Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)

Solving Simultaneous Target Assignment and Path Planning Efficiently with Time-Independent Execution

Keisuke Okumura, Xavier Défago

School of Computing, Tokyo Institute of Technology, Tokyo, Japan
{okumura.k, defago}@coord.c.titech.ac.jp

Abstract

Real-time planning for a combined problem of target assignment and path planning for multiple agents, also known as the unlabeled version of Multi-Agent Path Finding (MAPF), is crucial for high-level coordination in multi-agent systems, e.g., pattern formation by robot swarms. This paper studies two aspects of unlabeled-MAPF: (1) offline scenario: solving large instances by centralized approaches with small computation time, and (2) online scenario: executing unlabeled-MAPF despite timing uncertainties of real robots. For this purpose, we propose *TSWAP*, a novel sub-optimal complete algorithm, which takes an arbitrary initial target assignment then repeats one-timestep path planning with target swapping. TSWAP can adapt to both offline and online scenarios. We empirically demonstrate that *Offline TSWAP* is highly scalable; providing near-optimal solutions while reducing runtime by orders of magnitude compared to existing approaches. In addition, we present the benefits of *Online TSWAP*, such as delay tolerance, through real-robot demos.

Introduction

Target assignment and path planning for multiple agents, i.e., deciding where to go and how to go, are fundamental problems to achieve high-level coordination in multi-agent systems. This composite problem has attractive applications such as automated warehouses (Wurman, D'Andrea, and Mountz 2008), robot soccer (MacAlpine, Price, and Stone 2015), pattern formation of robot swarms (Turpin et al. 2014; Hönig et al. 2018b), a robot display (Alonso-Mora et al. 2012), to name just a few. These applications typically require real-time planning, i.e., planners have a limited time for deliberation until deadlines.

The problem above is a non-trivial composition of two fundamental problems: 1) *target assignment* is well-studied (Gerkey and Matarić 2004) with well-known efficient algorithms, such as the Hungarian algorithm (Kuhn 1955); 2) *path planning*, also known as *Multi-Agent Path Finding (MAPF)* (Stern et al. 2019), has been extensively studied in recent years. Given a graph, a set of agents, their initial locations, and their targets, a solution of MAPF maps collision-free paths to agents. This "labeled" MAPF regards targets as being assigned to each agent. This paper studies the "unlabeled" version of MAPF (*unlabeled-MAPF*) which considers agents and targets to be distinct, and hence requires to assign a target to each agent. In both labeled or unlabeled cases, the main objective is to minimize *makespan*, i.e., the maximum arrival time of agents.

Paradoxically, finding makespan-optimal solutions for unlabeled-MAPF is easier than for MAPF which is known to be NP-hard (Yu and LaValle 2013b; Ma et al. 2016). Indeed, unlabeled-MAPF has a polynomial-time optimal algorithm based on a reduction to maximum flow (Yu and LaValle 2013a); however, the size of the flow network is quadratic to the size of the original graph, making practical problems in large graphs (e.g., 500×500 grid) still challenging. Despite its importance, unlabeled-MAPF has received little attention compared to conventional MAPF, for which many scalable sub-optimal solvers have been developed (Surynek 2009; Wang and Botea 2011; de Wilde, ter Mors, and Witteveen 2013; Okumura et al. 2019).

The first objective of this paper is thus to *propose a centralized approach to solve large unlabeled-MAPF instances with sufficiently good quality in small computation time*. We present *Offline TSWAP*, a sub-optimal complete algorithm. Specifically, Offline TSWAP uses arbitrary assignment algorithms, then repeats one-timestep path planning with target swapping until all agents have reached targets.

We further extend TSWAP to an *online* version, aiming at *executing unlabeled-MAPF despite timing uncertainties of real robots*; the second objective of this paper. In practice, plan execution on robots is subject to timing uncertainties (e.g., kinematic constraints, unexpected delays, friction, clock drift). Even worse, the potential for unexpected interference increases with the number of agents because agents' actions usually depend on each other's; hence perfect on-time execution is unlikely to be expected.

To overcome this problem, we propose *Online TSWAP*, an online version of TSWAP based on the concept of time-independent planning (Okumura, Tamura, and Défago 2021). In other words, it abandons all timing assumptions (e.g., synchronization, traveling time, rotation time, delay probabilities) and regards the whole system as a transition system that changes its configuration according to atomic actions of agents. Regardless of movement timings, TSWAP ensures that all targets are eventually reached.

Our main contribution is proposing TSWAP to solve or execute unlabeled-MAPF, specifically; (1) *offline scenario:* we propose a novel algorithm and empirically demon-

strate that TSWAP is scalable and can yield near makespan-optimal solutions while reducing runtime by orders of magnitude in most cases when compared to the polynomial-time optimal algorithm (Yu and LaValle 2013a). Furthermore, TSWAP also yields good solutions with respect to *sum-of-costs*, another commonly used metric in MAPF studies. (2) *online scenario:* we formulate an online time-independent problem and propose a complete algorithm. We show the benefits of TSWAP, such as time independence and delay tolerance, through real-robot demos. Incidentally, (3) we also present efficient assignment algorithms with lazy evaluation of distances, assuming to use with TSWAP.

The remainder of this paper consists of the following eight sections: 1) related work about unlabeled-MAPF and time-independent execution methods, 2) formalization of offline and online time-independent problems of unlabeled-MAPF, 3) Offline TSWAP and its theoretical analysis, 4) assignment algorithms with lazy evaluation, 5) empirical results of offline planning, 6) Online TSWAP, 7) robot demos of online planning, and 8) conclusion. The technical appendix, code, and video are available on https://kei18.github.io/tswap.

Related Work

Target Assignment and Path Planning

The unlabeled-MAPF problem, also known as *anonymous MAPF*, consists of two sub-problems: (1) target assignment, more generally, task allocation, and (2) path planning. The *multi-robot task allocation* problems are a mature field (Gerkey and Matarić 2004). Path planning for multiple agents, embodied as MAPF, has been actively studied in recent years (Stern et al. 2019). We focus on reviews of related studies covering both aspects.

Unlike conventional MAPF, unlabeled-MAPF is always solvable (Kornhauser, Miller, and Spirakis 1984; Yu and LaValle 2013a; Ma et al. 2016). Among them, TSWAP relates to the analysis presented by Yu and LaValle (2013a) because both approaches use target swapping. Their analysis relies on optimal linear assignment whereas TSWAP works for any assignments. They also showed that unlabeled-MAPF has a *Pareto optimal structure* for makespan and sum-of-costs metrics (summation of traveling time of each agent; see the next section), i.e., there is an instance for which it is impossible to optimize both metrics simultaneously. Furthermore, they present a polynomial-time makespan-optimal algorithm, in contrast with conventional MAPF being known to be NP-hard (Yu and LaValle 2013b; Ma et al. 2016).

The *combined target assignment and path finding (TAPF)* problem (Ma and Koenig 2016), also called as *colored MAPF* (Barták, Ivanová, and Švancara 2021), generalizes both MAPF and unlabeled-MAPF by partitioning the agents into teams. The paper proposes a makespan-optimal algorithm for TAPF that combines Conflict-based Search (CBS) (Sharon et al. 2015), a popular optimal MAPF algorithm, with an optimal algorithm for unlabeled-MAPF. Hönig et al. (2018a) studied a sum-of-costs optimal algorithm for TAPF by extending CBS. They also proposed a bounded sub-optimal algorithm, called ECBS-TA; we compare TSWAP with ECBS-TA in the experiment. There is a study (Wagner, Choset, and Ayanian 2012) using another optimal MAPF algorithm (Wagner and Choset 2015) to solve the joint problem of target assignment and path planning.

The *multi-agent pickup and delivery (MAPD)* problem (Ma et al. 2017), motivated by applications in automated warehouses (Wurman, D'Andrea, and Mountz 2008), aims at making agents convey packages and has to solve target assignment and path planning jointly. Many approaches to MAPD have been proposed, e.g., (Ma et al. 2017; Liu et al. 2019; Okumura et al. 2019). Although MAPD is a problem different from unlabeled-MAPF, TSWAP is similar to an MAPD algorithm TPTS (Ma et al. 2017) in the sense that both algorithms swap assigned targets adaptively. One difference though is that, unlike TSWAP, TPTS sets additional conditions about start and target locations.

MAPF is a kind of *pebble motion* problem, in which objects are moved on a graph one-at-a-time, like a sliding tile puzzle. The unlabeled version of pebble motion has also been studied (Kornhauser, Miller, and Spirakis 1984; Călinescu, Dumitrescu, and Pach 2008; Goraly and Hassin 2010). However, in unlabeled-MAPF, agents can move simultaneously; different from those studies, TSWAP explicitly assumes this fact, resulting in practical outcomes.

Pattern formation of multiple agents (Oh, Park, and Ahn 2015) is one of the motivating examples of unlabeled-MAPF. Various approaches have been studied, e.g., (Alonso-Mora et al. 2011; Wang and Rubenstein 2020). We highlight two studies closely related to ours as follows. SCRAM (MacAlpine, Price, and Stone 2015) is a target assignment algorithm considering collisions and works only in open space without obstacles; hence its applications are limited. The assignment algorithm in this paper (Alg. 2) uses a scheme similar to SCRAM but differs in its use of lazy evaluation. Turpin et al. (2014) proposed a method that first solves the lexicographic bottleneck assignment (Burkard and Rendl 1991) then plans trajectories on graphs. To avoid collisions, the method uses the delay offset about when agents start moving, resulting in a longer makespan. TSWAP avoids using such offsets by swapping targets on demand.

Execution without Timing Assumptions

Ma, Kumar, and Koenig (2017) studied robust execution policies using offline MAPF plans as input, but assuming that agents might be delayed during the execution of timed schedules. The proposed Minimum Communication Policies (MCPs) make agents preserve two types of temporal dependencies: internal events within one agent and order relation of visiting a node. Regardless of delays, MCPs make all agents reach their destinations without conflicts. We later use MCPs for robot demos as a comparison of Online TSWAP.

Okumura, Tamura, and Défago (2021) studied *time-independent planning* to execute MAPF by modeling the whole system as a transition system that changes configurations according to atomic actions of agents. The online problem defined in this paper can be regarded as an unlabeled version of their model for conventional MAPF.

Problem Definition and Terminologies

Unlabeled-MAPF Instance A problem instance of *unlabeled-MAPF* is defined by a connected undirected graph $G = (V, E)$, a set of agents $A = \{a_1, \ldots, a_n\}$, a set of distinct initial locations $\mathcal{S} = \{s_1, \ldots, s_n\}$ and distinct target locations $\mathcal{T} = \{g_1, \ldots, g_m\}$, where $|\mathcal{T}| \leq |A|$.

Offline Problem Given an unlabeled-MAPF instance, let $\pi_i[t] \in V$ denote the location of an agent a_i at discrete time $t \in \mathbb{N}$. At each timestep t, a_i can move to an adjacent node, or can stay at its current location, i.e., $\pi_i[t+1] \in Neigh(\pi_i[t]) \cup \{\pi_i[t]\}$, where $Neigh(v)$ is the set of nodes adjacent to $v \in V$. Agents must avoid two types of conflicts: (1) *vertex conflict*: $\pi_i[t] \neq \pi_j[t]$, and, (2) *swap conflict*: $\pi_i[t] \neq \pi_j[t+1] \vee \pi_i[t+1] \neq \pi_j[t]$. A *solution* is a set of paths $\{\pi_1, \ldots, \pi_n\}$ such that a subset of agents occupies all targets at a certain timestep T. More precisely, assign a path $\pi_i = (\pi_i[0], \pi_i[1], \ldots, \pi_i[T])$ to each agent such that $\pi_i[0] = s_i$ and there exists an agent a_j with $\pi_j[T] = g_k$ for all $g_k \in \mathcal{T}$.

We consider four metrics to rate solutions:

- *makespan*: the first timestep when all targets are occupied, i.e., T.
- *sum-of-costs*: $\sum_i T_i$ where T_i is the minimum timestep such that $\pi_i[T_i] = \pi_i[T_i + 1] = \ldots = \pi_i[T]$.
- *maximum-moves*: the maximum of how many times each agent moves to adjacent nodes.
- *sum-of-moves*: the summation of moves of each agent.

Online Time-Independent Problem An *execution schedule* is defined by infinite sequence $\mathcal{E} = (a_i, a_j, a_k, \ldots)$ defining the order in which each agent is *activated* and can move one step.

Given an unlabeled-MAPF instance, a situation where all agents are at their initial locations, and an execution schedule $\mathcal{E}$, an agent a_i can move to an adjacent node if (1) it is a_i's turn in $\mathcal{E}$ and (2) the node is unoccupied by others. $\mathcal{E}$ is called *fair* when all agents appear infinitely often in $\mathcal{E}$. *Termination* is a configuration where all targets are occupied by a subset of agents simultaneously. An algorithm is called *complete* when termination is achieved within a finite number of activations for any fair execution schedules.

Given an execution schedule, we rate the efficiency of agents' behaviors according to two metrics: *maximum-moves* and *sum-of-moves*. Their definitions are the same as for the offline problem.

Remarks for Online Problem Since any complete algorithms must deal with any fair schedules, they inherently assume timing uncertainties. For simplicity, we assume that at most one agent is activated at any time, hence the execution is determined by a sequence over the agents. There is no loss of generality as long as an agent can atomically reserve its next node before each move. Note that we do not formally define sum-of-costs and makespan for the online problem since they should be measured according to actual time.

Other Assumptions and Notations For simplicity, we assume $|\mathcal{T}| = |A|$ unless explicitly mentioned. We denote the diameter of G by $diam(G)$, and its maximum degree by $\Delta(G)$. Let $dist\,(u, v)$ denote the shortest path length from $u \in V$ to $v \in V$. We assume the existence of admissible heuristics $h(u, v)$ for computing the shortest path length in constant time, i.e., $h(u, v) \leq dist\,(u, v)$, e.g., the Manhattan distance. This paper uses a simplified notation of the asymptotic complexity like $O(V)$ rather than $O(|V|)$.

Algorithm 1: **Offline TSWAP**

input: unlabeled-MAPF instance
output: plan $\boldsymbol{\pi}$

```
 1: get an initial assignment M: a set of pairs s ∈ S and g ∈ T
 2: a_i.v, a_i.g ← (s_i, g) ∈ M : for each agent a_i ∈ A
 3: t ← 0 {timestep}
 4: while ∃a ∈ A, a.v ≠ a.g do
 5:    for a ∈ A do
 6:       if a.v = a.g then continue
 7:       u ← nextNode (a.v, a.g)
 8:       if ∃b ∈ A s.t. b.v = u then
 9:          if u = b.g then
10:             swap targets of a, b; a.g ← b.g, b.g ← a.g
11:          else if detect deadlock for A' ⊆ A ∧ a ∈ A' then
12:             rotate targets of A'
13:          end if
14:       else
15:          a.v ← u
16:       end if
17:    end for
18:    t ← t + 1
19:    π_i[t] ← a_i.v : for each agent a_i ∈ A
20: end while
```

Offline TSWAP

This section presents *Offline TSWAP*, a sub-optimal path planning algorithm for the offline problem, which is complete for any initial target assignments. Here, an *assignment* is a set of pairs $s \in \mathcal{S}$ and $g \in \mathcal{T}$ such that all agents have a distinct target. Most proofs are deferred to the Appendix.

Algorithm Description

TSWAP assumes that an initial assignment is given externally, then mainly determines how to go but not only. This is because the initial assignment is potentially an unsolvable MAPF instance (e.g., see $t = 0$ at path planning in Fig. 1); we cannot apply MAPF solvers directly to design a complete unlabeled-MAPF algorithm for arbitrary assignments. The algorithm must consider swapping targets as necessary.

Algorithm 1 generates a solution $\boldsymbol{\pi}$ by moving agents incrementally towards their targets following the shortest paths, using the following function;

$$\mathsf{nextNode}\,(u, w) := \operatorname*{argmin}_{v \in Neigh(u) \cup \{u\}} dist\,(v, w)$$

nextNode is assumed to be deterministic; tie-break between nodes having the same scores is done deterministically.

Each agent a has two variables: $a.v$ is the current location, and $a.g$ is the current target. They are initialized by the initial assignment [Lines 1–2]. After that and until all agents reach their targets, one-timestep planning is repeated as follows [Lines 4–20]. If a is on its target

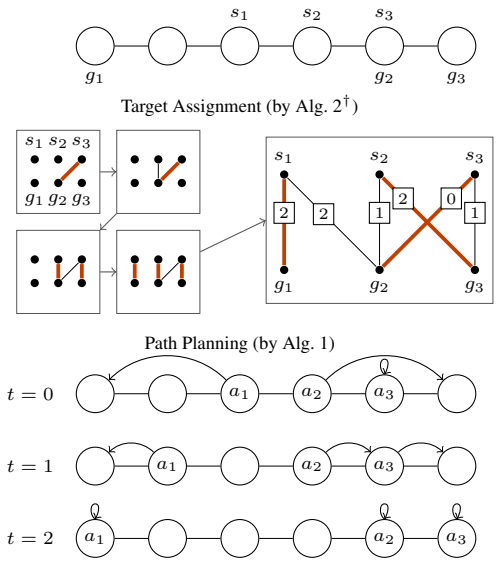

Figure 1: Example of Offline TSWAP with Alg. 2† as an assignment algorithm. An unlabeled-MAPF instance is shown at the top. The target assignment is illustrated in the middle, using a bipartite graph $\mathcal{B}$. The assignment $\mathcal{M}$ is denoted by red lines. The left corresponds to finding the bottleneck cost [Lines 3–15], i.e., incrementally adding pairs of an initial location and a target then updating the matching. The right part, with annotations of costs, corresponds to solving the minimum cost maximum matching problem [Line 16]. Two edges are added from the last situation due to line 12. Offline TSWAP (Alg. 1) is illustrated at the bottom. a_3, a_2, and a_1 repeat one-step planning. Current locations of agents, i.e., $a.v$, are shown within nodes. Arrows represent the targets, i.e., $a.g$. A target swapping happens between a_2 and a_3 at $t = 0$ [Line 10]. Note that we artificially assign s_2 to g_3 to show the example of target swapping.

$a.g$, it stays there ($a.v = a.g$). Otherwise, a attempts a move to the nearest neighbor of $a.v$ towards $a.g$, call it u [Line 7]. When u is occupied by another agent, the algorithm either performs target swapping or deadlock resolution [Lines 8–14]. Here, a deadlock is defined as follows. A set of agents $A' = (a_{i1}, a_{i2}, a_{i3}, \dots, a_{ij})$ is in a *deadlock* when $\mathsf{nextNode}(a_{i1}.v, a_{i1}.g) = a_{i2}.v \wedge \mathsf{nextNode}(a_{i2}.v, a_{i2}.g) = a_{i3}.v \wedge \dots \wedge \mathsf{nextNode}(a_{ij}.v, a_{ij}.g) = a_{i1}.v$. When detecting a deadlock for A', the algorithm "rotates" targets; $a_{i1}.g \leftarrow a_{ij}.g, a_{i2}.g \leftarrow a_{i1}.g, a_{i3}.g \leftarrow a_{i2}.g, \dots$ [Line 12]. The detection incrementally checks whether the next location of each agent is occupied by another agent and concurrently checks the existence of a loop.

Figure 1 shows an example of TSWAP, together with target assignment by Alg. 2† introduced in the next section.

Theoretical Analysis

Theorem 1. *Offline TSWAP (Algorithm 1) is complete for the offline problem.*

Proof. Let $\Pi(u, u') \subset V$ be a set of nodes in a shortest path from $u \in V$ to $u' \in V$, identified by $\mathsf{nextNode}$, except for u and u'. Consider the following potential function in Alg. 1.

$$\phi = \sum_{a \in A} \Big\{ dist\,(a.v, a.g) + \big|\{b \in A \mid b.g \in \Pi(a.v, a.g)\}\big| \Big\}$$

Observe that $\phi = 0$ means that the problem is solved. Furthermore, for each iteration of Lines 4–20, ϕ is non-increasing. We now proof that ϕ decreases for each iteration only when $\phi > 0$ by contradiction.

Suppose that $\phi (\neq 0)$ does not differ from the last iteration. Since $\phi \neq 0$, there are agents not on their targets. Let them be $B \subseteq A$. First, there is no swap operation by Line 10; otherwise, the second term of ϕ must decrease. Second, all agents in B do not move; otherwise, the first term of ϕ must decrease. Furthermore, for an agent $a \in B$, $\mathsf{nextNode}\,(a.v, a.g)$, let denote this as $a.u$, must be occupied by another agent $b \in B$; otherwise, a moves to $a.u$. This is the same for b, i.e., there is an agent $c \in B$ such that $c.v = b.u$. By induction, this sequence of agents must form a deadlock somewhere; however, by deadlock detection and resolution in Lines 11–12, both the first and second terms of ϕ must decrease. Hence, this is a contradiction. □

Any assignment algorithms can be applied to TSWAP because Theorem 1 does not rely on initial assignments. Furthermore, TSWAP can easily adapt to unlabeled-MAPF instances with $|A| > |\mathcal{T}|$ by assigning agents without targets to any non-target locations.

Proposition 1. *Offline TSWAP has upper bounds of;*

- *makespan:* $O(A \cdot diam(G))$
- *sum-of-costs:* $O(A^2 \cdot diam(G))$
- *maximum-moves:* $O(A \cdot diam(G))$
- *sum-of-moves:* $O(A \cdot diam(G))$

Compared to sum-of-costs, the upper bound on sum-of-moves is significantly reduced because it ignores all "wait" actions. The bound on sum-of-moves is tight in some scenarios, such as a line graph with all agents starting on one end and all targets on the opposite end. In general, the upper bound on makespan is greatly overestimated. We see empirically later that TSWAP yields near-optimal solutions for makespan depending on initial assignments.

Proposition 2. *Assume that the time complexity of* $\mathsf{nextNode}$ *and the deadlock resolution [Lines 11–12] in Alg. 1 are* α *and* β*, respectively. The time complexity of Offline TSWAP excluding Line 1 is* $O(A^2 \cdot diam(G) \cdot (\alpha + \beta))$.

From Proposition 2, TSWAP has an advantage in large fields compared to the time complexity $O(AV^2)$ of the makespan-optimal algorithm (Yu and LaValle 2013a) with a natural assumption that $E = O(V)$.

Target Assignment with Lazy Evaluation

Target assignment determines where to go; takes an unlabeled-MAPF instance as input, then returns an assignment, a set of pairs of an initial location and a target, as

Algorithm 2: **Bottleneck Assignment**

input: unlabeled-MAPF instance
output: $\mathcal{M}$: assignment, a set of pairs $s \in \mathcal{S}$ and $g \in \mathcal{T}$
1: initialize $\mathcal{M}$; Let $\mathcal{B}$ be a bipartite graph $(\mathcal{S}, \mathcal{T}, \emptyset)$
2: $\mathcal{Q}$: priority queue of tuple
$s \in \mathcal{S}$, $g \in \mathcal{T}$, real distance, and estimated distance
in increasing order of distance
(use real one if exists, otherwise use estimated one)
3: $\mathcal{Q}$.push $((s, g, \perp, h(s, g)))$: for each pair $s \in \mathcal{S}, g \in \mathcal{T}$
4: **while** $\mathcal{Q} \neq \emptyset$ **do**
5: $(s, g, d, \Delta) \leftarrow \mathcal{Q}$.pop()
6: **if** $d = \perp$ **then**
7: $\mathcal{Q}$.push $((s, g, dist\,(s, g)\,, \Delta))$; **continue**
8: **end if**
9: add a new edge (s, g) to $\mathcal{B}$
10: update $\mathcal{M}$ by finding an augmenting path on $\mathcal{B}$
11: **if** $|\mathcal{M}| = |\mathcal{T}|$ **then**
12: †optional: add all $(s', g', d', \cdot) \in \mathcal{Q}$ to $\mathcal{B}$ s.t. $d' = d$
13: **break**
14: **end if**
15: **end while**
16: †optional: $\mathcal{M} \leftarrow$ minimum cost maximum matching on $\mathcal{B}$

Algorithm 3: **Greedy Assignment with Refinement**

input: unlabeled-MAPF instance
output: $\mathcal{M}$: assignment, a set of pairs $s \in \mathcal{S}$ and $g \in \mathcal{T}$
1: initialize $\mathcal{M}$; initialize queue $\mathcal{U}$ by A
2: **while** $\mathcal{U} \neq \emptyset$ **do**
3: $a_i \leftarrow \mathcal{U}$.pop()
4: **while** true **do**
5: $g \leftarrow$ the non-evaluated nearest target from s_i
6: **if** $\nexists (s_j, g) \in \mathcal{M}$ **then**
7: add (s_i, g) to $\mathcal{M}$; **break**
8: **else if** $\exists (s_j, g) \in \mathcal{M} \wedge dist\,(s_i, g) < dist\,(s_j, g)$ **then**
9: replace $(s_j, g) \in \mathcal{M}$ by (s_i, g); $\mathcal{U}$.push(a_j); **break**
10: **end if**
11: **end while**
12: **end while**
13: **while** $\mathcal{M}$ is updated in the last iteration **do**
14: $(s_i, g_i) \leftarrow \underset{(s,g)\in\mathcal{M}}{\operatorname{argmax}}\, dist\,(s, g)$; $c_{\text{now}} \leftarrow dist\,(s_i, g_i)$
15: **for** $(s_j, g_j) \in \mathcal{M}$ **do**
16: **if** $h(s_j, g_i) \geq c_{\text{now}}$ **then continue** {for lazy evaluation}
17: $c_{\text{swap}} \leftarrow \max\,(dist\,(s_j, g_i)\,, dist\,(s_i, g_j))$
18: **if** $c_{\text{swap}} < c_{\text{now}}$ **then** swap g_i and g_j of $\mathcal{M}$; **break**
19: **end for**
20: **end while**

output. An initial assignment is crucial for TSWAP. Ideal assignment algorithms are quick, scalable, and with reasonable quality for solution metrics (e.g., makespan). It is possible to apply conventional assignment algorithms (Kuhn 1955). However, costs (i.e., distances) for each start-target pair are unknown initially, which is typically computed via breadth-first search with time complexity $O(A(V+E))$. This would be a non-negligible overhead. We thus present two examples that efficiently solve target assignment with *lazy evaluation*, which avoids exhaustive distance evaluation.

Bottleneck Assignment

Algorithm 2 aims to minimize makespan by solving the bottleneck assignment (Gross 1959), i.e., assign each agent to one target while minimizing the maximum cost, regarding distances between initial locations and targets as costs.

The algorithm incrementally adds pairs of initial location and target to a bipartite graph $\mathcal{B}$ [Line 9], in increasing order of their distances using a priority queue $\mathcal{Q}$. $\mathcal{B}$ is initialized as $(\mathcal{S}, \mathcal{T}, \emptyset)$ [Line 1]. This iteration continues until all targets are matched to initial locations, i.e., agents [Line 11]. At each iteration, the maximum bipartite matching problem on $\mathcal{B}$ is solved [Line 10]. In general, the Hopcroft-Karp algorithm (Hopcroft and Karp 1973) efficiently solves this problem in $O(\sqrt{V'}E')$ runtime for any bipartite graph (V', E'), but we use the reduction to the maximum flow problem and the Ford-Fulkerson algorithm (Ford and Fulkerson 1956). The basic concept of this algorithm is finding repeatedly an *augmenting path*, i.e., a path from source to sink with available capacity on all edges in the path, then making the flow along that path. Such paths are found, e.g., via depth-first or breadth-first search. Here, finding a single augmenting path in $O(E')$ runtime is sufficient to update the matching because the number of matched pairs increases at most once for each adding.

The algorithm uses *lazy evaluation* of real distance [Line 7]. We use the priority queue $\mathcal{Q}$ [Line 2] and admissible heuristics h [Line 3], then evaluate the real distance as needed. $\perp$ denotes that the corresponding real distance has not been evaluated yet. The lazy evaluation contributes to speedup of the target assignment, as we will see later.

The algorithm *optionally* solves the minimum cost maximum matching problem [Line 16], aiming at improving the sum-of-costs metric. The problem can be solved by reducing to the minimum cost maximum flow problem then using the successive shortest path algorithm (Ahuja, Magnanti, and Orlin 1993). Note that when finding the bottleneck cost, all edges in $\mathcal{Q}$ with their costs equal to the bottleneck cost are added to $\mathcal{B}$ to improve the sum-of-costs metric of the assignment [Line 12]. This operation includes lazy evaluation similar to the main loop [Line 4–15]. We denote the corresponding algorithm as Alg. $2^{\dagger}$.

Proposition 3. *The time complexity of Algorithm* $2^{(\dagger)}$ *is* $O\left(\max\left(A(V+E), A^4\right)\right)$.

Greedy Assignment with Refinement

Algorithm 3 aims at finding a reasonable assignment for makespan as quickly as possible, which uses;

- *greedy assignment* [Lines 1–12]; assigns one target to one agent step by step, while allowing reassignment if a better assignment will be expected [Lines 8–10].
- *iterative refinement* [Lines 13–20]; swaps targets of two agents until no improvements are detected.
- *lazy evaluation* of distances for start-target pairs, implemented by pausing the breadth-first search as soon as the query start-target pair is in the search tree.

This algorithm is expected to run in a very short time;

Proposition 4. *The time complexity of Algorithm 3 is* $O(A(V+E))$.

Algorithm 3 presents the refinement for makespan but the refinement for sum-of-costs is straightforward (see Alg. 5 in the Appendix).

Evaluation of Offline Planning

The experiments aim at demonstrating that Offline TSWAP is efficient, i.e., it returns near-optimal solutions within a short time and scales well, depending on initial assignments. In particular, this section has three aspects: (1) illustrating the effect of initial assignments including the proposed assignment algorithms, (2) comparing with the makespan-optimal polynomial-time algorithm (Yu and LaValle 2013a), (3) assessing another metric, sum-of-costs. We carefully picked up several 4-connected grids from MAPF benchmarks (Stern et al. 2019) as a graph G, shown in Fig. 2; they are common in MAPF studies. The simulator was developed in C++ and the experiments were run on a laptop with Intel Core i9 2.3 GHz CPU and 16 GB RAM. For each setting, we created 50 instances while randomly generating starts and targets. Implementation details of (Yu and LaValle 2013a) are described in the Appendix. Throughout this section, the runtime evaluation of TSWAP includes both target assignment and path planning.

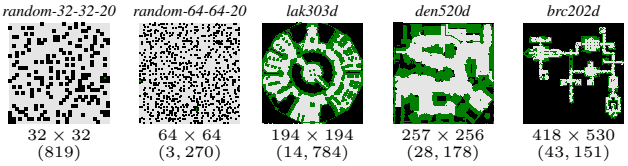

Figure 2: Used maps. $|V|$ is shown with parentheses.

Effect of Initial Target Assignment

The first part evaluates the effect of initial assignments on TSWAP while varying $|A|$. We tested Alg. 2 (bottleneck; minimizing maximum distance), Alg. $2^\dagger$ (with min-cost maximum matching), Alg. 3 (greedy with refinement for makespan), Alg. 5 (for sum-of-costs), naive greedy assignment (Avis 1983), and optimal linear assignment (minimizing total distances) solved by the successive shortest path algorithm (Ahuja, Magnanti, and Orlin 1993). Note that these assignment algorithms do not consider inter-agent collisions. The last two used distances for start-target pairs obtained by the breadth-first search as costs. To assess the effect of lazy evaluation, we also tested the adapted version of Alg. $2^\dagger$ and Alg. 3 without lazy evaluation, denoted as Alg. $2^{\dagger *}$ and Alg. 3^*.

Table 1 summarizes the results on *random-64-64-20*. In summary, Alg. 2 contributes to finding good solutions for makespan. Algorithm $2^\dagger$ significantly improves sum-of-costs. Algorithm 3 and Alg. 5 are blazing fast while solution qualities outperform those of the naive greedy assignment. The lazy evaluation speedups each assignment algorithm. The optimal linear assignment requires time because its time complexity is $O(A^3)$.

$\|A\|$		Alg.2	Alg.$2^\dagger$	Alg.$2^{\dagger *}$	Alg.3	Alg.3^*	Alg.5	greedy	linear
110	R	6	10	17	**4**	12	**4**	12	23
	M	**17**	**17**	**17**	20	20	36	84	36
	S	1079	**937**	**937**	1139	1136	**958**	1396	**940**
500	R	72	174	213	**15**	78	22	77	602
	M	**10**	**11**	**11**	13	13	34	79	32
	S	2595	**2169**	**2169**	2878	2860	2546	4653	2429
1000	R	335	757	882	**28**	233	58	221	3811
	M	**9**	**9**	**9**	11	11	28	71	26
	S	3591	**2922**	**2922**	4020	4043	3695	7571	3491
2000	R	1453	3035	3205	**66**	784	188	725	32944
	M	**8**	**7**	**7**	10	10	24	56	23
	S	4670	**3469**	**3469**	5200	5244	5465	12292	5122

Table 1: The results of TSWAP with different assignments in *random-64-64-20*. The second column corresponds to metrics as follows. R: runtime (ms). M: makespan. S: sum-of-costs. Bold characters are based on 95% confidence intervals of the mean in the Appendix, which also presents the result on another map *lak303d*.

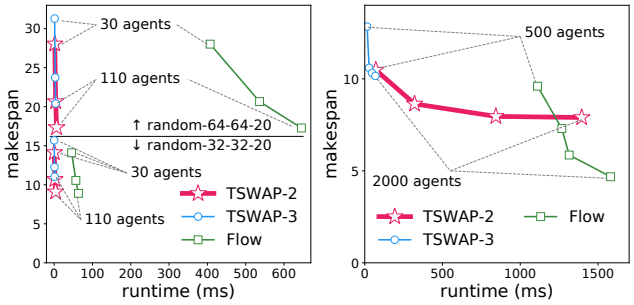

Figure 3: The average makespan and runtime. Alg. 2 (TSWAP-2) and Alg. 3 (TSWAP-3) were used in TSWAP. "Flow" is the optimal algorithm. *left*: $|A| \in \{30, 70, 110\}$. *right*: $|A| \in \{500, 1000, 1500, 2000\}$ on *random-64-64-20*.

Makespan-optimal Algorithm v.s. TSWAP

This part is further divided into two: (1) assessing scalability for both G and A, and (2) testing the solvers in large graphs. Figure 3 and Table 2 summarize the results.

Figure 3 (left) displays the average makespan and runtime of "quadrupling" the size of G, i.e., those of *random-64-64-20*, regarding the results of *random-32-32-20* as a baseline. Figure 3 (right) shows dense situations ($|A| \geq |V|/8$). The main observations are: (1) TSWAP quickly yields near-optimal solutions in non-dense situations with either Alg. 2 or Alg. 3. (2) The runtime of TSWAP remains small when enlarging G while the optimal algorithm increases dramatically. (3) As agents increase, the runtime of TSWAP with Alg. 2 quickly increases compared to the optimal algorithm, because Alg. 2 is quartic on $|A|$ (see Prop. 2). Meanwhile, TSWAP with Alg. 3 immediately yields solutions even with a few thousand agents. Note that, as the number of agents increases, the optimal makespan decreases because we set initial locations and targets randomly.

Table 2 shows the results on large graphs with a timeout of

		runtime (sec)			success rate(%)			sub-opt	
map	$\|A\|$	Flow	Alg.2	Alg.3	Flow	Alg.2	Alg.3	Alg.2	Alg.3
lak303d	100	26.2	0.0	**0.0**	100	100	100	1.001	1.001
	500	60.6	0.6	**0.1**	100	100	100	1.009	1.022
	1000	54.7	3.5	**0.2**	100	100	100	1.064	1.073
	2000	56.3	25.4	**0.4**	100	100	100	1.340	1.358
den520d	100	46.0	0.0	**0.0**	100	100	100	1.000	1.052
	500	67.5	0.3	**0.1**	100	100	100	1.003	1.118
	1000	82.3	1.6	**0.2**	98	100	100	1.014	1.097
	2000	89.8	9.3	**0.4**	100	100	100	1.043	1.169
brc202d	100	141.9	0.1	**0.1**	60	100	100	1.000	1.001
	500	214.5	0.7	**0.3**	48	100	100	1.001	1.003
	1000	238.5	2.7	**0.6**	42	100	100	1.002	1.007
	2000	230.2	14.8	**1.1**	16	100	100	1.021	1.026

Table 2: The results in large graphs. "Flow" is the optimal algorithm. The scores are averages over instances solved by all solvers. The sub-optimality ("sub-opt") is for makespan, dividing the makespan of TSWAP by the optimal scores.

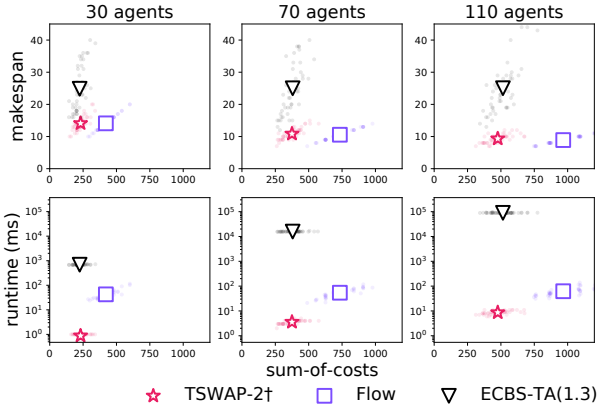

Figure 4: The results for the sum-of-costs metric. We used *random-32-32-20*. The average scores are plotted. We also show scatter plots of 50 instances by transparent points.

5 min. The optimal algorithm took time to return solutions or sometimes failed before the timeout, whereas TSWAP succeeded in all cases in a comparatively very short time, thus highlighting the need for sub-optimal algorithms of unlabeled-MAPF. In addition, TSWAP yields high-quality solutions for the makespan.

Sum-of-costs Metric

We next evaluate the sum-of-costs metric. As a baseline, we used ECBS-TA (Hönig et al. 2018a), which yields bounded sub-optimal solutions with respect to the sum-of-costs. The implementation of ECBS-TA was obtained from the authors. We used *random-32-32-20* with 30, 70, and 110 agents. The sub-optimality of ECBS-TA was set to 1.3, which was adjusted to solve problems within the acceptable time (5 min). TSWAP used Alg. $2^\dagger$. Note that we preliminary confirmed that ECBS-TA in denser situations failed to return solutions within a reasonable time.

Figure 4 shows that TSWAP yields solutions with acceptable quality while reducing computation time (lower, vertical axis) by orders of magnitude compared to the others; the quality of sum-of-costs (horizontal axis) is competitive with ECBS-TA, with makespan quality close to optimal (upper, vertical axis). TSWAP is significantly faster than ECBS-TA because, unlike ECBS-TA, TSWAP uses a one-shot target assignment and a simple path planning process.

Algorithm 4: **Online TSWAP**

input: unlabeled-MAPF instance
offline phase
1: get an initial assignment $\mathcal{M}$
2: $a_i.v, a_i.g \leftarrow (s_i, g) \in \mathcal{M}$: for each agent $a_i \in A$
online phase; when $a \in A$ **is activated**
3: execute Lines 6–16 in Algorithm 1
4: **if** $a.v$ is updated **then** move a to $a.v$

Online TSWAP

TSWAP is not limited to offline planning and can also adapt to online planning with timing uncertainties. Algorithm 4 presents *Online TSWAP* to solve the online time-independent problem. Before execution, TSWAP assigns targets to each agent [Lines 1–2]. During execution, the online version runs the procedure of the offline version for one agent [Line 3]. If the virtual location (i.e., $a.v$) is updated, then let the agent *actually* moves there [Line 4].

Theorem 2. *Online TSWAP (Algorithm. 4) is* complete *for the online time-independent problem.*

Proof. The most part is the same as for the offline version (Theorem 1). The problem assumes a fair execution schedule, therefore, ϕ must decrease within the sufficiently long period during which all agents are activated at least once; otherwise, $\phi = 0$. □

Proposition 5. *Regardless of execution schedules, Online TSWAP has upper bounds of;*

- *maximum-moves:* $O(A \cdot diam(G))$
- *sum-of-moves:* $O(A \cdot diam(G))$

Demos of Online Planning

We next rate Online TSWAP (with Alg. $2^\dagger$) with real robots.

Platform We used *toio* robots (https://toio.io/) to implement TSWAP. The robots, connected to a computer via Bluetooth, evolve on a specific playmat and are controllable by instructions of absolute coordinates.

Usage The robots were controlled in a *centralized* style, described as follows. We created a virtual grid on the playmat; the robots followed the grid. A central server (a laptop) managed the locations of all robots and issued the instructions (i.e., where to go) to each robot step-by-step. The instructions were periodically issued to each robot (per 50 ms) but they were issued *asynchronously* between robots. The code was written in Node.js.

Figure 5: Demo of time-independence of Online TSWAP. We prepared three scenarios with identical initial (left for each) and terminal (right for each) configurations. A graph is illustrated on the leftmost image. We further illustrated colored triangles to distinguish each robot. Although the experimenter disturbed robots' progression with chopsticks during the execution (middle and right settings), all robots reach the targets while flexibly swapping their assigned targets.

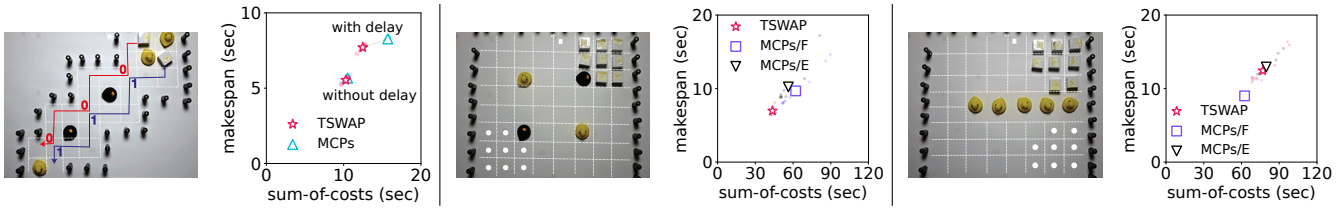

Figure 6: Robot demos with average scores. We also show scatter plots of 10 repetitions by transparent points. *left: Delay tolerance.* The initial configuration and the offline plan, accompanied by temporal orders, used in MCPs are illustrated. Targets are shown by the tips of arrows. We made the red one's speed half in one scenario (with delay). *middle & right: Eight robots execution.* The initial configurations and virtual grids are shown. Targets are marked by white-filled circles. "MCPs/F" represents MCPs with an offline plan obtained by the makespan-optimal algorithm. "MCPs/E" represents those with ECBS-TA. The sub-optimality of ECBS-TA was 1.12 and 1.2 respectively; obtained incrementally increasing values from 1.0 until solved.

Demo of Time Independence We first show a three-robots demo highlighting the time independence of TSWAP. In this demo, the experimenter disturbed robots' progression during the execution (Fig. 5). No matter when the robots move and no matter what order the robots move, the online problem is certainly solved.

Demo of Delay Tolerance We next show a simple setting that highlights the delay tolerance of TSWAP. The comparison baseline is MCPs (Ma, Kumar, and Koenig 2017). We prepared two scenarios with two robots while manipulating robots' speed. In one scenario, robots move at usual speeds. In another scenario, one of them halves its speed, i.e., with delays, assuming an accident happened.

Figure 6 shows the configuration and both makespan and sum-of-costs results over 10 repetitions measured in actual time. Without delays, TSWAP and MCPs do not differ in the results; however, with delays, the results of TSWAP are clearly better for both metrics. In MCPs, disturbing/delaying one robot may critically affect the entire performance because it cannot change the temporal orders during the execution, while TSWAP can adaptively address such effects.

Demo with Eight Robots We finally present eight robots demos in Fig 6. Two scenarios were carefully designed to clarify the characteristics of TSWAP. MCPs' offline plans were obtained by the makespan-optimal algorithm (Yu and LaValle 2013a) and ECBS-TA. In the first scenario, TSWAP performed better than the other two, but not so in the second scenario. This is because the latter has bottleneck nodes that all shortest paths from starts to targets use (two middle nodes in the second column). Since TSWAP makes robots move following the shortest paths, all robots must use the bottleneck nodes, causing unavoidable congestion. In contrast, the former does not have such bottleneck nodes, resulting in a small execution time compared to the others.

Conclusion

This paper presented a novel algorithm TSWAP to solve or execute unlabeled-MAPF; simultaneous target assignment and path planning problems for indistinguishable agents. TSWAP is complete for both offline and online problems regardless of initial assignments. We empirically demonstrated that it can solve large offline instances with acceptable solution quality in a very short time, depending on assignment algorithms. It can also work in online situations with timing uncertainties as shown in our robot demos.

Future directions are: (1) applying TSWAP to other situations, e.g., lifelong scenarios, and (2) decentralized execution with only local interactions by Online TSWAP.

Acknowledgments

We thank the anonymous reviewers for their many insightful comments. This work was partly supported by JSPS KAKENHI Grant Numbers 20J23011. Keisuke Okumura thanks the support of the Yoshida Scholarship Foundation.

References

Ahuja, R. K.; Magnanti, T. L.; and Orlin, J. B. 1993. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall.

Alonso-Mora, J.; Breitenmoser, A.; Rufli, M.; Siegwart, R.; and Beardsley, P. 2011. Multi-robot system for artistic pattern formation. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*.

Alonso-Mora, J.; Breitenmoser, A.; Rufli, M.; Siegwart, R.; and Beardsley, P. 2012. Image and animation display with multiple mobile robots. *The International Journal of Robotics Research (IJRR)*.

Avis, D. 1983. A survey of heuristics for the weighted matching problem. *Networks*.

Barták, R.; Ivanová, M.; and Švancara, J. 2021. From Classical to Colored Multi-Agent Path Finding. In *Proc. Int. Symp. on Combinatorial Search (SOCS)*.

Burkard, R. E.; and Rendl, F. 1991. Lexicographic bottleneck problems. *Operations Research Letters*.

Călinescu, G.; Dumitrescu, A.; and Pach, J. 2008. Reconfigurations in graphs and grids. *SIAM Journal on Discrete Mathematics*.

de Wilde, B.; ter Mors, A.; and Witteveen, C. 2013. Push and rotate: cooperative multi-agent path planning. In *Proc. Int. Joint Conf. on Autonomous Agents & Multiagent Systems (AAMAS)*.

Ford, L. R.; and Fulkerson, D. R. 1956. Maximal flow through a network. *Canadian journal of Mathematics*.

Gerkey, B. P.; and Matarić, M. J. 2004. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research (IJRR)*.

Goraly, G.; and Hassin, R. 2010. Multi-color pebble motion on graphs. *Algorithmica*.

Gross, O. 1959. The bottleneck assignment problem. Technical report, RAND CORP SANTA MONICA CALIF.

Hönig, W.; Kiesel, S.; Tinka, A.; Durham, J.; and Ayanian, N. 2018a. Conflict-based search with optimal task assignment. In *Proc. Int. Joint Conf. on Autonomous Agents & Multiagent Systems (AAMAS)*.

Hönig, W.; Preiss, J. A.; Kumar, T. S.; Sukhatme, G. S.; and Ayanian, N. 2018b. Trajectory planning for quadrotor swarms. *IEEE Transactions on Robotics (TR-O)*.

Hopcroft, J. E.; and Karp, R. M. 1973. An n^5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*.

Kornhauser, D. M.; Miller, G.; and Spirakis, P. 1984. *Coordinating pebble motion on graphs, the diameter of permutation groups, and applications*. Master's thesis, MIT.

Kuhn, H. W. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly*.

Liu, M.; Ma, H.; Li, J.; and Koenig, S. 2019. Task and Path Planning for Multi-Agent Pickup and Delivery. In *Proc. Int. Joint Conf. on Autonomous Agents & Multiagent Systems (AAMAS)*.

Ma, H.; and Koenig, S. 2016. Optimal Target Assignment and Path Finding for Teams of Agents. In *Proc. Int. Joint Conf. on Autonomous Agents & Multiagent Systems (AAMAS)*.

Ma, H.; Kumar, T. S.; and Koenig, S. 2017. Multi-agent path finding with delay probabilities. In *Proc. AAAI Conf. on Artificial Intelligence (AAAI)*.

Ma, H.; Li, J.; Kumar, T. K. S.; and Koenig, S. 2017. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *Proc. Int. Joint Conf. on Autonomous Agents & Multiagent Systems (AAMAS)*.

Ma, H.; Tovey, C.; Sharon, G.; Kumar, T. S.; and Koenig, S. 2016. Multi-agent path finding with payload transfers and the package-exchange robot-routing problem. In *Proc. AAAI Conf. on Artificial Intelligence (AAAI)*.

MacAlpine, P.; Price, E.; and Stone, P. 2015. SCRAM: scalable collision-avoiding role assignment with minimal-makespan for formational positioning. In *Proc. AAAI Conf. on Artificial Intelligence (AAAI)*.

Oh, K.-K.; Park, M.-C.; and Ahn, H.-S. 2015. A survey of multi-agent formation control. *Automatica*.

Okumura, K.; Machida, M.; Défago, X.; and Tamura, Y. 2019. Priority Inheritance with Backtracking for Iterative Multi-agent Path Finding. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*.

Okumura, K.; Tamura, Y.; and Défago, X. 2021. Time-Independent Planning for Multiple Moving Agents. In *Proc. AAAI Conf. on Artificial Intelligence (AAAI)*.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence (AIJ)*.

Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T.; et al. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Proc. Int. Symp. on Combinatorial Search (SOCS)*.

Surynek, P. 2009. A novel approach to path planning for multiple robots in bi-connected graphs. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*.

Turpin, M.; Mohta, K.; Michael, N.; and Kumar, V. 2014. Goal assignment and trajectory planning for large teams of interchangeable robots. *Autonomous Robots*.

Wagner, G.; and Choset, H. 2015. Subdimensional expansion for multirobot path planning. *Artificial Intelligence (AIJ)*.

Wagner, G.; Choset, H.; and Ayanian, N. 2012. Subdimensional Expansion and Optimal Task Reassignment. In *Proc. Int. Symp. on Combinatorial Search (SOCS)*.

Wang, H.; and Rubenstein, M. 2020. Shape Formation in Homogeneous Swarms Using Local Task Swapping. *IEEE Transactions on Robotics (TR-O)*.

Wang, K.-H. C.; and Botea, A. 2011. MAPP: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. *Journal of Artificial Intelligence Research (JAIR)*.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2008. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*.

Yu, J.; and LaValle, S. M. 2013a. Multi-agent path planning and network flow. In *Algorithmic foundations of robotics X*.

Yu, J.; and LaValle, S. M. 2013b. Structure and intractability of optimal multi-robot path planning on graphs. In *Proc. AAAI Conf. on Artificial Intelligence (AAAI)*.