Conveyor Parcel Routing with Order-Contiguous Arrivals

Takuro Kato[1] and **Keisuke Okumura**[2]
[1]Toyota Industries Corporation, Japan
[2]National Institute of Advanced Industrial Science and Technology (AIST), Japan
takuro.kato@mail.toyota-shokki.co.jp, okumura.k@aist.go.jp

Abstract

In warehouse logistics, parcels released from the outfeed of an automated storage system must be routed through conveyor networks to workstations. Beyond collision avoidance, practical operations impose an additional requirement of *order-contiguous arrivals*: at each delivery point, parcels belonging to the same order must arrive as a consecutive block in the arrival sequence to reduce downstream re-sorting effort. We formalize this problem as *online multi-agent path finding with order-contiguity (online MAPF-OC)*, where agents (i.e., parcels) appear over time and exit upon delivery. To efficiently solve online MAPF-OC, we propose *Dual-Ordering Prioritized Planning (DOPP)*, a complete polynomial-time algorithm with a three-level structure that *(i)* searches order-level arrival sequences, *(ii)* refines agent-level priorities, and *(iii)* synthesizes feasible solutions via prioritized planning. Experiments on various conveyor-network layouts, including those derived from actual warehouses, demonstrate DOPP's practical scalability and ability to generate high-quality plans within tight time budgets.

1 Introduction

In modern fulfillment centers, parcels retrieved at the outfeed of an automated storage system are injected into the conveyor network and delivered to workstations. Figure 1 illustrates a typical setup. While conveyors are a standard component of intra-facility transport, networks with many merges and splits are prone to congestion and blocking. Thus, even if the upstream storage system releases parcels quickly and the terminal workstations process parcels sufficiently fast, overall throughput can still be constrained by delays within the intermediate conveyor network. Therefore, efficient outfeed-to-workstation routing is crucial for system performance.

Such a scenario can be naturally abstracted as *multi-agent path finding (MAPF)* [Stern *et al.*, 2019], which has been widely studied in warehouse automation to synthesize the collision-free motion of many robots performing upstream retrieval. However, parcel routing on conveyor networks, which often form the downstream transport layer, has received comparatively less attention. We model this transport as *online MAPF* [Švancara *et al.*, 2019], where agents (i.e., parcels) are released over time at the storage outfeed, traverse the directed conveyor network under collision avoidance, are processed at workstations by human/robotic operators, and then exit.

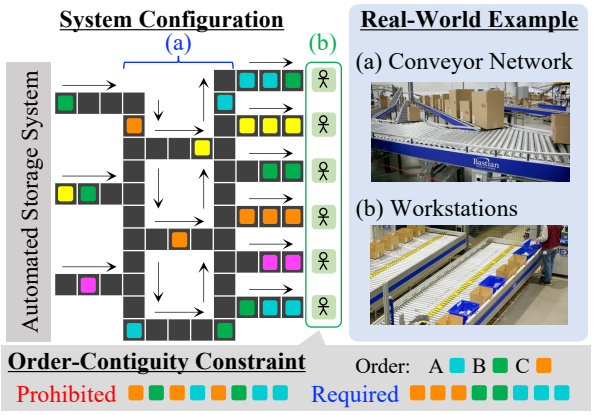

Figure 1: Conveyor-based fulfillment system in warehouse logistics. Parcels retrieved at the outfeed of an automated storage system (e.g., grid-based robotic storage systems [Salzman and Stern, 2020; Chen *et al.*, 2021]) enter a directed one-way conveyor network and are routed online to workstations under collision avoidance. The *order-contiguity* constraint requires parcels of the same order to arrive contiguously (i.e., without interleaving by other orders) at each workstation. Photos sourced from [Bastian Solutions, 2025].

Beyond collision avoidance captured by online MAPF, practical fulfillment operations impose an additional requirement: parcels must be grouped by *order* and handled together at the same workstation. If parcels of different orders interleave in the arrival sequence, operators must re-sort them, increasing workload and processing delay. This poses a novel *order-contiguity* constraint: at each workstation, parcels with the same order must arrive consecutively, without interleaving with other orders, thereby forming a consecutive block.

To this end, we formalize *online MAPF with order-contiguity (online MAPF-OC)*. To solve this, we propose a search-based anytime algorithm, called *Dual-Ordering Prioritized Planning (DOPP)*, consisting of a three-level optimization process: *(i)* it first searches over order-level priorities

to determine the arrival sequence of orders at workstations; *(ii)* it refines priorities within orders at the agent level; and *(iii)* given these priorities, it synthesizes collision-free and order-contiguous paths via prioritized planning (PP) [Erdmann and Lozano-Pérez, 1987; Silver, 2005]. DOPP is complete and runs in polynomial time; it quickly constructs a feasible solution and then iteratively improves the makespan over time. We evaluate DOPP on various conveyor-network maps, including realistic layouts derived from operational warehouses, demonstrating its practical scalability within tight time budgets and strong makespan performance relative to an ad-hoc reactive policy.

2 Related Work

Parcel transport. Routing and control for baggage/parcel transport systems have been widely studied, including approaches for online route choice and flow coordination in track-like networks [Tarau *et al.*, 2010; Zeinaly *et al.*, 2015]. These approaches typically focus on coarse routing decisions, while junction-level coordination relies on local rule-based checks [Klotz *et al.*, 2013], thereby limiting opportunities for globally efficient scheduling. For conveyor systems, often targeting specific conveyor layouts, several optimization-based schemes have been studied [Ago *et al.*, 2007; Novak *et al.*, 2023], which reduce routing to structured problems solvable by off-the-shelf solvers, such as MILP or network-flow formulations. However, this direction introduces a large number of variables and is computationally prohibitive, making it unsuitable for online planning at scale. This motivates our search-based MAPF approach for online synthesis of collision-free parcel trajectories on general directed graphs.

Multi-agent path finding (MAPF) is a generic abstraction for coordinating multiple entities on a shared graph under collision avoidance with prominent applications in warehouse automation [Stern *et al.*, 2019]. To date, various MAPF solvers have been proposed, ranging from computationally heavy but optimal approaches [Sharon *et al.*, 2015] to scalable but suboptimal approaches [Okumura, 2023b]. To capture persistent warehouse-like operations, *lifelong MAPF* has been proposed, where each agent receives a new goal upon reaching its current one [Ma *et al.*, 2017]. *Online MAPF* models a stream of agents that appear over time, while previously revealed agents may already be executing their plans [Švancara *et al.*, 2019]. Our conveyor routing problem inherits this lifelong/online nature but additionally introduces an operational requirement on the arrival sequence at each destination. As another closely related variant, *MAPF-PC (with precedence constraints)* [Zhang *et al.*, 2022] assumes that each agent is given a sequence of goals together with precedence constraints between goal-completion events. In contrast, our focus, MAPF-OC, imposes an order-level non-interleaving constraint at each destination: parcels sharing an order must arrive as a contiguous block, while the order of these blocks is a decision variable optimized for the makespan objective. Beyond warehouse robotics, MAPF-based techniques have been applied to broad domains, such as rail networks [Li *et al.*, 2021a] and autonomous intersection coordination [Li *et al.*, 2023; Yan *et al.*, 2024]; we similarly extend the application scope of MAPF to conveyor systems.

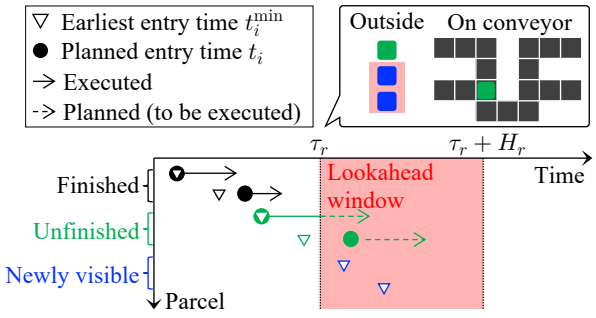

Figure 2: Online replanning at τ_r. Executed trajectories up to τ_r are fixed and form the initial state of the snapshot. Visible agents are classified as *finished* (already disappeared; black), *unfinished* (still on the conveyor or not entered yet; green), and *newly visible* (revealed within the lookahead H_r; blue). The snapshot at τ_r plans trajectories for unfinished and newly visible agents from τ_r onward.

3 Problem Definition

This section formalizes MAPF-OC. We present the system model and constraints, and define the offline/online problems.

3.1 System Model

Conveyor network. We model the conveyor network as a *directed* graph $G = (V, E)$. Each vertex $v \in V$ represents a unit-capacity conveyor segment, and each directed edge $(u, v) \in E$ represents one-way transport. We distinguish *entry* vertices $\mathcal{S} \subset V$, where parcels are injected into the conveyor network, and *destination* vertices $\mathcal{G} \subset V$, corresponding to workstations in the physical system (leftmost and rightmost cells in Fig. 1, respectively).

Agents and orders. Each parcel corresponds to an agent $i \in \mathcal{A}$ that moves on G in discrete time $t \in \mathbb{N}_{\geq 0}$ and is associated with a tuple $\langle o_i, s_i, g_i, t_i^{\min} \rangle$, where $o_i \in \mathcal{O}$ is an order label, $s_i \in \mathcal{S}$ is an entry vertex, and $g_i \in \mathcal{G}$ is its destination. We assume a fixed order-to-destination assignment $d : \mathcal{O} \to \mathcal{G}$ and set $g_i = d(o_i)$, and require that g_i is reachable from s_i in G. Each agent has an *earliest entry time* $t_i^{\min}$. The actual entry time t_i is a decision variable and can be any $t_i \geq t_i^{\min}$. Upon reaching its destination, an agent must stay there for a *service time* of k steps and then leaves the network. This models the handling time at the workstation (e.g., unloading/packing). We assume a uniform k for simplicity.

3.2 Constraints

Time-dependent path for an agent $i \in \mathcal{A}$ is a function $\pi_i : \mathbb{N}_{\geq 0} \to V \cup \{\bot\}$, where $\pi_i(t)$ denotes the location at time t and $\bot$ represents "outside the network." We define the *destination arrival time* $T_i := \min\{t \mid \pi_i(t) = g_i\}$. A path π_i is *valid* if there exists an entry time $t_i \geq t_i^{\min}$ such that

$$\pi_i(t) = \bot \ (t < t_i), \quad \pi_i(t_i) = s_i,$$
$$\pi_i(t+1) \in \{\pi_i(t)\} \cup \{v \mid (\pi_i(t), v) \in E\} (t_i < t < T_i),$$
$$\pi_i(t) = g_i \ (T_i \leq t \leq T_i + k), \quad \pi_i(t) = \bot \ (t > T_i + k).$$

Visible Agents $\mathcal{A}^r$

i	o_i	s_i	g_i	$t_i^{\min}$
1		$u^{\uparrow}$	$v^{\downarrow}$	20
2		$u^{\uparrow}$	$v^{\downarrow}$	28
3		$u^{\downarrow}$	$v^{\downarrow}$	25
4		$u^{\downarrow}$	$v^{\downarrow}$	29
5		$u^{\downarrow}$	$v^{\uparrow}$	31
6		$u^{\uparrow}$	$v^{\uparrow}$	34

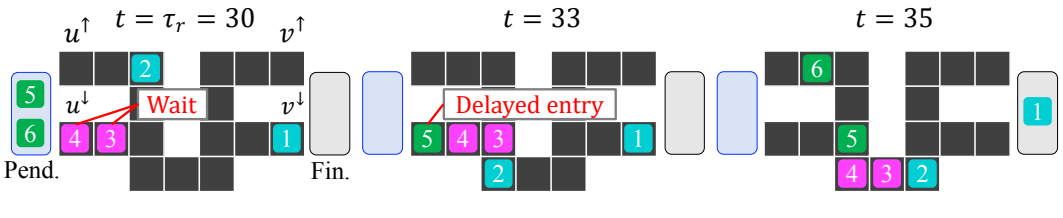

Figure 3: Example of snapshot with $\tau_r = 30$, $H_r = 5$, and $k = 4$. The colors correspond to order labels. Agent 1 arrives at its destination at $t = 30$ and stays in service during $t = 31, \dots, 34$, and disappears at $t = 35$. To preserve order-contiguity at destination $v^{\downarrow}$, agents 3 and 4 wait until agent 2 passes the junction. This waiting blocks the entry cell, so although $t_5^{\min} = 31$, the entry of agent 5 is delayed to $t_5 = 33$. By $t = 35$, the remaining parcels are queued so that future arrivals at destinations form consecutive order blocks (i.e., order-contiguous).

That is, an agent waits or moves to an outgoing neighbor each step, must stay at its destination during service, and then leaves the network. A set of paths $\pi = \{\pi_i \mid i \in \mathcal{A}\}$ is *collision-free* if $|\{i \in \mathcal{A} \mid \pi_i(t) = v\}| \leq 1$ for all $v \in V$ and $t \in \mathbb{N}_{\geq 0}$. Note that *swap collisions* are impossible on a directed one-way network.

Order-contiguity. Given a destination $g \in \mathcal{G}$, let $(i_1, \dots, i_m)$ be the agents with destination g sorted by increasing arrival times, i.e., $T_{i_1} < \cdots < T_{i_m}$. We say π satisfies *order-contiguity* at g if for any $1 \leq p < q < r \leq m$, $o_{i_p} = o_{i_r} \Rightarrow o_{i_p} = o_{i_q}$. Thus, the arrival sequence at each destination can be partitioned into consecutive order blocks. The set of paths π is *order-contiguous* if it satisfies the above for all $g \in \mathcal{G}$.

3.3 Offline MAPF with Order-Contiguity

The offline setting corresponds to the *one-shot* MAPF problem: an instance specifies G and a finite set of agents $\mathcal{A}$. A set of paths $\pi = \{\pi_i \mid i \in \mathcal{A}\}$ is a *feasible solution* if it is valid, collision-free, and order-contiguous. We define the *makespan* of π as $\max_{i \in \mathcal{A}} (T_i + k)$.

3.4 Online MAPF with Order-Contiguity

Batch-based replanning. In practice, conveyor systems operate continuously for extended periods (e.g., several hours to a full day). Planning the entire operation in one-shot is impractical, both because the problem scale becomes enormous and because future order information may be incomplete or change over time. We therefore replan in *batches* over rounds $r \in \{1, \dots, r_{\text{end}}\}$. Let τ_r be the replanning timestep of round r, with $\tau_1 := 0$ and $\tau_{r+1} > \tau_r$. We define the lookahead as $H_r := \tau_{r+1} - \tau_r$. At replanning time τ_r, we call agents whose earliest entry time is before $\tau_r + H_r$, *visible*, defined as:

$$\mathcal{A}^r := \{i \in \mathcal{A} \mid t_i^{\min} < \tau_r + H_r\}. \tag{1}$$

We assume that all agents of the same order are revealed within a single replanning window (*single-window reveal*). This assumption is not merely for modeling simplicity: without sufficient order information, the order-contiguity constraint can become ill-posed online, since newly revealed parcels of an already-processed order may force unavoidable violations of contiguity. Formally, for each $o \in \mathcal{O}$, there exists an index r_o such that

$$\tau_{r_o} \leq t_i^{\min} < \tau_{r_o+1} \qquad \forall i \in \{j \in \mathcal{A} \mid o_j = o\}. \tag{2}$$

Snapshot problem. The *initial state* of the snapshot at replanning time τ_r is given by the executed trajectories until τ_r, as depicted in Fig. 2. In addition, newly visible agents in the current lookahead window are given. Agents that have already finished service and disappeared by τ_r remain known, but are excluded from replanning. At replanning time τ_r, a *snapshot plan* π^r is a feasible solution if *(i)* $\pi_i^r(t) = \pi_i^{r-1}(t)$ for all agents $i \in \mathcal{A}^{r-1}$ and all $t \leq \tau_r$, and *(ii)* the resulting trajectories for agents in $\mathcal{A}^r$ are valid, collision-free, and order-contiguous over the entire time horizon. Let T_i^r denote the arrival time of agent i under π_i^r, and define the *snapshot makespan* as $\max_{i \in \mathcal{A}^r} (T_i^r + k)$. A feasible solution is *snapshot-optimal* if it minimizes this makespan. Figure 3 illustrates examples of a snapshot instance and a feasible solution.

Lifelong objective. In this *online* setting, agents keep becoming visible over replanning steps, and we assume the arrival stream eventually terminates. We aim to minimize the completion time of the last agent, although no online algorithm can guarantee to achieve this global optimum in general [Švancara *et al.*, 2019]. We therefore optimize the snapshot makespan at each replanning time as a practical surrogate for this objective.

Computational complexity. Finding a snapshot-optimal solution is NP-hard. This follows from the NP-hardness of makespan-minimizing MAPF shown in [Ma *et al.*, 2016]: their reduction can be adapted to our snapshot setting by considering a single replanning window and assigning each agent to a distinct order. Thus, in this work, we do not aim to compute optimal solutions; rather, our focus is on establishing suboptimal yet scalable real-time planning.

4 Dual-Ordering Prioritized Planning

In MAPF-OC, agents can wait outside the network before entering and leave their destination after a bounded service time. These structural conditions correspond to a *well-formed* setting, under which *prioritized planning (PP)* is guaranteed to find a feasible solution in polynomial time [Čáp *et al.*, 2015]. Motivated by this, we develop a PP-based *anytime* algorithm, termed *Dual-Ordering Prioritized Planning (DOPP)*, which quickly constructs a feasible solution to the snapshot problem defined in Sec. 3.4, then improves the makespan as time permits. Repeated over replanning rounds, DOPP yields a

Algorithm 1 High-level of DOPP (snapshot at τ_r)

Input: Visible agents $\mathcal{A}^r$, previous snapshot solution π^{r-1}

1: Initialize precedence constraints $\prec^r_{\mathcal{O}}, \prec^r_{\mathcal{A}}$ ▷ Section 4.2
2: Topologically sort $\prec^r_{\mathcal{A}}$ to obtain $\mathbf{L}^r_{\mathcal{A}}$ (a total order over $\mathcal{A}^r$)
3: $\pi^r \leftarrow$ run Level 1 with $\mathbf{L}^r_{\mathcal{A}}$ and $\prec^r_{\mathcal{O}}$
4: **while** $\neg$`interrupt()` **do**
5: Construct neighbor constraints $\tilde{\prec}^r_{\mathcal{O}}, \tilde{\prec}^r_{\mathcal{A}}$ ▷ Section 4.4
6: Topologically sort $\tilde{\prec}^r_{\mathcal{A}}$ to obtain $\tilde{\mathbf{L}}^r_{\mathcal{A}}$
7: $\tilde{\pi}^r \leftarrow$ run Level 1 with $\tilde{\mathbf{L}}^r_{\mathcal{A}}$ and $\tilde{\prec}^r_{\mathcal{O}}$
8: **if** $\text{makespan}(\tilde{\pi}^r) < \text{makespan}(\pi^r)$ **then**
9: $\pi^r \leftarrow \tilde{\pi}^r$; $\prec^r_{\mathcal{O}} \leftarrow \tilde{\prec}^r_{\mathcal{O}}$; $\prec^r_{\mathcal{A}} \leftarrow \tilde{\prec}^r_{\mathcal{A}}$
10: **return** π^r ▷ Snapshot solution at τ_r

lifelong execution; the offline setting is the special case of a single round.

We focus on PP for practicality, among the mainstream MAPF paradigms: CBS and its extensions [Sharon *et al.*, 2015; Li *et al.*, 2021b] are ill-suited to real-time planning with hundreds of agents, while PIBT and its successors [Okumura *et al.*, 2022; Okumura, 2023b] are known to perform poorly in narrow corridors [Okumura, 2023a]. Besides, enforcing order-contiguity in PIBT necessitates conservative operation, which reduces efficiency, as we observe empirically in Sec. 5.

In the following, we first provide an overview of DOPP, and then describe initialization for obtaining a feasible plan, followed by theoretical properties and anytime refinement.

4.1 Overview

Structure. DOPP searches priority constraints at high level and runs PP at low level, similar to PBS [Ma *et al.*, 2019] for classical MAPF. To enforce order-contiguity, DOPP maintains precedence constraints at both order level and agent level, yielding a three-level structure: Level 3 searches *order precedence*, Level 2 searches *agent priority* subject to Level 3, and Level 1 applies PP under the resulting priorities. Pseudocode and conceptual illustration are shown in Alg. 1 and Fig. 4, respectively.

Priority constraints. Let $\mathcal{O}^r := \{o_i \mid i \in \mathcal{A}^r\}$ be the *visible orders* at τ_r, where $\mathcal{A}^r$ is the set of visible agents in Eq. (1). DOPP maintains two sets of constraints: an *order-precedence set* $\prec^r_{\mathcal{O}}$ over $\mathcal{O}^r$ and an *agent-priority set* $\prec^r_{\mathcal{A}}$ over $\mathcal{A}^r$. An element $o \prec o'$ of $\prec^r_{\mathcal{O}}$ (or $i \prec j$ of $\prec^r_{\mathcal{A}}$) means that o (resp. i) has higher priority than o' (resp. j).

Replanning policy. In the online setting, while it is possible to replan priorities of all visible agents at each replanning time τ_r, DOPP instead adopts a *committed-suffix* policy to preserve well-formedness. That is, agents that are already on the network at τ_r are given higher priority so that they are cleared from the network first. The relative priorities among these agents are inherited from the previous plan for consistency. As a result, DOPP optimizes priorities for agents/orders that have not entered the network at τ_r yet.

Agents and orders classification. Herein, to illustrate DOPP, we call an agent *active* if it is already on the conveyor at τ_r; otherwise it is *pending*:

$$\mathcal{A}^r_{\text{act}} := \{i \in \mathcal{A}^r \mid \pi^{r-1}_i(\tau_r) \in V\}, \mathcal{A}^r_{\text{pend}} := \mathcal{A}^r \setminus \mathcal{A}^r_{\text{act}}. \quad (3)$$

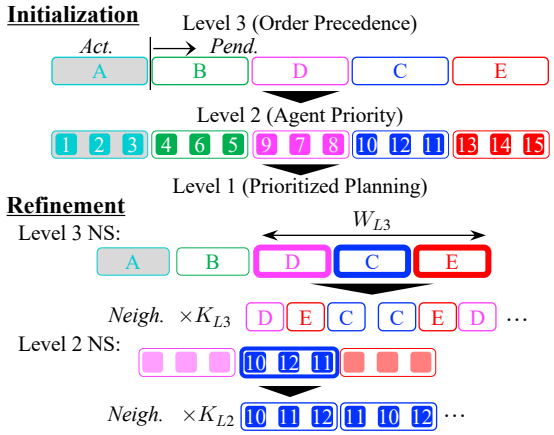

Figure 4: Workflow of DOPP. Initialization constructs an order precedence (Level 3) and an order-consistent agent priority (Level 2), and then computes a feasible snapshot solution by PP (Level 1). Refinement performs neighborhood search: Level 3 perturbs the pending-order subsequence (window permutation), while Level 2 perturbs the agent order within a selected pending order.

Likewise, an order is *active* if it has at least one active agent; otherwise it is *pending*:

$$\mathcal{O}^r_{\text{act}} := \{o \in \mathcal{O}^r \mid \exists i \in \mathcal{A}^r_{\text{act}} : o_i = o\}, \mathcal{O}^r_{\text{pend}} := \mathcal{O}^r \setminus \mathcal{O}^r_{\text{act}}. \quad (4)$$

4.2 Initialization

We detail how to quickly compute an initial solution via PP (lines 1–3 in Alg. 1). As PP conducts agent-wise path planning based on priorities assigned to each agent, we need to obtain a total ordering over agents, while being careful about maintaining the feasibility of PP. The Level 2 and Level 3 procedures construct such an order.

Level 3: order precedence. We initialize the order-precedence constraints $\prec^r_{\mathcal{O}}$ as follows: *(i)* place all active orders before pending ones to avoid breaking ongoing arrival blocks, *(ii)* inherit the relative precedence among active orders from the previous round, and *(iii)* set precedence among pending orders by an earliest-entry heuristic. Formally, let $\prec^{-1}_{\mathcal{O}} := \emptyset$ for $r = 0$. For $r \geq 0$, we define $\prec^r_{\mathcal{O}}$ as the union of the following constraints:

$$\prec^r_{\mathcal{O}} := \{o \prec o' \mid o \in \mathcal{O}^r_{\text{act}}, o' \in \mathcal{O}^r_{\text{pend}}\} \quad (5)$$

$$\cup \{o \prec o' \mid o, o' \in \mathcal{O}^r_{\text{act}}, (o \prec o') \in \prec^{r-1}_{\mathcal{O}}\} \quad (6)$$

$$\cup \{o \prec o' \mid o, o' \in \mathcal{O}^r_{\text{pend}}, \rho_o < \rho_{o'}\}, \quad (7)$$

where $\rho_o := \min\{t^{\min}_i \mid i \in \mathcal{A}^r,\ o_i = o\}$. Ties in ρ_o are broken uniformly at random.

Level 2: agent priority. We initialize the agent-priority constraints $\prec^r_{\mathcal{A}}$ to be consistent with the order precedence and to preserve feasibility across replanning rounds. Intuitively, we *(i)* enforce order-consistency, i.e., agents of higher-precedence orders are planned before those of lower-precedence orders; then, *(ii)* for *active* orders $o \in \mathcal{O}^r_{\text{act}}$, those having at least one agent on the conveyor at τ_r, inherit the pairwise precedence among their agents from the previous

round; and *(iii)* for *pending* orders $o \in \mathcal{O}^r_{\mathrm{pend}}$, those having no agent on the conveyor at τ_r, initialize precedence among their agents with $t_i^{\min}$. Formally, let $\prec_{\mathcal{A}}^{-1} := \emptyset$ for $r = 0$. For $r \geq 0$, define $\prec^r_{\mathcal{A}}$ as the union of the following constraints:

$$\prec^r_{\mathcal{A}} := \{i \prec j \mid i, j \in \mathcal{A}^r, (o_i \prec o_j) \in \prec^r_{\mathcal{O}}\} \quad (8)$$

$$\cup \{i \prec j \mid o_i = o_j, o_i \in \mathcal{O}^r_{\mathrm{act}}, (i \prec j) \in \prec^{r-1}_{\mathcal{A}}\} \quad (9)$$

$$\cup \{i \prec j \mid o_i = o_j, o_i \in \mathcal{O}^r_{\mathrm{pend}}, t_i^{\min} < t_j^{\min}\}. \quad (10)$$

We then apply topological sort to $\prec^r_{\mathcal{A}}$ in order to obtain a total ordering over $\mathcal{A}^r$, denoted by $\mathbf{L}^r_{\mathcal{A}}$, which is used in Level 1. Ties in $t_i^{\min}$ are broken uniformly at random.

Level 1: PP. Level 1 synthesizes a snapshot solution by applying standard PP under $\mathbf{L}^r_{\mathcal{A}}$: agents are planned sequentially using space-time A* [Silver, 2005] while avoiding previously planned paths. We must additionally enforce order-contiguity by applying *destination blocking*, which forbids agent i from occupying its destination g_i before time Γ_i:

$$\Gamma_i := \max\{T_j^r + k + 1 \mid j \in \mathcal{A}^r, g_j = g_i, (o_j \prec o_i) \in \prec^r_{\mathcal{O}}\}.$$

If the set is empty, we set $\Gamma_i := 0$. Since $\mathbf{L}^r_{\mathcal{A}}$ respects $\prec^r_{\mathcal{O}}$, this yields order-contiguous arrivals.

4.3 Theoretical Analysis

DOPP's initialization in Sec. 4.2 ensures that the snapshot instance at each τ_r falls into a well-formed setting: agents may wait outside the network, and destinations are released after bounded service. This reduces the snapshot feasibility of MAPF-OC to feasibility under PP (Level 1). Therefore, running Level 1 once under the initialized priorities yields a feasible snapshot solution in polynomial time. Full proofs are provided in Appendix.

Lemma 1. *For every round r, $\prec^r_{\mathcal{O}}$ and $\prec^r_{\mathcal{A}}$ admit a total ordering over $\mathcal{O}^r$ and $\mathcal{A}^r$, respectively, via topological sorting.*

Lemma 2. *At every round r, the snapshot solution produced by Level 1 is order-contiguous.*

Theorem 1. *At each replanning time τ_r, DOPP constructs a feasible snapshot solution in polynomial time.*

4.4 Refinement

The solution obtained by initialization is feasible but is based on heuristically chosen priorities, leaving room for makespan improvement. In the batch-based replanning assumed in Sec. 3.4, planning for the next batch can proceed while the current batch is being executed, providing additional computation time. DOPP therefore performs an *anytime refinement* (lines 4–9 in Alg. 1) that searches better priority constraints at Level 3 and/or Level 2, while preserving well-formedness. Implementation details are provided in Appendix.

Order-precedence refinement. Level 3 refines $\prec^r_{\mathcal{O}}$ by modifying the constraints in Eq. (7), while keeping the constraints in Eq. (5) and (6) unchanged; this preserves well-formedness by retaining precedence among active orders. We perform *neighborhood search (NS)* over pending orders by selecting a contiguous window of W_{L3} orders and sampling K_{L3} candidate perturbations within it (Fig. 4, bottom). For each candidate, we rebuild $\tilde{\prec}^r_{\mathcal{O}}$ accordingly, re-initialize the corresponding agent-priority constraints, and evaluate it by running Level 1. These candidate evaluations are independent and can be parallelized.

Agent-priority refinement. Level 2 refines $\prec^r_{\mathcal{A}}$ by modifying the constraints in Eq. (10), while keeping the constraints in Eq. (8) and (9) unchanged; this preserves well-formedness by retaining priorities among active agents. We perform NS by selecting a pending order $o \in \mathcal{O}^r_{\mathrm{pend}}$ and sampling K_{L2} alternative orderings of its agents. For each candidate, we rebuild the constraints $\tilde{\prec}^r_{\mathcal{A}}$ accordingly and evaluate it by rerunning Level 1.

Combination strategy. Refinement can be applied at Level 3 only, at Level 2 only, or by combining both. For example, we may *(i)* refine $\prec^r_{\mathcal{O}}$ and, for each candidate, rebuild $\prec^r_{\mathcal{A}}$ using the initialization rule and evaluate it by rerunning Level 1; and/or *(ii)* additionally refine $\prec^r_{\mathcal{A}}$ for each Level 3 candidate, generating multiple candidates before rerunning Level 1. We compare these strategies in Sec. 5.

5 Evaluation

This section evaluates the effectiveness of *DOPP* in terms of snapshot quality, runtime, scalability, anytime behavior, and lifelong performance. All solvers are implemented in C++ and run on a MacBook Pro (Apple M3 Max, 128 GB RAM).

5.1 Setup

Maps. We use directed conveyor maps shown in Fig. 5 (top row), designed to cover both typical warehouse layouts and stress-test settings. `Small-A/B` are simple maps that capture basic topology, and `Medium-A/B` are practical-scale maps derived from existing conveyor systems. To stress-test beyond typical layouts, we also include `Complex` and `Large` maps to evaluate planners in more complex and large-scale environments. Herein, we denote the length of exit-buffer as L_{buf} (e.g., 8 for `Medium-A/B`).

Parcel generation. At each time $t = 0, 1, \ldots$, we generate λ new agents with the earliest entry time $t_i^{\min} = t$. To maintain a comparable workload density across map scales, λ is set to 1 for `Small`/`Medium-A/B`, and 3 for `Complex` and `Large`. For each planning time τ_r, we set the number of agents visible per window $|\mathcal{A}^r|$ to 300 for `Small`/`Medium-A/B`, 500 for `Complex`, and 1,000 for `Large`. In the lifelong scenario, the next planning time τ_{r+1} is set to $\max_{i \in \mathcal{A}^r} t_i^{\min}$. Agents are generated in order blocks: we sample an order size l_o uniformly at random from $\{1, \ldots, L_{\mathrm{buf}}\}$, generate l_o agents for the order, and then switch to the next order. Each agent chooses its entry s_i at random from the map-specific set $\mathcal{S}$. Destinations are assigned per order to maintain a balanced workload across $\mathcal{G}$ by selecting the destination with the current minimum cumulative count and updating the count after assignment. These settings are intended to mimic typical fulfillment operations.

DOPP variants. For snapshot experiments, we evaluate five variants of *DOPP*: *Vanilla* (no refinement, i.e., initial solution), *Level3-NS* (refine only Level 3; rebuild Level 2 as in initialization for each Level 3 candidate), *Level2-NS* (refine only Level 2; keep Level 3 fixed), *Sequential NS* (first refine

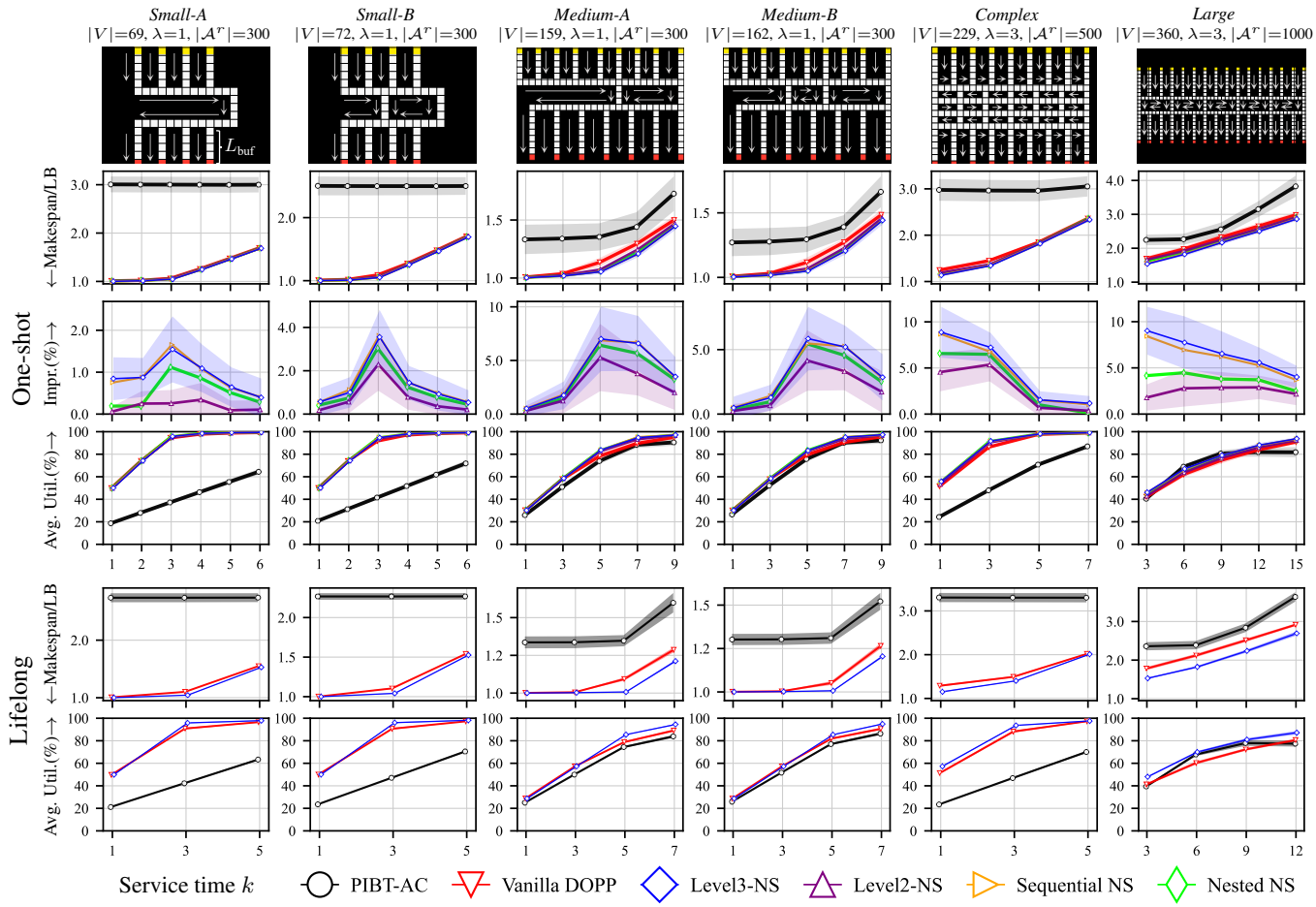

Figure 5: Results for one-shot and lifelong experiments. Headers give the map name, graph size $|V|$, arrival rate λ, and the number of visible agents per window $|\mathcal{A}^r|$, along with the layout (yellow: entries $\mathcal{S}$; red: destinations $\mathcal{G}$). *Upper* rows shows the results for the one-shot problem: makespan M/M_{LB}, relative makespan improvement over *Vanilla DOPP* by anytime refinement, Impr. (%) $:= 100\cdot(M_{\mathrm{Vanilla}}-M)/M_{\mathrm{Vanilla}}$, and the utilization $\bar{U}$. *Lower* rows show those for the lifelong problem: makespan and utilization over 10 consecutive online windows. Each curve shows the mean over 20 instances. Shaded areas indicate the standard deviation for representative methods.

Level 3, then refine Level 2 for the best Level 3 candidate), and *Nested NS* (for each Level 3 candidate, refine Level 2 to generate multiple candidates). For neighborhood search, we use $W_{\mathrm{L3}} = 7$ and $K_{\mathrm{L3}} = K_{\mathrm{L2}} = 10$. These hyperparameters were set via a pilot study; see Appendix. We use a 60 s refinement budget per snapshot instance: *Level2/3-NS* run for 60 s, *Sequential NS* runs Level 3 for 40 s then Level 2 for 20 s, and *Nested NS* uses 60 s total for Level 3 with a 1 s cap for each inner Level 2 call. Level 3 refinement is parallelized with up to 16 threads.

Baseline. We also evaluate the baseline *PIBT-AC*, which runs PIBT [Okumura *et al.*, 2022]—a representative MAPF algorithm that prevents collisions locally—and enforces order-contiguity via *admission control (AC)*. For each destination g, while an order to g is active, agents of other orders destined for g are not admitted into the network, and the lock is released when the remaining agents of the active order are within L_{buf} steps of g. This provides an intuitive and scalable rule-based solution to MAPF-OC. Note that adapting other MAPF solvers, such as CBS variants, is non-trivial due to the order-contiguity constraint. Furthermore, these methods are not well suited for real-time planning on large-scale problems, which led to the current experimental design.

Metrics. We use the *normalized makespan* M/M_{LB}, where $M = \max_i(T_i + k)$, $M_{\mathrm{LB}} = \max_i\big(t_i^{\min} + \mathrm{dist}(s_i, g_i) + k\big)$, and dist is the shortest travel time on G; hence M_{LB} serves as the makespan lower bound. We also report *average destination utilization* $\bar{U}$ (%) to diagnose whether performance is limited by destination capacity. When $\bar{U}$ is close to 100%, improving makespan requires increasing destination capacity (e.g., adding workstations) rather than better routing. Concretely, for each destination g, let $x_g(t) = 1$ iff $\exists i : \pi_i(t) = g$ (otherwise 0), and denote $t_a(g) = \min\{t \mid x_g(t) = 1\}$ and $t_b(g) = \max\{t \mid x_g(t) = 1\}$. The utilization is $U(g) := 100 \cdot \big(\sum_{t=t_a(g)}^{t_b(g)} x_g(t)\big)/(t_b(g) - t_a(g) + 1)$, and $\bar{U}$ is the average over destinations.

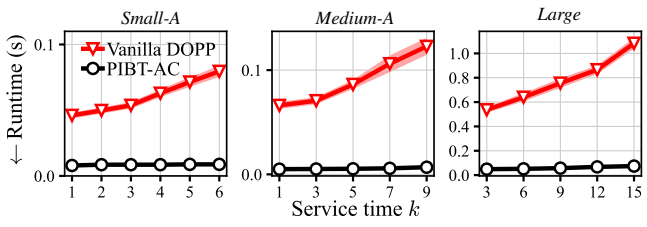

Figure 6: Runtime at representative snapshot settings in Fig. 5. Each point shows the mean over 20 instances.

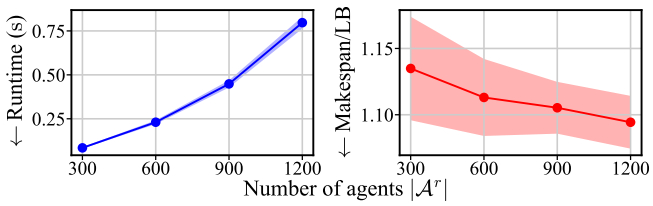

Figure 7: DOPP's scalability on `Medium-A`. Runtime and makespan against the number of visible agents $|\mathcal{A}^r|$ are shown. Service time k is set to five. Each point shows the mean over 20 instances.

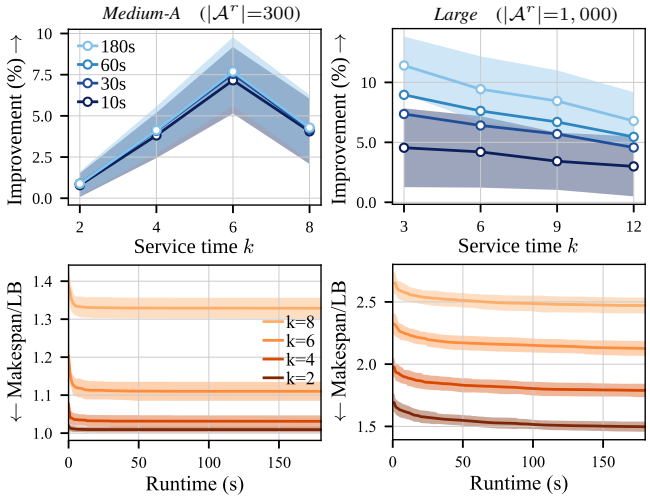

Figure 8: Anytime refinement on `Medium-A` (left) and `Large` (right), over 20 instances. *Top*: relative makespan improvement of *Level3-NS* over *Vanilla DOPP* after a planning budget 10/30/60/180 s. *Bottom*: best-so-far makespan versus runtime under the 180 s budget across the various service time k.

5.2 Results

We first solve the snapshot offline problem (Sec. 3.3) to analyze the behavior of *DOPP variants*, and then evaluate the representative ones on the lifelong setting (Sec. 3.4).

Snapshot quality. Figure 5 (upper) summarizes makespan, improvement over *Vanilla DOPP*, and destination utilization across six maps while sweeping the service time k. Across all maps and k, *DOPP* consistently outperforms *PIBT-AC*, and even *Vanilla* achieves strong makespans. The advantage over *PIBT-AC* is striking on `Small-A/B`, where admission control induces long waits relative to the map size, whereas *DOPP* remains close to the lower bound for $k \leq 3$. As k increases, the utilization approaches saturation, and the makespan degrades for all methods, suggesting that performance becomes dominated by workstation capacity rather than routing decisions. The benefit of refinement is most visible at intermediate k values where both routing conflicts and destination bottlenecks matter; in these cases, *Level3-NS* yields the best results, followed by *Sequential NS*. This highlights that order precedence (Level 3) is a primary driver of performance. The same trend appears on `Medium-A/B`, where the gap among *DOPP* variants is particularly pronounced, and it persists on the `Complex` and `Large` maps, demonstrating robustness to complex layouts and larger instances.

Runtime and scalability. Figure 6 reports runtime at representative snapshot cases sampled from the one-shot results in Fig. 5. *PIBT-AC* is extremely lightweight and runs in 50 ms even on `Large` ($|\mathcal{A}^r| = 1{,}000$) with k=3, while *Vanilla DOPP* remains practical (≈0.53 s) despite achieving much better solution quality. Runtime increases mildly as k grows due to longer makespans and deeper space-time searches, but remains below 0.1 s on `Small/Medium-A` and below 1 s even on `Large`. Figure 7 stresses *Vanilla* on `Medium-A` by increasing the number of visible agents. Runtime grows with $|\mathcal{A}^r|$ but stays below 1 s even with $|\mathcal{A}^r| = 1{,}200$. This scalability performance covers practically relevant horizons for conveyor control, since far-future parcels are less certain in online operation. The normalized makespan improves with larger $|\mathcal{A}^r|$, because the denominator M_{LB} increases with later release times while M stays close to M_{LB}.

Anytime behavior. We analyze how solution quality improves with additional computation time. In Fig. 8, the top row quantifies the benefit of refinement by plotting the relative makespan improvement of *Level3-NS* over *Vanilla DOPP* under different time budgets. `Medium-A` largely saturates by 10 s (≈7.5% improvement at k=6), whereas `Large` continues to improve as time permits (e.g., exceeding 10% improvement at k=3) due to its instance scale. The bottom row plots best-so-far makespan versus time, showing that most gains are obtained early on `Medium-A`. On `Large`, rapid initial improvements are followed by slower but persistent progress up to the time limit.

Lifelong performance. Finally, we evaluate the online setting by solving 10 consecutive windows[1]. Figure 5 (lower) shows that the snapshot improvements translate to lifelong replanning—*DOPP* continues to produce effective replans, resulting in consistently lower makespan than *PIBT-AC*. Refinement via *Level3-NS* remains beneficial in the lifelong setting; for example, on `Medium-A` with k=5, *Vanilla* finds a feasible plan in 0.3 s per window on average, and *Level3-NS* removes about 92% of the excess over the lower bound (from 1.093 to 1.007). *Vanilla* is slightly slower than in one-shot runs (Fig. 6) because agents carry over across windows, but the overhead is modest since PP dominates its runtime and

[1] Pilot experiments with longer scenarios showed similar qualitative trends.

scales linearly with the number of planned agents. As an implementation note, replanning intervals $\tau_{r+1} - \tau_r$ span hundreds of timesteps across maps in our setup, so the $60\,\mathrm{s}$ refinement budget fits within each replanning cycle. These results indicate that the gains observed in snapshot planning carry over to settings where decisions must be made repeatedly online, with the system state evolving from previous decisions.

6 Conclusion

We formulated online MAPF-OC on conveyor networks and presented *DOPP*, an anytime algorithm that quickly finds feasible solutions and improves makespan as time permits. Experiments demonstrated practical scalability and strong solution quality. This work enriches an application example of foundational MAPF algorithms beyond typical warehouse sortation. Future work includes extending online MAPF-OC to incorporate the parcel entry positions and destination assignment for more advanced logistics automation.

References

[Ago *et al.*, 2007] Masatoshi Ago, Tatsushi Nishi, and Masami Konishi. Simultaneous optimization of storage allocation and routing problems for belt-conveyor transportation. *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, 2007.

[Bastian Solutions, 2025] Bastian Solutions. Sawtooth merge / goods-to-person workstation. `https://www.bastiansolutions.com/solutions/technology/sortation/merges-combiners/sawtooth/`, `https://www.bastiansolutions.com/solutions/technology/conveyor-systems/bastian-solutions-conveyor/goods-to-person-workstation/`, 2025. Accessed: 2025-12-24.

[Chen *et al.*, 2021] Zhe Chen, Javier Alonso-Mora, Xiaoshan Bai, Daniel D. Harabor, and Peter J. Stuckey. Integrated task assignment and path planning for capacitated multi-agent pickup and delivery. *IEEE Robotics and Automation Letters (RA-L)*, 2021.

[Erdmann and Lozano-Pérez, 1987] Michael Erdmann and Tomás Lozano-Pérez. On multiple moving objects. *Algorithmica*, 1987.

[Klotz *et al.*, 2013] Thomas Klotz, Jens Schönherr, Norman Seßler, Bernd Straube, and Karsten Turek. Automated formal verification of routing in material handling systems. *IEEE Transactions on Automation Science and Engineering (T-ASE)*, 2013.

[Li *et al.*, 2021a] Jiaoyang Li, Zhe Chen, Yi Zheng, Shao-Hung Chan, Daniel Harabor, Peter J. Stuckey, Hang Ma, and Sven Koenig. Scalable rail planning and replanning: Winning the 2020 flatland challenge. In *Proc. Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 2021.

[Li *et al.*, 2021b] Jiaoyang Li, Wheeler Ruml, and Sven Koenig. Eecbs: Bounded-suboptimal search for multi-agent path finding. In *Proc. AAAI Conf. on Artificial Intelligence (AAAI)*, 2021.

[Li *et al.*, 2023] Jiaoyang Li, The Anh Hoang, Eugene Lin, Hai L. Vu, and Sven Koenig. Intersection coordination with priority-based search for autonomous vehicles. In *Proc. AAAI Conf. on Artificial Intelligence (AAAI)*, 2023.

[Ma *et al.*, 2016] Hang Ma, Craig Tovey, Guni Sharon, T. K. Satish Kumar, and Sven Koenig. Multi-agent path finding with payload transfers and the package-exchange robot-routing problem. In *Proc. AAAI Conf. on Artificial Intelligence (AAAI)*, 2016.

[Ma *et al.*, 2017] Hang Ma, Jiaoyang Li, T.K. Satish Kumar, and Sven Koenig. Lifelong multi-agent path finding for online pickup and delivery tasks. In *Proc. Int. Conf. on Autonomous Agents & Multiagent Systems (AAMAS)*, 2017.

[Ma *et al.*, 2019] Hang Ma, Daniel Harabor, Peter J. Stuckey, Jiaoyang Li, and Sven Koenig. Searching with consistent prioritization for multi-agent path finding. In *Proc. AAAI Conf. on Artificial Intelligence (AAAI)*, 2019.

[Novak *et al.*, 2023] Antonin Novak, Matous Pikokus, and Zdenek Hanzalek. Optimization of circular conveyor belt systems with multi-commodity network flows. In *Proc. Int. Conf. on Operations Research and Enterprise Systems (ICORES)*, 2023.

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

[Okumura, 2023a] Keisuke Okumura. Improving lacam for scalable eventually optimal multi-agent pathfinding. In *Proc. Int. J. Conf. on Artificial Intelligence (IJCAI)*, 2023.

[Okumura, 2023b] Keisuke Okumura. Lacam: Search-based algorithm for quick multi-agent pathfinding. In *Proc. AAAI Conf. on Artificial Intelligence (AAAI)*, 2023.

[Salzman and Stern, 2020] Oren Salzman and Roni Stern. Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems. In *Proc. Int. Conf. on Autonomous Agents & Multiagent Systems (AAMAS)*, 2020.

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

[Silver, 2005] David Silver. Cooperative pathfinding. In *Proc. AAAI Conf. on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2005.

[Stern *et al.*, 2019] Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Kumar, Roman Barták, and Eli Boyarski. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proc. Ann. Symp. on Combinatorial Search (SoCS)*, 2019.

[Tarau *et al.*, 2010] Alina N. Tarau, Bart De Schutter, and Hans Hellendoorn. Model-based control for route choice in automated baggage handling systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 2010.

[Čáp *et al.*, 2015] Michal Čáp, Peter Novák, Alexander Kleiner, and Martin Selecký. Prioritized planning algorithms for trajectory coordination of multiple mobile robots. *IEEE Transactions on Automation Science and Engineering (T-ASE)*, 2015.

[Švancara *et al.*, 2019] Jiří Švancara, Marek Vlk, Roni Stern, Dor Atzmon, and Roman Barták. Online multi-agent pathfinding. In *Proc. AAAI Conf. on Artificial Intelligence (AAAI)*, 2019.

[Yan *et al.*, 2024] Zhongxia Yan, Han Zheng, and Cathy Wu. Multi-agent path finding for cooperative autonomous driving. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2024.

[Zeinaly *et al.*, 2015] Yashar Zeinaly, Bart De Schutter, and Hans Hellendoorn. An integrated model predictive scheme for baggage-handling systems: Routing, line balancing, and empty-cart management. *IEEE Transactions on Control Systems Technology*, 2015.

[Zhang *et al.*, 2022] Han Zhang, Jingkai Chen, Jiaoyang Li, Brian C. Williams, and Sven Koenig. Multi-agent path finding for precedence-constrained goal sequences. In *Proc. Int. Conf. on Autonomous Agents & Multiagent Systems (AAMAS)*, 2022.

Appendix

A Proofs

We provide omitted proofs for lemmas and theorems in Sec. 4.3.

Lemma 1. *For every round r, $\prec_{\mathcal{O}}^r$ and $\prec_{\mathcal{A}}^r$ admit a total ordering over $\mathcal{O}^r$ and $\mathcal{A}^r$, respectively, via topological sorting.*

Proof. We prove the claim for $\prec_{\mathcal{O}}^r$ and $\prec_{\mathcal{A}}^r$ separately.
Order precedence $\prec_{\mathcal{O}}^r$. We argue by induction on r. At $r = 0$, visible orders are totally ordered by the key ρ_o with tie-breaking. Assume $\prec_{\mathcal{O}}^{r-1}$ induces a total ordering on the visible orders of round $r-1$. In round r, DOPP places every active order before every pending order (Eq. (5)), inherits the relative precedence among active orders from $\prec_{\mathcal{O}}^{r-1}$ (Eq. (6)), and totally orders pending orders by sorting ρ_o with tie-breaking (Eq. (7)). Thus any two distinct orders in $\mathcal{O}^r$ are comparable, and $\prec_{\mathcal{O}}^r$ admits a total ordering via topological sorting.
Agent priority $\prec_{\mathcal{A}}^r$. We argue by induction on r. We show that $\prec_{\mathcal{A}}^r$ admits a total ordering over $\mathcal{A}^r$ by establishing that any two distinct agents $i, j \in \mathcal{A}^r$ are comparable. If $o_i \neq o_j$, then $\prec_{\mathcal{O}}^r$ totally orders o_i and o_j, and Eq. (8) transfers this order to i and j; hence i and j are comparable in $\prec_{\mathcal{A}}^r$. Now consider the case $o_i = o_j = o$. If $o \in \mathcal{O}_{\text{act}}^r$, then by the *single-window reveal* assumption (Eq. (2)), all agents of order o were already visible in round $r-1$ and hence were included in $\prec_{\mathcal{A}}^{r-1}$. By the induction hypothesis, $\prec_{\mathcal{A}}^{r-1}$ admits a total ordering over these agents, and Eq. (9) preserves their pairwise precedence in round r; hence i and j are comparable. If $o \in \mathcal{O}_{\text{pend}}^r$, then all agents of o are pending and are ordered by $t^{\min}$ with tie-breaking (Eq. (10)), hence i and j are comparable. Hence every pair of distinct agents in $\mathcal{A}^r$ is comparable, and $\prec_{\mathcal{A}}^r$ admits a total ordering via topological sorting. □

Lemma 2. *At every round r, the snapshot solution produced by Level 1 is order-contiguous.*

Proof. We show that destination blocking in Level 1, introduced in Sec. 4.2, enforces order-contiguity. Fix a destination $g \in \mathcal{G}$ and consider the agents destined to g sorted by increasing arrival times $T_{i_1}^r < \cdots < T_{i_m}^r$. Suppose for contradiction that the arrival sequence at g is not order-contiguous. Then there exist indices $1 \le p < q < r \le m$ such that $o_{i_p} = o_{i_r}$ and $o_{i_p} \neq o_{i_q}$.

Let $o := o_{i_r} (= o_{i_p})$ and $o' := o_{i_q}$. By Lemma 1, either $o \prec o'$ or $o' \prec o$ holds. If $o \prec o'$, then for agent i_q we have

$$\Gamma_{i_q} \;\ge\; T_{i_r}^r + k + 1,$$

because i_r is an agent of the higher-precedence order o assigned to the same destination g. Hence $T_{i_q}^r \ge \Gamma_{i_q} > T_{i_r}^r$, contradicting $T_{i_q}^r < T_{i_r}^r$. The case $o' \prec o$ symmetrically contradicts $T_{i_p}^r < T_{i_q}^r$. Therefore no such triple exists and arrivals at g are order-contiguous. □

Theorem 1. *At each replanning time τ_r, DOPP constructs a feasible snapshot solution in polynomial time.*

Proof. We first establish *completeness* (existence of a feasible snapshot solution returned by PP under the constructed priorities), and then bound the computational complexity.

Completeness. We prove by induction on r that Level 1 always plans all agents in $\mathcal{A}^r$ without failure, producing a snapshot-feasible solution. For the case $r = 0$, Lemma 1 ensures that the initialized precedences $\prec_{\mathcal{O}}^0$ and $\prec_{\mathcal{A}}^0$ admit total orderings over $\mathcal{O}^0$ and $\mathcal{A}^0$. Thus PP under the induced total agent order produces a collision-free snapshot plan, and order-contiguity follows from Lemma 2. Hence π^0 is feasible.

Active agents. Consider the active agents $\mathcal{A}_{\text{act}}^r$. By definition in Eq. (3), every agent in $\mathcal{A}_{\text{act}}^r$ was already planned at τ_{r-1}. Level 1 plans all active agents before any pending agent, and preserves their relative priority from round $r-1$ (Eq. (5) and Eq. (9)). By the induction hypothesis, π^{r-1} is snapshot-feasible, so a feasible continuation exists at τ_r for all active agents. Therefore, PP succeeds for all agents in $\mathcal{A}_{\text{act}}^r$.

Pending agents. After planning agents in $\mathcal{A}_{\text{act}}^r$, let $t^\star$ be any time strictly after all already planned agents have disappeared from the network. A finite $t^\star$ exists because every agent that reaches its destination leaves the network after k steps. When planning a pending agent $i \in \mathcal{A}_{\text{pend}}^r$, choose an entry time

$$t_i \ge \max\{t_i^{\min}, t^\star, \Gamma_i\}.$$

Then follow any directed path from s_i to g_i, which exists under the reachability assumption in Sec. 3. Since the network is empty after $t^\star$, this yields a valid collision-free path that also respects destination blocking. Repeating this for all pending agents shows that PP succeeds for all agents in $\mathcal{A}_{\text{pend}}^r$.

Finally, order-contiguity follows from Lemma 2. Thus the resulting snapshot solution is feasible.

Polynomial time. Levels 3 and 2 consist of sorting and topological sorting over $\mathcal{O}^r$ and $\mathcal{A}^r$, thus run in polynomial time. For Level 1, we provide a finite horizon that always contains a feasible solution. We define,

$$C^r_{\mathrm{act}} := \begin{cases} \max_{i \in \mathcal{A}^r_{\mathrm{act}}} \left(T_i^{r-1} + k + 1\right) & \text{if } \mathcal{A}^r_{\mathrm{act}} \neq \emptyset, \\ \tau_r & \text{otherwise,} \end{cases}$$

and

$$t^r_{\max} := \begin{cases} \max_{i \in \mathcal{A}^r_{\mathrm{pend}}} t_i^{\min} & \text{if } \mathcal{A}^r_{\mathrm{pend}} \neq \emptyset, \\ \tau_r & \text{otherwise.} \end{cases}$$

Let $D := |V| - 1$ and define

$$\overline{T}^r := \max\{C^r_{\mathrm{act}},\, t^r_{\max},\, \tau_r\} + |\mathcal{A}^r_{\mathrm{pend}}|\,(D + k + 1).$$

When planning an agent, Level 1 searches on the time-expanded graph truncated at $\overline{T}^r$, which has $O(|V| \cdot \overline{T}^r)$ vertices and $O(|E| \cdot \overline{T}^r)$ edges. Space-time A* search on this graph runs in polynomial time in $|V|, |E|$, and $\overline{T}^r$. Repeating this for all agents in $\mathcal{A}^r$ yields an overall polynomial running time.

□

B Implementation Details of Refinement

We describe the implementation of the neighborhood searches in Level 3 and Level 2 introduced in Sec. 4.4.

Order-precedence refinement (Level 3). Let $\mathbf{L}^r_{\mathcal{O}}$ be a total ordering over $\mathcal{O}^r$ obtained by topologically sorting $\prec^r_{\mathcal{O}}$, and let $\mathbf{L}^r_{\mathcal{O},\mathrm{pend}}$ be the subsequence of $\mathbf{L}^r_{\mathcal{O}}$ containing only pending orders in $\mathcal{O}^r_{\mathrm{pend}}$. We select a contiguous window of W_{L3} orders in $\mathbf{L}^r_{\mathcal{O},\mathrm{pend}}$, and sample K_{L3} perturbed sequences by randomly permuting its elements. For each perturbed sequence $\tilde{\mathbf{L}}^r_{\mathcal{O},\mathrm{pend}} = (o_1, \ldots, o_m)$, we construct candidate precedence constraints among pending orders as the chain $\tilde{\prec}^r_{\mathcal{O},\mathrm{pend}} := \{o_l \prec o_{l+1} \mid l = 1, \ldots, m-1\}$, and form $\tilde{\prec}^r_{\mathcal{O}}$ by replacing Eq. (7) with $\tilde{\prec}^r_{\mathcal{O},\mathrm{pend}}$ while keeping Eq. (5) and (6) unchanged. We then re-initialize the agent-priority constraints under $\tilde{\prec}^r_{\mathcal{O}}$ and evaluate the candidate by running Level 1.

Agent-priority refinement (Level 2). Let $\mathbf{L}^r_{\mathcal{A}}$ be the total ordering over $\mathcal{A}^r$ obtained in initialization by topologically sorting $\prec^r_{\mathcal{A}}$. We select a pending order $o \in \mathcal{O}^r_{\mathrm{pend}}$ and extract the subsequence of its agents from $\mathbf{L}^r_{\mathcal{A}}$. We then sample K_{L2} perturbed orderings of these agents, yielding $\tilde{\mathbf{L}}^r_{\mathcal{A},o} = (i_1, \ldots, i_q)$. For each candidate, we encode it as a chain priority constraint among agents of o, $\tilde{\prec}^r_{\mathcal{A},o} := \{i_l \prec i_{l+1} \mid l = 1, \ldots, q-1\}$, and build $\tilde{\prec}^r_{\mathcal{A}}$ by replacing the corresponding part of Eq. (10) while keeping Eq. (8) and (9) unchanged. Then, we evaluate each candidate by running Level 1.

C Pilot Study for Hyperparameter Selection

Setup. We conducted a pilot study to select the hyperparameters for neighborhood search in Level 2 and Level 3. We used the same protocol as the snapshot evaluations in Sec. 5.

K_{L2}	Medium-A	Large
3	1.076	**1.913**
10	**1.075**	1.922
50	1.104	1.937
100	1.103	1.943
500	1.123	1.935

Table 1: Results of pilot study for Level 2: sensitivity to K_{L2} (average Makespan/LB over 20 instances; best in bold).

		K_{L3}				
	W_{L3}	3	10	50	100	500
Medium-A	3	1.07	1.06	–	–	–
	7	1.06	**1.05**	1.06	1.06	1.06
	15	1.07	1.06	1.07	1.07	1.07
	50	1.13	1.13	1.13	1.13	1.13
Large	3	1.86	1.86	–	–	–
	7	1.83	1.82	1.86	1.86	1.87
	15	1.82	**1.81**	1.83	1.86	1.87
	50	1.95	1.93	1.93	1.94	1.94

Table 2: Results of pilot study for Level 3: sensitivity to $(W_{\mathrm{L3}}, K_{\mathrm{L3}})$ (average Makespan/LB over 20 instances; best in bold, "–" indicates omitted combinations).

For `Medium-A`, we fixed $\lambda = 1$, $|\mathcal{A}^r| = 300$, and $k = 5$; for `Large`, we fixed $\lambda = 3$, $|\mathcal{A}^r| = 1000$, and $k = 6$. For each setting, we solved 20 instances and report the average normalized makespan (Makespan/LB) over snapshots. The k values were chosen because the refinement effect of DOPP is most pronounced in these regimes.

Results. Section C summarizes sensitivity to K_{L2} (Level 2), and Sec. C summarizes sensitivity to $(W_{\mathrm{L3}}, K_{\mathrm{L3}})$ (Level 3). Entries marked "–" are omitted: for $W_{\mathrm{L3}} = 3$, the window admits only $3! = 6$ permutations, so we tested only $K_{\mathrm{L3}} \in \{3, 10\}$ and skipped larger values. These results show that moderate values of K_{L2} and K_{L3} perform best. Larger K_{L2} and K_{L3} spend too much time enumerating permutations for a single window, reducing the number of distinct candidates explored within the given refinement time budget. Based on these results, we set $W_{\mathrm{L3}} = 7$ and $K_{\mathrm{L3}} = K_{\mathrm{L2}} = 10$ for experiments in Sec. 5.