Decentralized Unlabeled Multi-Agent Pathfinding Via Target and Priority Swapping

Stepan Dergachev^{a, b,*} and Konstantin Yakovlev^{a, b, c}

^aFederal Research Center "Computer Science and Control" of the Russian Academy of Sciences ^bHSE University

°AIRI

ORCID (Stepan Dergachev): https://orcid.org/0000-0001-8858-2831, ORCID (Konstantin Yakovlev): https://orcid.org/0000-0002-4377-321X

Abstract. In this paper we study a challenging variant of the multiagent pathfinding problem (MAPF), when a set of agents must reach a set of goal locations, but it does not matter which agent reaches a specific goal - Anonymous MAPF (AMAPF). Current optimal and suboptimal AMAPF solvers rely on the existence of a centralized controller which is in charge of both target assignment and pathfinding. We extend the state of the art and present the first AMAPF solver capable of solving the problem at hand in a fully decentralized fashion, when each agent makes decisions individually and relies only on the local communication with the others. The core of our method is a priority and target swapping procedure tailored to produce consistent goal assignments (i.e. making sure that no two agents are heading towards the same goal). Coupled with an established rulebased path planning, we end up with a TP-SWAP, an efficient and flexible approach to solve decentralized AMAPF. On the theoretical side, we prove that TP-SWAP is complete (i.e. TP-SWAP guarantees that each target will be reached by some agent). Empirically, we evaluate TP-SWAP across a wide range of setups and compare it to both centralized and decentralized baselines. Indeed, TP-SWAP outperforms the fully-decentralized competitor and can even outperform the semi-decentralized one (i.e. the one relying on the initial consistent goal assignment) in terms of flowtime (a widespread cost objective in MAPF).

1 Introduction

Multi-agent navigation is a vital and non-trivial problem which arises in various practical applications such as mobile robotics, transportation systems, video-games etc. Generally, the problem asks to find a set of non-colliding trajectories (paths) for a group of agents operating in a shared workspace. Numerous modifications, setups and approaches for this problem exist. One of the most well-studied setups is when each agent is asked to reach a specific goal location, i.e., the assignment of goals to agents is given as the problem input [19]. Another variant is the so-called, unlabeled or anonymous multi-agent pathfinding (AMAPF) [25]. In this setting it is assumed that the agents are interchangeable in a sense that for a single agent there is no strict requirement to achieve a particular goal. It is this problem that we focus on in this work.



Figure 1. An example of a decentralized AMAPF instance with a consistent (left side) and an inconsistent (right side) initial assignments. Solid circles depict agents. Red squares are the goals. The colored area around each agent is its communication zone (it is able to communicate with the others only if they reside inside this zone).

Numerous methods have been recently proposed to solve MAPF. Some of them are intended to find optimal solutions w.r.t. space-time discretization (CBS [17], M* [22], ICTS [16] to name a few), while the others trade-off optimality for lower runtimes in a controlled fashion, like ECBS [2], EECBS [10], ODrM* [6], or completely ignore the cost objective in favor of smaller runtime and scalability like Push and Rotate [4], PIBT [13] and others. The same applies to AMAPF, where both optimal solvers [25] and more scalable suboptimal solvers [12] do exist.

Still, most of the state-of-the-art (A)MAPF solvers intrinsically assume that there exist a centralized controller that fully observes the environment and is in charge of constructing plans that the agents need to execute. In practice, however, deploying such centralized systems may be costly and decentralized methods to tackle (A)MAPF are desirable. In this case each agent has to decide on its own, based on the limited observation/communication, how to choose the target and move towards it while avoiding the collisions.

Most of the recent decentralized MAPF solvers are learning-based: PRIMAL [15], G2RL [23], SCRIMP [24], FOLLOWER [18] to name a few. Surprisingly, the decentralized AMAPF is much less studied and to the best of our knowledge no established decentralized AMAPF solver exists. Our work aims to fill this gap.

We start with a prominent (suboptimal) centralized AMAPF solver, i.e. TSWAP [12], and show how can one build a semi-

^{*} Corresponding Author. Email: dergachev@isa.ru

decentralized and then fully decentralized AMAPF algorithm on top of it. The main bottleneck in doing so is resolving inconsistencies in individual goal assignments, i.e. dealing with the problem when several agents are heading towards the same goal – see Fig. 1. We introduce both a naive way to cope with it and a more involved one, that relies only on the local communication and is based on a specific target and priority swapping procedure. We prove that the most advanced version of our solver, which we call TP-SWAP, is complete, i.e. it guarantees that each agent will arrive to a unique goal, thus all goals will be reached.

We conduct a thorough empirical evaluation of the proposed decentralized AMAPF solvers and compare them with the centralized baseline, i.e. TSWAP. We show that our most enhanced algorithm, i.e. TP-SWAP, indeed, outperforms the other suggested decentralized variants. Moreover, its flowtime (one of the widely used measures of the solution cost) is consistently lower (better) compared to the centralized TSWAP with the random goal assignment, and its makespan (another widely used measure of the solution cost) approaches the one of the latter when the number of agents increases. Overall, our findings pave the way to creating efficient fully decentralized AMAPF solvers that rely only on the local communication/observations, while still guarantying completeness.

2 Related Works

MAPF is a well-studied problem with a large number of different formulations [19]. The dominant number of papers devoted to *MAPF* assume the existence of a centralized controller and rely on a discretized representation of the workspace (e.g. a grid).

The classical *MAPF* problem formulation assumes that each agent is assigned a specific target to reach. It is known that finding an optimal solution, whether in terms of makespan or flowtime, is an NP-hard problem [25]. The techniques to solve *MAPF* are plentiful. Some of them are aimed at obtaining provably optimal [17] or bounded-suboptimal [2] solutions. These methods typically do not scale well to a large number of agents. On the other hand, if it is necessary to quickly find a solution and the cost is not of utmost importance, then the rule-based solvers can be applied [4]. A possible compromise between the solution cost and the performance may be provided by the prioritized planning [3], which often finds close-tooptimal solutions and is also fast and scalable. However, prioritized planning is incomplete in general.

Another variant of *MAPF* is the *anonymous/unlabeled MAPF* (*AMAPF*) [19, 12, 25], when the goals are not assigned to the agents initially. Unlike classical MAPF, AMAPF is always solvable [12]. Similarly to MAPF, most of the AMAPF solver are centralized. One of the key approaches is method [25], that allows finding makespanoptimal solutions in polynomial time (in contrast to the classical MAPF), although making practical problems on large graphs is inefficient. The paper [1] addresses this limitation and proposes improvements to find solutions more efficiently. Alternatively, there is the fast TSWAP solver [12], which allows obtaining suboptimal solutions.

The number of methods that consider both decentralized scenarios and unlabeled case is very limited. An adaptation of the centralized method of [21] to the decentralized setting was presented in the same paper. However, as was shown in [14], it does not guarantee the absence of collisions between the agents. The latter paper presents another decentralized method that is based on consideration of different number of potential goal assignments. This number can be prohibitively large. Moreover, the algorithm itself is not suited to operate in the non-empty environments.

Finally, a rapidly evolving research line is the one that suggests utilization of deep learning and multi-agent reinforcement learning for both centralized [7] and decentralized navigation and goal assignment, see [11, 8, 9]. However, the learnable methods are not typically able to provide any sorts of guarantees, require extensive training and often perform poorly on the problem instances that are not alike the ones used for training.

3 Problem Statement

We first present the centralized variant of the problem and then switch to the decentralized one.

Centralized AMAPF Consider a set \mathcal{N} of n agents, each confined to a connected, undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. There is a mapping $s : \mathcal{N} \to \mathcal{V}$ that assigns each agent to a specific start vertex, and a set $\mathcal{T} \subset \mathcal{V}$ of n target/goal vertices.

Time is discretized into timesteps. At each timestep, an agent can choose either to move to an adjacent vertex (a *move* action) or to remain at its current vertex (a *wait* action). A path for an agent *i* from vertex $v \in \mathcal{V}$ to vertex $v' \in \mathcal{V}$, denoted by $\pi^i(v, v')$, is defined as a sequence of actions that takes the agent from v to v'. The cost of the path is determined by the timestep at which the agent reaches its final destination. Additionally, we assume that once an agent reaches its target, it remains there and waits.

Paths should not include two types of conflicts:

- Vertex conflict: occurs between the agents i, j ∈ N iff they stay at the same vertex at the same timestep.
- Swapping conflict: occurs between the agents $i, j \in \mathcal{N}$ iff they traverse the same edge at the same timestep.

The problem is to find a sequence of actions (a path) for each agent such that (i) each individual path for agent i starts at the predefined start location s(i) and ends at one of the predefined goals $\tau \in \mathcal{T}$; (ii) all goal locations are reached; and (iii) all pairs of paths are conflict-free.

The quality of an AMAPF solution is typically evaluated using either *flowtime* or *makespan*, with lower values indicating better solutions. Flowtime is the sum of the costs of all paths in the solution, while makespan is the maximum cost among these paths. In this work, we do not impose a strict requirement to optimize the cost of the solution, but naturally, solutions with lower costs are preferable.

Decentralized AMAPF In a decentralized setup, each agent independently decides on its actions at each timestep, based on the limited information it obtains through local observation and communication. We assume that each agent has knowledge of the entire graph and can exchange information with other agents located within a distance of k edges from its current vertex. In our experiments, we use grid environments where the communication range is defined by a $(2k+1) \times (2k+1)$ cell area with the agent positioned at the center.

Moreover, we allow for chain communication between agents. This means that if agent i is within the communication range of agent j, and agent j is within the communication range of agent k, then agent i can exchange information with agent k through agent j, and vice versa.

For the purposes of this study, we abstract away from communication issues and assume instantaneous, error-free information exchange.



Figure 2. Examples of conflict and deadlock resolutions in TSWAP: (a) Illustration of the target-swapping mechanism when an agent occupying its target location blocks the path for another agent. (b) Illustration of the deadlock resolution mechanism, where a sequence of agents forms a loop, causing them to block each other's paths.

4 Methods

Our decentralized solver, TP-SWAP, is developed on the basis of the rule-based centralized method, TSWAP [12]. Therefore, we begin by explaining TSWAP and then gradually explore how it can be adapted for the decentralized seeting.

4.1 TSWAP

TSWAP [12] solves the AMAPF problem in two stages. In the first stage, it creates an initial *consistent goal assignment*, i.e. the one where each goal is uniquely assigned to a single agent, ensuring that no two agents share the same goal. In the second stage, the algorithm iteratively moves the agents toward their assigned goals and, if necessary, reassigns goals between them while always maintaining the consistency of the goal assignment.

Initial Goal Assignment In general, TSWAP can handle any consistent goal assignment. In their work, the authors of TSWAP explored several methods for initial goal assignment and evaluated them empirically. For our experiments, when using TSWAP as a centralized baseline, we adopted the assignment method that demonstrated the most promising results in the original paper.

Moving towards the goals with target swapping At each planning iteration, TSWAP sequentially examines all agents. For each agent i, it identifies the current vertex v and deterministically selects the next vertex v' based on the shortest path to its goal. If vertex v' is free, then it is marked as occupied by agent i.

If vertex v' is already occupied by another agent j, the agent i picks a wait action and checks two possible cases. First, if vertex v' is the target of agent j, agents i and j swap their targets, as illustrated in Fig. 2-a. Otherwise, TSWAP checks whether agent i is involved in a deadlock. A deadlock occurs when a loop sequence of agents (including i) is formed, such that each agent's next vertex in their shortest path is currently occupied by the next agent in the sequence. If a deadlock is detected, the targets of the agents within the sequence are rotated, meaning each agent is reassigned the target of the next agent in the loop. This scenario is depicted in Fig. 2-b.

After the planning iteration is complete, the algorithm moves each agent to its designated vertex, if necessary.

4.2 Decentralized TSWAP with Consistent Target Assignment

TSWAP can, in principle, be adapted to a decentralized setup where a centralized controller is not present, and agents have limited communication capabilities within a certain range. However, an initial consistent target assignment is still mandatory.

At each step, each agent can execute an iteration of the TSWAP algorithm using only the information available within its communication range. It is important to note that all agents within the same subgroup have access to the same information, ensuring that the outcome of the algorithm's execution will be consistent across all members of the group.

To correctly perform the TSWAP iteration, an agent needs specific information. First, all members of the subgroup must examine the agents in the same order. Therefore, each agent needs to know the priority pr of the other members. We assume that each agent i is identified by a unique number (e.g., the serial number of the robot). For simplicity, we will assume that the agent, its identifier, and its priority are identical, i.e., pr = i.

Second, each agent in the subgroup must be aware of the positions of the other subgroup members and their assigned targets. This information is essential for avoiding vertex conflicts and for performing target-swapping or deadlock resolution.

Finally, we must establish the communication conditions that are sufficient to safely and correctly execute the TSWAP iteration. Specifically, we need to ensure that for each agent i and its next vertex v', no other agent j is located at v' or plans to move there in the next step. To meet this requirement, each agent must have information about other agents within a range of at least two vertices from its own location. Therefore, the minimum necessary communication range must cover this distance.

Additionally, to detect and resolve deadlocks, an agent must have information about any chain of adjacent agents to which it belongs. This condition is satisfied if chain communication is allowed, as discussed in the section on the decentralized scenario in Section 3.

It is important to note that if the initial assignment is inconsistent, the described variant of TSWAP may fail to solve the instance. Thus, it can arguably be considered *semi*-decentralized, as achieving a consistent initial goal assignment requires some form of centralization, such as global information sharing.

In the next section, we will focus on fully decentralized AMAPF solvers (based on TSWAP) that do not require the initial assignment to be consistent.

4.3 Naive Fully-decentralized TSWAP

In a fully decentralized setting, each agent must independently choose its goal. This might be done randomly, or each agent might select the closest goal to its starting location. However, in such cases, multiple agents may head toward the same goal, while some goals might remain unassigned. A naive approach to resolve these inconsistencies involves memorizing which goals have already been achieved by other agents and selecting a different goal from those not on this list if necessary.

Specifically, at each step, every agent *i* must verify through local communication that its current goal (the one it is heading toward) is not already occupied by another agent. If it is, the agent should add this goal to a dedicated list of occupied goals (initialized as \emptyset) and then select another goal (randomly or according to a specific rule) from those not included in this list. Additionally, agents can

Algorithm 1: TP-SWAP Algorithm						
	Input: i – agent unique id; \mathcal{G} – graph; \mathcal{T} – set of all goals.					
1 2	$\tau \leftarrow \text{selectTarget}(\mathcal{G}, \mathcal{T});$ $pr \leftarrow \text{generateUniquePriority}(i);$					
3	$TP[\tau'] \leftarrow -\infty \ \forall \tau' \in \mathcal{T}; TP[\tau] \leftarrow pr;$					
4	while TRUE do					
5	$NH \leftarrow \text{identifyLocalAvailableAgents}();$					
6	$TA \leftarrow \{\}; PR \leftarrow \{\}; V \leftarrow \{\};$					
7	$V[j] \leftarrow \text{RECIEVEPOSITION}(j) \ \forall j \in NH;$					
8	$TA[j] \leftarrow \text{GetTarget}(j) \ \forall j \in NH;$					
9	$PR[j] \leftarrow \text{getPriority}(j) \ \forall j \in NH;$					
10	$TP[\tau] \leftarrow \max_{j \in NH} TP^j[\tau] \forall j \in NH;$					
11	$\tau', pr', v', TP' \leftarrow \text{TP-UPDATE}(i, NH, V, TA, PR, TP, \mathcal{T});$					
12	$\tau \leftarrow \tau'; pr \leftarrow pr'; TP \leftarrow TP';$					
13	MOVETO $(v');$					

exchange their lists of occupied goals to increase the likelihood that each agent will choose an unoccupied goal.

While this approach can indeed restore consistency in goal assignments and resolve conflicts where multiple agents are heading toward the same goal, it does not fully utilize the potential of local information exchange. As a result, agents may still need to explore all the targets individually until finding an unoccupied one. To address this limitation, we propose TP-SWAP – an improved, fully decentralized AMAPF solver that allows agents to select goals in a more informed manner.

4.4 TP-SWAP: Target-Priority Swapping For Decentralized AMAPF

The approach used to enhance the previously described fully decentralized AMAPF solver is based on two key ideas. First, it is advantageous not only to identify and memorize already occupied goals (and possibly share this information) but also to track the desired goals of other agents. Second, agent priorities can be utilized to restore consistency in goal assignments, with the possibility of exchanging these priorities, as will be explained later.

To achieve this, each agent individually maintains a *target-priority* assignment table, TP, which is a mapping from agent priorities (not identifiers!) to targets. The TP table is maintained throughout the entire process and is used by agents to resolve conflicts in their current goal assignments.

In addition to the target-priority mapping, agents also temporarily create and share information at each time step regarding their current locations (graph vertices), targets, and priorities. To facilitate this, temporary tables/dictionaries V, TA, and PR are introduced, which are generated from scratch at each iteration of the algorithm.

The general outline of TP-SWAP is presented in Algorithm 1. Similar to the naive approach, each agent individually selects its target (line 1) and determines its priority pr (line 2) before starting the movement.

Initially, the value for each target $\tau \in \{T\}$ in the table is set to $TP[\tau] = -\infty$. When an agent selects a goal $\tau \in \mathcal{T}$, it updates the corresponding entry in the table to $TP[\tau] = pr$ (line 3). This indicates that, at the start, the agent only has information about its own goal.

An iterative process then begins, guiding the agent toward its goal. Each iteration starts by identifying the agents available for communication (line 5) and gathering information about them. The agent collects information on the locations (V), current targets (TA), and current priorities (PR) of the members in the connected subgroup (lines 6-9).

Algorithm 2: TP-UPDATE Procedure

Input: i – agent unique id; NH – current subgroup; V – positions of agents in the current group; TA – target assignment of the current group; PR – table of priorities for agents in the current group; TP – table of targets and corresponding agents' priorities; \mathcal{T} – set of all targets; \mathcal{G} – graph. 1 $NH \leftarrow$ sortDecreasingPriorities(NH, PR); **2** for $j \in NH$ do if TP[TA[j]] > PR[j] then 3 $TA[j] \leftarrow \tau' \in \mathcal{T} \text{ s.t. closest to } V[j], TP[\tau'] \leq PR[j];$ $TP[TA[j]] \leftarrow PR[j];$ 1 5 6 for $j \in NH$ do if S[j] = TA[j] then 7 continue; 8 $v \leftarrow \text{NEXTVERTEX}(V[j], TA[j], \mathcal{G});$ 9 10 if $\exists k \in NH$ s.t. V[k] = v then if v = TA[k] then 11 $\begin{array}{l} \text{SWAP}(TA[j], TA[k]);\\ \text{SWAP}(PR[j], PR[k]); \end{array}$ 12 13 else if inDeadlock(j, V, TA) then 14 $D \leftarrow \text{GETDEADLOCKSEQUENCE}(j, V, TA);$ 15 16 ROTATE TARGETS PRIORITIES (D, TA, PR); 17 else $V[j] \leftarrow v;$ 18 19 return TA[i], PR[i], V[i], TP;

These tables are populated with up-to-date data relevant to the current group to which the agent belongs, and they are updated at each step. The position of a neighboring agent j at the current timestep is denoted as V[j], its target as TA[j], and its priority as PR[j].

Next, the agent updates its TP table using the collective knowledge of all subgroup members. If the agent receives information that a target τ was selected by another agent (who may not be part of the current subgroup) with a priority $pr' \in \mathbb{N}$ higher than what is currently recorded in the table $(pr' > TP[\tau])$, the agent updates the table to $TP[\tau] = pr'$ (line 10).

Once all necessary information has been gathered, the agent initiates the procedure for resolving assignment conflicts, updating targets and priorities, and selecting the next vertex. This is done by a (core) routine named TP-UPDATE(line 11). It eliminates inconsistencies in goal assignment within the current subgroup and prevents collisions, similarly to the TSWAP algorithm. A more detailed description of the target-priority update procedure is provided in the next section.

Finally, the algorithm updates the agent's current state and moves the agent to its next determined location (lines 12-13).

4.4.1 TP-UPDATE: Procedure to resolve conflicts in TP-table

The most critical component enabling the correct execution of the suggest algorithm is the target-priority update procedure, which is described below¹.

The TP-UPDATE procedure pursues two goals. First, it resolves assignment conflicts and updates the target-priority table, ensuring that the target assignment within the current connected subgroup is consistent. Second, it aligns the target-priority assignment of the current group with the available information. This alignment prevents any agent in the group from being assigned a target that has already been selected by another agent with a higher priority. This rule is enforced even if the higher-priority agent is not part of the current

¹ It is important to note that the operations described below are performed solely based on information available to a specific group of agents that are able to communicate and share information with each other.



Figure 3. An example of solving a decentralized AMAPF instance. Agents are depicted as disks (with the number showing their current priority). The dashed lines illustrate the agents' communication ranges. The red cells represent the goals that the agents need to reach.

group, provided that its information is present in the TP table.

This approach not only ensures the correct execution of the TSWAP algorithm but also prevents agents from selecting targets already claimed by higher-priority agents, thereby maintaining global consistency.

Let's examine Alg. 2 in more detail. To implement the key ideas, all agents within the current group are considered and sorted according to their priority (lines 1-5). If the TP table indicates that the target of a particular agent *i* has been selected by another agent with a higher priority, then the agent *i* will select a new target (line 4).

The new target is selected based on the following rules: (i) it must be the closest available target to the agent, and (ii) the priority recorded for it in the TP table must be lower than the current priority of the agent. In Section 4.4.3, we will demonstrate that there always exists a target satisfying these conditions.

After the TP table is updated (along with the auxiliary TA and PR tables), these updated tables are used as input for the iteration of the TSWAP algorithm (lines 6-18). Additionally, if two agents exchange targets during this TSWAP step, they must also exchange their priorities (lines 13, 16).

As a result, an agent gets an updated table TP along with the updated priority, target and a vertex to move to (line 19).

4.4.2 Running Example

Let's examine an example of solving an AMAPF problem in a decentralized fashion, as illustrated in Fig. 3.

Here, three agents (shown as blue, green, and orange disks) are confined to a grid. The goals (known to each of the agents) are depicted as the red squares labeled as τ_1 , τ_2 , τ_3 . The communication range of each agent is marked by the dashed lines matching the agents' colors.

At time t = 0, each agent selects the closest target and plans a path towards it, shown by the colored arrow. Initially, all agents choose the same target, τ_1 , and assign themselves priorities, displayed inside the circles. The agents then begin moving toward their selected targets. By time t = 4, the blue agent has reached τ_1 and is within communication range of the green agent. The green agent, having a lower priority, switches to a new target, τ_3 , and recalculates its path. Meanwhile, the orange agent continues moving toward τ_1 , unaware that a higher-priority agent has already reached it.

At time t = 7, the orange and green agents meet, and the orange agent receives information that both τ_1 and τ_3 have been claimed by higher-priority agents. The orange agent then chooses the remaining target, τ_2 .

Between t = 7 and t = 16, the green agent reaches τ_3 before the orange agent can reach τ_2 , blocking the orange agent's path. To resolve the conflict, the orange and green agents swap their goals and priorities. By time t = 16, the orange agent has successfully reached τ_3 , while the green agent is now heading toward τ_2 .

4.4.3 Theoretical Analysis

Theorem 1. *There exists a finite time t at which all goals will be achieved by the agents utilizing Alg. 1.*

Proof. Consider the following function ϕ :

$$\begin{aligned} \phi(t) &= \phi_1(t) + \phi_2(t) + C \cdot \phi_3(t) \\ \phi_1(t) &= \sum_{i \in \mathcal{N}} dist(\overline{V_t}[i], \overline{TA_t}[i]) \\ \phi_2(t) &= \sum_{i \in \mathcal{N}} \left| \{ j \, : \, j \in \mathcal{N}, \, \overline{TA_t}[j] \in \Pi(\overline{V_t}[i], \, \overline{TA_t}[i]) \} \right| \quad (1) \\ \phi_3(t) &= \sum_{pr \in \overline{PR_t}} \left| \{ \tau \, : \, \tau \in \mathcal{T}, \, TP_t^{pr}[\tau] \leq pr \} \right| \end{aligned}$$

where

- dist(v, v') the shortest path length between vertices $v, v' \in \mathcal{V}$,
- $\Pi(v, v')$ a set of vertices in the shortest path between $v, v' \in \mathcal{V}$,
- $\overline{V_t}$ the table of agents' positions on the graph at time t
- $\overline{TA_t}$ the global goal assignment at time t
- $\overline{PR_t}$ the global priority assignment at time t
- TP_t^{pr} the target-priority table at time t of agent with priority pr
- C finite scalar value, at least equal $\left(2 \max_{v,v' \in \mathcal{V}} dist(v,v') + 1\right)$

The function consists of the following components:

- \$\phi_1(t)\$ represents the total distance from each agent's current position to its assigned target.
- $\phi_2(t)$ counts for each agent the number of other agents that lie in its path.
- \$\phi_3(t)\$ counts for each agent the number of targets that agent either knows nothing about or knows are chosen by lower-priority agents.

Let's demonstrate that there exists a specific moment in time when all agents will have successfully achieved their goals. To establish this, it suffices to show that (*i*) the TP-UPDATE procedure is correct, i.e. it will always find consistent target assignment for a connected subgroup of agents (*ii*) the function ϕ is bounded from below, (*iii*) ϕ is decreasing, and (*iv*) ϕ reaching the lower bound can occur *iff* all agents have achieved their targets.

TP-UPDATE Correctness It can be observed that Alg. 2 (lines 2-5) creates and maintains a consistent goal assignment TA within a subgroup of agents. Crucially, each agent is guaranteed to find a new target if it must abandon its current one (line 4 of Alg. 2).



Figure 4. Maps that are used in the experiments.

To prove this, let's assign a new numbering to the agents, reversed according to their priorities (i.e., the agent with the highest priority is numbered 1, and the last one is n). We will prove by induction that an agent k can reject no more than k - 1 targets.

- **Base case:** the first agent can reject no targets (see lines 2-5 of Alg. 2).
- Induction step: assume that an agent k can reject no more than k 1 targets. Now, consider an agent k + 1. Suppose this agent can reject more than k targets. Among these rejected targets, at least two must have been targets that were not rejected by agent k. If agent k + 1 rejected these goals, it means they were at some point chosen by agents with priorities higher than k + 1. Since the priority associated with each goal does not decrease, there are two possibilities: either both of these goals were achieved by agent k (which is impossible), or one of the targets must have been achieved by an agent with a higher priority than k. This would mean agent k would have also had to reject that target. Hence, this is a contradiction.

Boundedness Functions ϕ_1 and ϕ_2 are each bounded from below by 0 for any assignment, while ϕ_3 is the sum of a finite set of nonnegative integers. This implies that ϕ_3 is also bounded from below. Consequently, the overall function ϕ is bounded below.

Monotonicity If a consistent target assignment is provided as input to Alg. 2 and remains unchanged between lines 2 and 5, the function $\phi_1(t) + \phi_2(t)$ strictly decreases, as established in the analysis of TSWAP in [12].

The function $\phi_3(t)$ is non-increasing because, at each timestep, the algorithm updates TP tables, ensuring the known number of targets selected by lower-priority agents, or left unselected, either stays the same or decreases (line 10 of Alg 1).

If ϕ_3 remains unchanged at a timestep, so $\phi_1 + \phi_2$ decreases unless all targets are achieved. Otherwise, if the subgroup configuration changes, ϕ_3 decreases. Although $\phi_1 + \phi_2$ may increase, the magnitude of change in $(C \cdot \phi_3)$ is always greater, ensuring ϕ decreases overall.

Since agent subgroups do not intersect, ϕ can be expressed as the sum of ϕ values for each subgroup, making it a sum of decreasing functions.

Targets Achievement Finally, we demonstrate that the function ϕ reaches its lower bound *iff* all agents have reached their goals.

It can be seen that, if all agents reach their targets, ϕ stops changing after the next time step, reaching its lower bound.

Conversely, assume ϕ is at its minimum, but at least one agent in a connected subgroup hasn't reached its target. If target assignments within this subgroup remain unchanged during lines 2-5 of Alg. 2, ϕ_3 stays constant while $\phi_1 + \phi_2$ decreases, leading to a contradiction.

If the target assignment changes, new goal information reduces ϕ_3 , decreasing ϕ , which contradicts the assumption that ϕ has stopped changing.

Note that while the proof idea is inspired by the completeness proof of the TSWAP algorithm, our scenario is significantly more



Figure 5. Average *flowtime*, *makespan* and standard deviations (shaded ahead) of the evaluated AMAPF solvers.

complex due to its decentralized nature. To the best of our knowledge, this is the first time completeness has been proven in such a context.

5 Empirical Evaluation

Algorithms Our evaluation considers the proposed decentralized AMAPF solvers and a centralized baseline, TSWAP. The latter is denoted as C-TSWAP (where "C" means centralized). In particular the following decentralized methods are evaluated: decentralized adaptation of TSWAP that relies on the initial random but consistent target assignment – D-TSWAP-C; naive fully decentralized AMAPF solver that does not rely on consistent initial target assignment – D-SWAP-N; its advanced variant that utilizes the suggested target and priority swapping procedure – TP-SWAP. All decentralized methods are implemented by us². For C-TSWAP we used the original authors' implementation that utilize the bottleneck assignment (as this way of assigning targets was shown to perform better on average in the original paper).

Problem Instances We utilized three grid maps from the MovingAI benchmark, which is well-known within the MAPF community [19, 20]. The selected maps include random-32-32-10, maze-32-32-4 and den404d (see Fig. 4). They all have roughly the same size (32×32) but differ significantly in topology.

For each map, we generate 250 different scenarios. Every scenario is a list of 100 start/target locations. To create an instance of n agents from a scenario, we take first n start-goal pairs from the list. In our experiments, we varied the number of agents from 10 to 100 with an increment of 10. In total, for each map and each number of agents, we have 250 different problem instances.

The communication range for the decentralized algorithms was set to an area 5×5 cells with an agent in the center. The primary performance indicator we are interested in is the solution quality, measured as makespan and flowtime.

Makespan and Flowtime Metrics Top row of Fig. 5 shows the average flowtime. The first important observation is that TP-SWAP, indeed, notably outperforms its naive decentralized counterpart, D-SWAP-N. The difference in their performance is getting more pronounced when the number of agents increases. On average, TP-SWAP is 2.3 times better than D-SWAP-N across all the maps and numbers of agents. The standard deviation of flowtime values is also consistently smaller for TP-SWAP.

² Source code: https://github.com/PathPlanning/TP-SWAP

n	5×5	11×11	21×21
20	819	508	414
40	1423	755	727
60	1906	994	989
80	2279	1297	1284
100	2464	1598	1560

 Table 1. Average *flowtime* for TP-SWAP with varying communication ranges on the maze-32-32-4 map

Interestingly, TP-SWAP also surpasses a semi-decentralized TSWAP variant with consistent initial target assignment, suggesting that TP-SWAP's initial assignment (where each agent picks the nearest target) is more effective, even if agents have to restore the assignment consistency.

The importance of the initial assignment is also exemplified by the performance of C-TSWAP (which is much better compared to the decentralized solvers). It confirms, that in case of smart centralized initial target assignment, one can achieve much better flowtime.

Regarding makespan (the bottom row of Fig. 5), similar trends emerge, though TP-SWAP does not outperform D-TSWAP-C. This indicates that consistent target assignment has a stronger impact on makespan than flowtime. Notably, TP-SWAP's makespan nearly converges with D-TSWAP-C as the number of agents increases. Moreover, other algorithms (except D-TSWAP-C) also show decreased makespan with more agents.

This decrease in makespan may result from higher agent density, which aids in quicker recovery of a consistent assignment. C-TSWAP also benefits, as more agents allow finding closer targets initially. In contrast, D-TSWAP-C's consistent assignment negates this effect, because its assignment is initially consistent, eliminating the need to restore consistency. Since the assignment is random, there is no advantage from goal proximity.

Impact of the Communication Range To investigate the impact of communication range on the performance of the proposed algorithm, we conducted a series of experiments on maze-32-32-4 map using three different communication range sizes: 5×5 , 11×11 , and 21×21 cells.

Table 1 displays the flowtime values for varying communication ranges across different numbers of agents. The results demonstrate a significant performance boost when the communication radius increases from 5×5 to 11×11 . However, this improvement diminishes when the radius is further extended to 21×21 . These findings suggest that expanding the communication range enhances problemsolving efficiency, but only up to a certain point. Beyond this point, the overlap in communication ranges likely causes most agents to form a single, large connected group, meaning that further increases in the communication range do not substantially improve agent connectedness.

Step	den312d		room-64-64-16	
limit	TP-SWAP	D-TSWAP-N	TP-SWAP	D-TSWAP-N
600	100 %	100 %	100 %	100 %
500	100 %	96 %	100 %	92 %
400	94 %	67 %	92 %	62 %
300	60 %	26 %	52 %	21 %
200	11 %	4 %	12 %	2 %

 Table 2.
 The success rates of the fully decentralized AMAPF solvers under different timestep limits.

Additional Comparison of the Fully Decentralized Solvers To get a more nuanced picture of how the performance of TP-SWAP differs from that of the basic fully-decentralized solver, D-TSWAP-N, we run additional experiments on two extra maps of slightly bigger size: room-64-64-16 and den312d (see Fig. 4). For each map, we generate 250 different instances involving 100 agents. Moreover, we introduce a timestep limit T_{max} – if the agents do not reach all the goals before the timestep T_{max} we count this run as *failure (success* otherwise). We vary T_{max} from 200 to 600 with an increment of 100. The results (success rate) are shown in Table 2.

As one can see, TP-SWAP solves a larger number of instances under any limit except 600 (when both methods solve all the tasks). Coupled with the results presented in the previous section, this confirms that the proposed target-priority swapping procedure is of utmost importance to the performance of the decentralized solvers.

Extended Results In the Supplementary material [5], extended empirical results are provided, including the ones on maps of different sizes and a more detailed analysis for various communication ranges, including statistics on the average number and size of connected subgroups.

6 Discussion and Limitations

This paper primarily addresses the theoretical aspects of decentralized multi-agent navigation, focusing on target selection and action choice, but several practical issues remain unaddressed.

Firstly, we assume that the agents possess all the necessary information about the others within their subgroup when choosing actions, which would require a specialized information-sharing mechanism in practice.

Secondly, we assume synchronized movements of agents. In the real world, e.g. when implementing our algorithm on robots, a decentralized motion synchronization might be needed, which, we believe, could be achieved via communication.

7 Conclusion

In this work, we have proposed a novel method to address the problem where a set of agents needs to reach a set of targets, and it does not matter which agent reaches a particular target. We focused on a particularly challenging and previously unsolved scenario in which the system is decentralized, allowing only local communication between the agents, and the initial goal assignment is inconsistent. To tackle this, we introduced an algorithm, TP-SWAP, specifically designed to solve this problem, and studied it both theoretically and empirically.

The experimental results demonstrated that TP-SWAP outperforms fully decentralized competitors in various scenarios and can achieve parity with, or even surpass, a semi-centralized solver that has access to consistent goal assignments, particularly in terms of flowtime. Future research directions include exploring more general AMAPF problem settings (e.g., colored MAPF), addressing communication issues, and implementing and evaluating our method on real robots.

Acknowledgements

The reported study was supported by the Ministry of Science and Higher Education of the Russian Federation under Project 075-15-2024-544.

References

- Z. A. Ali and K. Yakovlev. Improved Anonymous Multi-Agent Path Finding Algorithm. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17291–17298, 2024.
- [2] M. Barer, G. Sharon, R. Stern, and A. Felner. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Proceedings of The 7th Annual Symposium on Combinatorial Search (SoCS 2014)*, pages 19–27, 2014.
- [3] M. Čáp, P. Novák, A. Kleiner, and M. Seleckỳ. Prioritized planning algorithms for trajectory coordination of multiple mobile robots. *IEEE Transactions on Automation Science and Engineering*, 12(3):835–849, 2015.
- [4] B. de Wilde, A. W. ter Mors, and C. Witteveen. Push and rotate: cooperative multi-agent path planning. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems* (AAMAS 2013), pages 87–94, 2013.
- [5] S. Dergachev and K. Yakovlev. Supplementary materials for "decentralized unlabeled multi-agent pathfinding via target and priority swapping", 2024. URL https://zenodo.org/doi/10.5281/zenodo.13379279.
- [6] C. Ferner, G. Wagner, and H. Choset. Odrm* optimal multirobot path planning in low dimensional search spaces. In *Proceedings of The 2013 IEEE International Conference on Robotics and Automation (ICRA* 2013), pages 3854–3859. IEEE, 2013.
- [7] X. Ji, H. Li, Z. Pan, X. Gao, and C. Tu. Decentralized, unlabeled multiagent navigation in obstacle-rich environments using graph neural networks. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2021)*, pages 8936–8943, 2021.
- [8] A. Khan, C. Zhang, S. Li, J. Wu, B. Schlotfeldt, S. Y. Tang, A. Ribeiro, O. Bastani, and V. Kumar. Learning safe unlabeled multi-robot planning with motion constraints. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2019)*, pages 7558–7565, 2019.
- [9] A. Khan, V. Kumar, and A. Ribeiro. Large scale distributed collaborative unlabeled motion planning with graph policy gradients. *IEEE Robotics and Automation Letters*, 6(3):5340–5347, 2021.
- [10] J. Li, W. Ruml, and S. Koenig. Eecbs: A bounded-suboptimal search for multi-agent path finding. In *Proceedings of the 35th AAAI Conference* on Artificial Intelligence (AAAI 2021), pages 12353–12362, 2021.
- [11] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Proceedings of the Advances in neural information processing systems (NIPS 2017)*, 30, 2017.
- [12] K. Okumura and X. Défago. Solving simultaneous target assignment and path planning efficiently with time-independent execution. *Artificial Intelligence*, 321:103946, 2023.
- [13] K. Okumura, M. Machida, X. Défago, and Y. Tamura. Priority inheritance with backtracking for iterative multi-agent path finding. *Artificial Intelligence*, 310:103752, 2022.
- [14] D. Panagou, M. Turpin, and V. Kumar. Decentralized goal assignment and safe trajectory generation in multirobot networks via multiple lyapunov functions. *IEEE Transactions on Automatic Control*, 65(8): 3365–3380, 2019.
- [15] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. S. Kumar, S. Koenig, and H. Choset. Primal: Pathfinding via reinforcement and imitation multiagent learning. *IEEE Robotics and Automation Letters*, 4(3):2378– 2385, 2019.
- [16] G. Sharon, R. Stern, M. Goldenberg, and A. Felner. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial intelligence*, 195:470–495, 2013.
- [17] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant. Conflict-based search for optimal multiagent path finding. *Artificial Intelligence Journal*, 218:40–66, 2015.
- [18] A. Skrynnik, A. Andreychuk, M. Nesterova, K. Yakovlev, and A. Panov. Learn to Follow: Decentralized Lifelong Multi-Agent Pathfinding via Planning and Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17541–17549, 2024.
- [19] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, E. Boyarski, and R. Bartak. Multi-agent pathfinding: Definitions, variants, and benchmarks. *Symposium on Combinatorial Search (SoCS)*, pages 151–158, 2019.
- [20] N. Sturtevant. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games*, 4(2):144 148, 2012. URL http://web.cs.du.edu/~sturtevant/papers/benchmarks.pdf.
- [21] M. Turpin, N. Michael, and V. Kumar. Capt: Concurrent assignment and planning of trajectories for multiple robots. *The International Journal* of Robotics Research, 33(1):98–112, 2014.
- [22] G. Wagner and H. Choset. M*: A complete multirobot path plan-

ning algorithm with performance bounds. In *Proceedings of The 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems* (*IROS 2011*), pages 3260–3267, 2011.

- [23] B. Wang, Z. Liu, Q. Li, and A. Prorok. Mobile robot path planning in dynamic environments through globally guided reinforcement learning. *IEEE Robotics and Automation Letters*, 5(4):6932–6939, 2020.
- [24] Y. Wang, B. Xiang, S. Huang, and G. Sartoretti. Scrimp: Scalable communication for reinforcement-and imitation-learning-based multi-agent pathfinding. In *Proceedings of The 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2023)*, pages 9301– 9308, 2023.
- [25] J. Yu and S. M. LaValle. Multi-agent path planning and network flow. In Algorithmic Foundations of Robotics X: Proceedings of the Tenth Workshop on the Algorithmic Foundations of Robotics, pages 157–173. Springer, 2013.