MACGA also bears similarity to prioritized planning algorithms, since in its planning phase the agents plan sequentially, blocking future agents from occupying some locations. Unlike PrP, in MACGA agents are not required to create full paths to their targets, reducing computational effort.

MACGA have some similarities to PIBT [Okumura *et al.*, 2022], as it plans for every agent only a few steps ahead. However, in PIBT the planning is only a single step ahead, which leads to deadlocks and livelocks in MAPF, especially in narrow closed corridors. MACGA delicately solves these cases with its `Evacuate` procedure. However, MACGA orders agents arbitrarily while PIBT uses heuristics to prior-

[2] Note that, in contrast to the completeness property, reachability does not guarantee that the agents will reach their goals simultaneously.

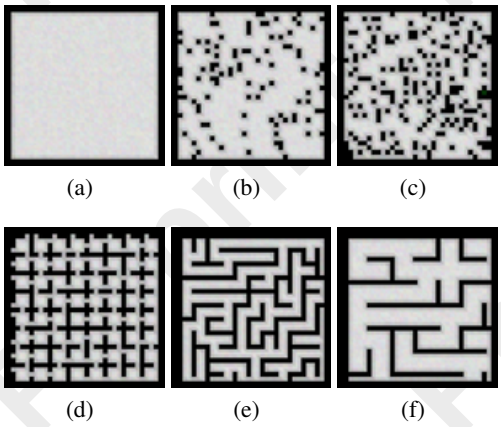


Figure 4: MAPF Grids: (a) *empty-32-32*, (b) *random-32-32-10*, (c) *random-32-32-20*, (d) *room-32-32-4*, (e) *maze-32-32-2*, (f) *maze-32-32-4*

itize agents' movements. This is very beneficial when we are not close to a long corridor. To enjoy the complementary benefits of PIBT and MACGA, we developed a hybrid algorithm called MACGA+PIBT. In MACGA+PIBT, when planning for an agent a, we first use PIBT to choose the next vertex to go to, considering the vertices of the other active plans as obstacles. If the vertex chosen by PIBT is a non-SV, we set the active plan to go to it, forming a trivial corridor of size 2. Otherwise, we plan for that agent as in the regular MACGA algorithm.

5 Experimental Results

We conducted an experimental evaluation comparing MACGA with PrP, LNS2, PIBT, LaCAM, and LaCAM*, where PrP and LNS2 are implemented with SIPPS [Li *et al.*, 2022]. The chosen baselines are considered to be state-of-the-art algorithms for solving MAPF suboptimality [Stern *et al.*, 2019; Okumura, 2023b]. We did not include other rule-based MAPF algorithms such as PS [Luna and Bekris, 2011] and PR [De Wilde *et al.*, 2013] in our baselines as they were shown to be significantly inferior to PIBT [Okumura *et al.*, 2022].

All experiments were performed on six different maps from the MAPF benchmark [Stern *et al.*, 2019]: *empty-32-32*, *random-32-32-10*, *random-32-32-20*, *room-32-32-4*, *maze-32-32-2*, and *maze-32-32-4* as they present different levels of difficulty. The maps are visualized in Figure 4. The number of agents used in our experiments varied from 100 to 700. We executed 20 random instances per every number of agents, map, and algorithm. A time limit of 30 seconds was imposed on every instance. The ratio of instances solved within this time limit by a given algorithm is referred to as the "success rate" of that algorithm. All algorithms were implemented in Python and ran on a MacBook Air with an Apple M1 chip and 8GB of RAM.[3]

[3]Prior works implemented MAPF algorithms on different programming languages, including C++, Java, C#, and Python. Thus, our results cannot be compared blindly with the published results from other papers. Nevertheless, we made significant efforts to verify that our implementations carefully matched published implementations of other algorithms if such existed.

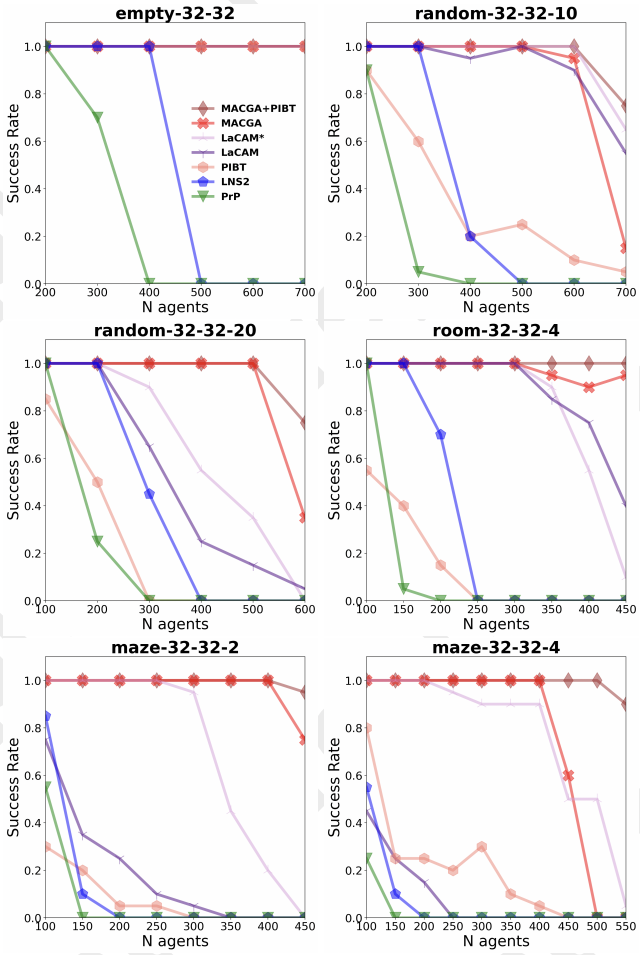


Figure 5: Success Rate

Fig. 5 presents the *success rate* (SR) of algorithms (y-axis) in different grids per number of agents (x-axis), where the SR is the ratio of problems that could be solved within the allocated time limit. In general, the results show that MACGA and MACGA+PIBT solve the majority of the problems, outperforming all the others evaluated algorithms. The best performance of our approaches relative to baseline algorithms was in the grids that contain narrow spaces, such as *room-32-32-4* or *maze-32-32-2*. As an example, in *maze-32-32-2* grids, our approaches solved most of the instances within 450 agents, while other baseline algorithms did not succeed in solving any of them. MACGA+PIBT outperforms MACGA in terms of success rate in almost all maps. This suggests the goal-oriented behavior of PIBT in trivial corridors indeed yields the intended effect of moving the agents faster toward their goal.

Fig. 6 plots the of runtime (y-axis) required to solve instances (x-axis) in increasing order of time over all instances that an

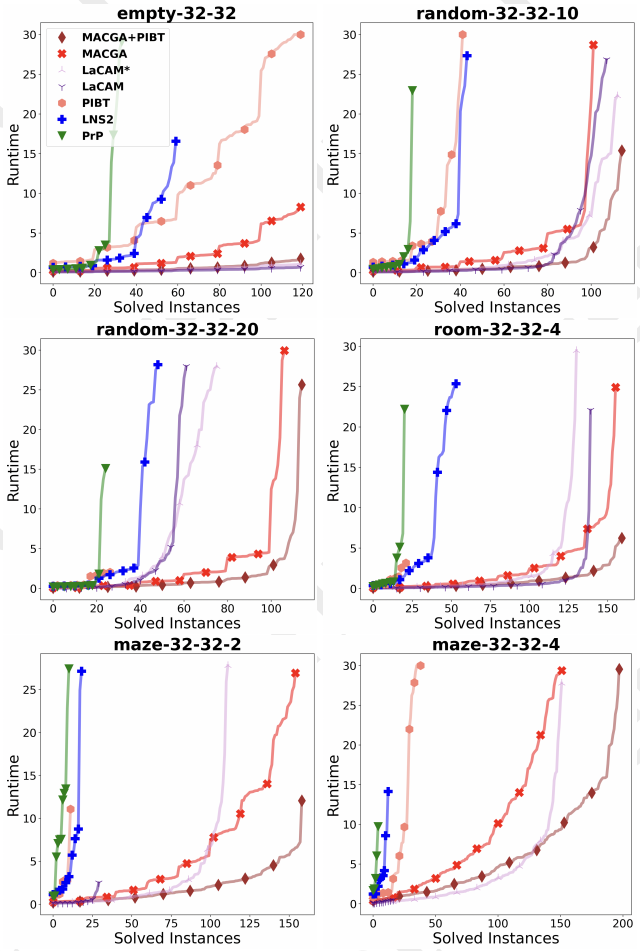


Figure 6: Runtime (Cactus Graph)

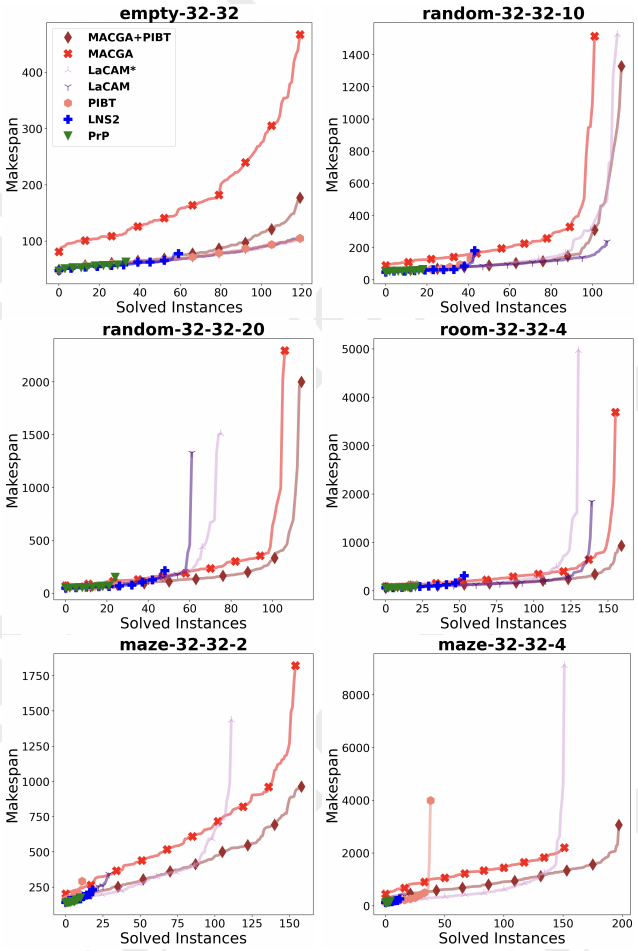


Figure 7: Makespan (Cactus Graph)

algorithm solved among all the instances given above, i.e., instances with different number of agents are all included here. This kind of graph is sometimes referred to as *cactus graph* in literature. In some cases, LaCAM variants were faster than MACGA, such as in *empty-32-32* grid. Nevertheless, in nearly all other instances, MACGA+PIBT outperformed other algorithms by a large margin. For example, in *maze-32-32-2*, MACGA+PIBT solved more than 100 instances more than LaCAM* (who was second best). This can be explained by the fact that with more dense grids and a rising number of agents, the search tree of LaCAM variants grows rapidly, whereas in our algorithms (MACGA and MACGA+PIBT), the execution remains rule-based, and the runtime grows much slower. In all the grids, MACGA was significantly slower than MACGA+PIBT.

Fig. 6 compares the *makespan* obtained by each algorithm by solving instances with a varying number of agents. Those graphs are also presented in the form of *cactus graphs*. The y-axis is a makespan value and the x-axis designates the solved instances in increasing order of makespan In most of the cases MACGA returns a solution with higher costs, that are substantially higher than the baseline algorithms. However, MACGA+PIBT preserves much lower costs that are close to the baseline algorithms in the majority of the cases.

6 Conclusion

We introduced two incomplete rule-based algorithms, MACGA and MACGA+PIBT, for solving the Multi-Agent Path Finding (MAPF) problem. Both algorithms run in polynomial time and guarantee reachability. Experimentally, we showed that the proposed approaches solve MAPF problems more efficiently than baseline algorithms in terms of success rate. In the future, we intend to tailor MACGA and MACGA+PIBT for lifelong MAPF problems. Another interesting direction for future work is to adapt the proposed approaches to multi-agent path planning in continuous spaces.

Acknowledgements

This research was partly supported by the Helmsley Charitable Trust through the Agricultural, Biological and Cognitive Robotics Initiative and by the Marcus Endowment Fund, both at Ben-Gurion University of the Negev; by the Binational Science Foundation (BSF) fund #2022189 awarded to Roie Zivan; by ISF grant #909/23 and BSF grant #2021643 awarded to Ariel Felner; and by the Israel Science Foundation (ISF) grant #1238/23 awarded to Roni Stern.

References

[Barták *et al.*, 2019] Roman Barták, Jiří Švancara, Věra Škopková, David Nohejl, and Ivan Krasičenko. Multi-agent path finding on real robots. *AI Communications*, 2019.

[De Wilde *et al.*, 2013] Boris De Wilde, Adriaan W Ter Mors, and Cees Witteveen. Push and rotate: cooperative multi-agent path planning. In *AAMAS*, pages 87–94, 2013.

[Felner *et al.*, 2017] Ariel Felner, Roni Stern, Solomon Shimony, Eli Boyarski, Meir Goldenberg, Guni Sharon, Nathan Sturtevant, Glenn Wagner, and Pavel Surynek. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *SoCS*, 2017.

[Li *et al.*, 2021] Jiaoyang Li, Zhe Chen, Daniel Harabor, P Stuckey, and Sven Koenig. Anytime multi-agent path finding via large neighborhood search. In *IJCAI*, 2021.

[Li *et al.*, 2022] Jiaoyang Li, Zhe Chen, Daniel Harabor, Peter J Stuckey, and Sven Koenig. Mapf-lns2: Fast repairing for multi-agent path finding via large neighborhood search. In *AAAI*, 2022.

[Luna and Bekris, 2011] Ryan J Luna and Kostas E Bekris. Push and swap: Fast cooperative path-finding with completeness guarantees. In *IJCAI*, 2011.

[Ma *et al.*, 2017] Hang Ma, Jingxing Yang, Liron Cohen, T. K. Satish Kumar, and Sven Koenig. Feasibility study: Moving non-homogeneous teams in congested video game environments. In *AIIDE*, 2017.

[Morris *et al.*, 2016] Robert Morris, Corina S Pasareanu, Kasper Søe Luckow, Waqar Malik, Hang Ma, TK Satish Kumar, and Sven Koenig. Planning, scheduling and monitoring for airport surface operations. In *AAAI Workshop: Planning for Hybrid Systems*, 2016.

[Okumura *et al.*, 2022] Keisuke Okumura, Manao Machida, Xavier Défago, and Yasumasa Tamura. Priority inheritance with backtracking for iterative multi-agent path finding. *Artificial Intelligence*, 310:103752, 2022.

[Okumura, 2023a] Keisuke Okumura. Improving lacam for scalable eventually optimal multi-agent pathfinding. *arXiv*, 2023.

[Okumura, 2023b] Keisuke Okumura. Lacam: Search-based algorithm for quick multi-agent pathfinding. In *AAAI*, volume 37, pages 11655–11662, 2023.

[Pertzovsky *et al.*, 2024] Arseni Pertzovsky, Roni Stern, and Roie Zivan. Cga: Corridor generating algorithm for multi-agent environments. In *IROS*. IEEE, 2024.

[Salzman and Stern, 2020] Oren Salzman and Ron Zvi Stern. Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems blue sky ideas track. In *AAMAS*, 2020.

[Sharon *et al.*, 2015] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 2015.

[Silver, 2005] David Silver. Cooperative pathfinding. In *AIIDE*, 2005.

[Stern *et al.*, 2019] Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Eli Boyarski, and Roman Bartak. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *SoCS*, pages 151–158, 2019.

[Wurman *et al.*, 2008] Peter R Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, 29(1):9–9, 2008.