Deadline-Aware Multi-Agent Tour Planning

Taoan Huang[1] , **Vikas Shivashankar**[2] , **Michael Caldara**[2] , **Joseph Durham**[2] ,
Jiaoyang Li[1] , **Bistra Dilkina**[1] , **Sven Koenig**[1]

[1]University of Southern California
[2]Amazon Robotics
taoanhua@usc.edu, {vikshiv, caldaram, josepdur}@amazon.com, {jiaoyanl, dilkina, skoenig}@usc.edu

Abstract

The increasing demand for same-day delivery and the commitment of e-commerce companies to this service raise a number of challenges in logistics. One of these challenges for fulfillment centers is to coordinate hundreds of mobile robots in their automated warehouses efficiently to allow for the retrieval and packing of thousands of ordered items within the promised delivery deadlines. We formulate this challenge as the new problem of deadline-aware multi-agent tour planning, where the objective is to coordinate robots to visit multiple picking stations in congested warehouses to allow as many orders to be packed on time as possible. To solve it, we propose *La***R***ge Neighb***O***rhood* **S***earch for D***E***adline-aware Mul***T***i-Agent* **T***our Pl***A***nning* (ROSETTA). We conduct extensive experiments to evaluate ROSETTA with up to 350 robots in simulated warehouses inspired by KIVA systems. We show that it increases the number of orders completed on time by up to 38% compared to several baseline algorithms and also significantly outperforms them in terms of throughput and station utilization.

1 Introduction

E-commerce companies, such as Amazon and Alibaba, have increasingly leveraged multi-robot systems to automate their warehouse operations [Wurman *et al.*, 2008]. Instead of human operators walking around in storage areas to fetch inventory items, mobile *agents* transport them from the storage area to stations where workers pick the ordered items. However, the recent trend in e-commerce to promise faster deliveries poses a number of technical challenges for these systems. For instance, Amazon focuses on *same-day delivery* with promises to deliver orders in as little as five hours [Alexander, 2020]. The same is true for other e-commerce companies, such as Jingdong [Moon, 2013].

The key challenge arises from having tighter deadlines. Instead of having several hours before a customer order leaves the building, e-commerce companies have only a few minutes to prepare the order, resulting in a smaller margin for error. Therefore, they need to plan and schedule the movement of the agents more carefully to ensure that the order deadlines are met. Another challenge is *resource contention*. A single agent might be needed at multiple stations, e.g., an agent carrying a popular item, such as bananas, is likely needed at several stations that process grocery orders. Optimally sequencing these visits is critical in ensuring that the maximum number of deadlines are met. Figure 1 shows an illustrative example of a warehouse with two picking stations. The agents are represented by the shapes that correspond to the items they carry. On the one hand, each picking station needs a set of items by a given deadline, requiring the relevant agents to visit the station on time. On the other hand, agents, such as the ones represented by the red circle and blue square, need to visit both stations, requiring the system to decide on the order in which the agents visit the stations. Motivated by this application, we formulate and solve a large-scale deadline-aware multi-agent tour planning (DA-MATP) problem for inventory-laden mobile robots (which we call *agents*) in a shared environment, where they need to visit multiple locations by given deadlines.

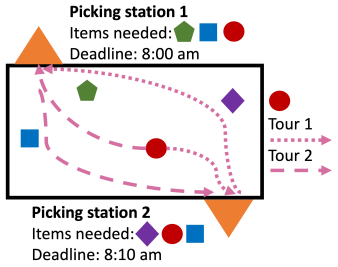

Figure 1: An illustrative example of the planning problem in warehouses for same-day delivery. The agents are represented by the shapes that correspond to the items they carry. Two picking stations are located on the perimeter of the warehouse. Each picking station needs certain items by a given deadline. The red-circle agent can first visit Station 2 and then Station 1 (Tour 1) or first Station 1 and then Station 2 (Tour 2).

DA-MATP is a generalization of multi-agent path finding (MAPF) [Stern *et al.*, 2019; Yu and LaValle, 2013], the popular problem of planning collision-free paths for a team of agents in a shared environment. In the classical MAPF problem, time is discretized into time steps, and, at each time step,

every agent can either move to an adjacent vertex or wait at its current vertex in the graph. The objective is to find a set of paths on the graph, one for each agent, that moves each agent from its given start vertex to its given goal vertex without collisions and minimizes the flowtime or the makespan. The DA-MATP problem can be formulated as a MAPF problem with two main differences from the classical MAPF problem. First, instead of having a unique goal vertex, each agent has a set of goal vertices to visit, each by a given deadline. It is up to the planner to determine the order in which the agent visits these goal vertices. Second, instead of minimizing the flowtime or makespan, the objective is to maximize the number of tasks completed by their deadlines.

Our first contribution is to introduce and formally define the DA-MATP problem. Our second contribution is to propose *La***R***ge Neighb***O***rhood* **S***earch for D***E***adline-aware Mul***T***i-Agent* **T***our Pl***A***nning* (ROSETTA), a novel algorithm for solving the DA-MATP problem. Solving the DA-MATP problem for commercial warehouses requires planning for hundreds of agents over time horizons of 30 to 60 minutes. It takes a classical MAPF solver a few minutes to plan paths at this scale, without reasoning about the goal orderings [Li *et al.*, 2021]. To deal with the added complexity of DA-MATP, we adopt a two-stage planning framework. We first use ROSETTA to construct a goal ordering, or a *tour*, for each agent that minimizes the number of missed deadlines. We then use MAPF algorithms to refine these tours to paths in real-time during execution on a rolling-time-horizon basis. As its name suggests, ROSETTA leverages large neighborhood search (LNS) [Ahuja *et al.*, 2002] to compute high-quality tours by first finding an initial set of tours quickly and then reducing the number of missed deadlines over time. Our third contribution is to develop a novel service time estimator (STE) that allows ROSETTA to construct high-quality tours without any path planning. STE uses simple models of station processing rates and agent motion to predict the queuing delays (i.e., the times spent waiting to be serviced at picking stations) and travel times of agents, respectively. This hierarchical approach allows ROSETTA to compute high-quality tours for tens of stations and hundreds of agents under a minute.

In our experiments, we use simulated warehouses inspired by KIVA systems [Wurman *et al.*, 2008] to evaluate ROSETTA. We compare ROSETTA to several baselines and show that it significantly outperforms them in terms of not only the number of on-time tasks (our main objective) but also other important metrics that reflect service quality but that ROSETTA does not directly optimize, such as the throughput and station utilization.

2 Related Work

Besides the classical MAPF problem [Stern *et al.*, 2019], another closely related problem to DA-MATP is the multi-agent pickup and delivery (MAPD) problem, which is an extension to MAPF with pickup and delivery tasks. A solution to a MAPD instance determines the assignments of tasks to each agent and their orders so as to minimize the flowtime or makespan [Liu *et al.*, 2019; Farinelli *et al.*, 2020]. Multi-goal planning has been studied in the context of lifelong MAPF [Li *et al.*, 2021] and lifelong MAPD [Ma *et al.*, 2017; Ma *et al.*, 2019], where the goal orderings are either not part of the decision space or determined online. Optimal algorithms [Surynek, 2021; Ren *et al.*, 2021] have been proposed for MAPF where the goal orderings are part of the decision space. None of these algorithms solve our problem directly since they minimize the flowtime or makespan and do not scale beyond 35 agents. They also find collision-free paths, whereas ROSETTA plans tours which are refined into paths during execution.

[Ma *et al.*, 2018] study a variant of MAPF with deadlines where they maximize the number of agents that reach their given goal vertices before given deadlines. Deadline requirements have also been considered in MAPD, where the state-of-the-art is a greedy algorithm [Wu *et al.*, 2021] that assigns tasks based on the closest deadlines on the fly during path planning to maximize the number of tasks completed by the deadlines. DA-MATP is different from MAPD with deadlines in that our tasks do not require pickup actions. However, in this paper, we adapt their algorithm ST-SAP, introduced in Section 4, and compare it to ROSETTA.

3 DA-MATP

In this section, we formulate the *deadline-aware multi-agent tour planning (DA-MATP)* problem. Solving it requires finding a tour for each of k agents $\mathcal{A} = \{a_1, \ldots, a_k\}$ to that allows them to cooperatively finish a set of tasks $\mathcal{T}$ on a 2D four-neighbor grid map, where an agent is a mobile robot that carries a large number of identical inventory items.

The grid map is represented as an unweighted directed graph $G = (V, E)$. Each agent a_i has a start vertex $s_i \in V$. A subset of vertices $V_\mathsf{G} \subseteq V$ are goal vertices. Time is discretized into equidistant time steps $[T] = \{1, 2, \ldots, T\}$. An agent can either move to an adjacent vertex or stay at its current vertex from one time step to the next. At any goal vertex, an agent can perform a picking action for δ time steps, i.e., if the agent starts a picking action at time step t, it has to be at the goal vertex from time step t to time step $t + \delta - 1$ so that the human operator can pick up the required inventory items.

Tasks and Deadlines. A task $\tau \in \mathcal{T}$ is specified by a tuple (a_τ, v_τ, d_τ) consisting of an agent $a_\tau \in \mathcal{A}$, a goal vertex $v_\tau \in V_\mathsf{G}$ and a task deadline $d_\tau \in [T]$. We say that task τ is *on-time* if agent a_τ arrives at goal vertex v_τ and starts a picking action at or before time step d_τ, otherwise task τ is late (meaning that it misses the deadline). We say that task τ is completed if agent a_τ arrives at goal vertex v_τ at any time step and perform a picking action at or before time step T, even if it is late. Multiple tasks can be completed by the same agent at the same goal vertex simultaneously with one picking action.

Warehouses typically have a separate *stow* process to top up agents with new inventory. We ignore the stow process in this paper and assume that agents are pre-filled with sufficient inventory in the initial state. We assume that each agent has sufficient capacity to hold all items required by $\mathcal{T}$ since storage containers used in KIVA systems consist of a stack of trays and typically have a large capacity. We also assume that an agent carries exactly one type of items in large numbers

and that it takes a human operator approximately the same amount of time to pick up all required items from an agent at a picking station.

Path Planning and Execution. The movement of the agents in the DA-MATP problem is subject to the following constraints: (1) no more than one agent can be at the same vertex at the same time step, and (2) no agent can travel along an edge if another agent is traveling along the edge in opposite direction (if exists) at the same time step. A *tour* p_i or, syconymously, a goal ordering for agent a_i is a sequence of goal vertices indicating the order of the goal vertices that agent a_i visits. During execution, a set of tours $P = \{p_i : i \in [k]\}$ is passed to a path planner, which generates a set of collision-free paths, one per agent, such that the agents visit the goal vertices in accordance with their tours. We assume that DA-MATP instances are well-formed instances [Čáp *et al.*, 2015; Ma *et al.*, 2017], a class of instances that is important for warehouse logistics. In a well-formed DA-MATP instance, there exists a path between the start vertex of any agent and any goal vertex that traverses no other start and goal vertices. Then, a set of collision-free paths always exists for any given set of tours [Čáp *et al.*, 2015].

Optimization Objective. The objective of DA-MATP is to find a set of tours $\{p_i : i \in [k]\}$ such that the number of on-time tasks is maximized during execution. More concretely, every agent a_i follows a collision-free path π_i. We let $\mathcal{T}_i = \{(a', v', d') \in \mathcal{T} : a' = a_i\}$ be the tasks of agent a_i and $t_{i,v}$ be the time step when agent a_i starts a picking action at goal vertex v when following π_i (w.l.o.g., we assume that each agent performs a picking action at most once at each goal vertex and let $t_{i,v} = \infty$ if agent a_i never performs a picking action at goal vertex v at or before time step T). From them, we can infer a score function $R_i = \sum_{\tau \in \mathcal{T}_i} \mathbb{1}[t_{i,v_\tau} \leq d_\tau]$ for agent a_i which gives a score of 1 for each on-time task. The objective then is to maximize the score across all agents, i.e., $\sum_{a_i \in \mathcal{A}} R_i$.

4 Single-Agent Planning Algorithms

In this section, we discuss two single-agent planning (SAP) algorithms: ST-SAP optimizes for the short term, and LT-SAP optimizes for the long term.

ST-SAP is a myopic algorithm adapted from [Wu *et al.*, 2021] that determines the tour for each agent on the fly by assigning the next goal vertex to the agent every time it finishes a picking action at its current one. To assign the next goal vertex, ST-SAP finds the closest goal vertex that the agent has not yet visited among those with the soonest deadline and breaks ties uniform random. Specifically, suppose that agent a_k finishes the picking action at time step t at goal vertex $v \in V_\mathsf{G}$. For each $v' \in V_\mathsf{G}$ that has not yet been visited by the agent, ST-SAP first finds the task with the soonest deadline $\tilde{t}_{v'}$ whose goal vertex is v' that the agent can complete on-time if it follows the shortest path from v to v' without being blocked by other agents. If no such task exists, $\tilde{t}_{v'}$ is set to ∞. Among the goal vertices with the soonest deadline $V'_\mathsf{G} = \arg\min_{v' \in V_\mathsf{G}} \{\tilde{t}_{v'}\}$, ST-SAP breaks ties in favor of the closest one, i.e., it chooses the next goal vertex from $\arg\min_{v' \in V'_\mathsf{G}} \{\mathsf{dist}(v, v')\}$. Any remaining ties are broken uniformly at random.

ST-SAP runs in polynomial time but is a myopic algorithm that optimizes only for the short term, i.e., it determines the tour for each agent on the fly based on only the tasks with the soonest deadline. We therefore also propose LT-SAP, another SAP algorithm that optimizes for the long term by taking into account all tasks in the future. Specifically, LT-SAP uses local search to plan an individually suboptimal tour for each agent as if it were the only agent moving on the graph. It uses the same local search as ROSETTA, that is introduced in Section 5.2.

5 ROSETTA

To overcome the weaknesses of SAP algorithms for DA-MATP, we propose ROSETTA in this section, which is a multi-agent planning algorithm that takes into account all tasks and optimizes for the long term. ROSETTA is shown in Algorithm 1. ROSETTA takes as input the grid map represented as a directed unweighted graph G, the set of agents $\mathcal{A}$ with start vertices, the set of goal vertices V_G, the time horizon $[T]$ and the set of tasks $\mathcal{T}$. Given this input, ROSETTA first computes an initial solution with prioritized planning (PP) (Line 3). If there is time remaining, it applies adaptive LNS [Ropke and Pisinger, 2006] to improve the solution (Lines 5-15). LNS is a popular local search algorithm that, in each iteration, destroys and re-optimizes a part of the solution. We propose two destroy heuristics to determine which part of the solution to destroy for ROSETTA, namely, a random heuristic and a goal-based heuristic, that generate a subset of agents whose tours will be removed from the solution and then replanned. Adaptive LNS effectively selects one of the destroy heuristics by maintaining a weight vector $\boldsymbol{w}$ for them (Lines 4 and 11). In each iteration of LNS, ROSETTA first selects a destroy heuristic $\mathcal{H}$ (Line 6) and uses $\mathcal{H}$ to generate an agent set $\mathcal{A}'$ (Line 7). It then removes the tours P^- of all agents in $\mathcal{A}'$ (Line 8) and uses PP to replan their tours (Line 10). If the replanned tours P^- improve the solution (Line 12), then ROSETTA replaces P^- with P^+ (Line 13). Finally, it returns the set of tours P (Line 16).

ROSETTA relies on a service time estimator during planning (Lines 2, 3, 7, 9, 10 and 15). To evaluate the score of a given set of tours, ROSETTA could call a MAPF solver to plan the path of each agent w.r.t. its tour. This allows ROSETTA to plan the actions of every agent and calculate the exact score. However, planning such paths for hundreds of agents and for hundreds of time steps on large graphs can take up to a few minutes, which limits the scalability of LNS since it has to plan the paths for a subset of agents in every iteration. We thus propose the novel notion of a service time estimator (STE). The stations are usually the most congested area on a warehouse floor

A STE keeps track of the estimated arrival time and the queuing delay of each agent at each goal vertex to estimate the service time (i.e., the time when the agent finishes the picking action at its last goal vertex) and its score given its tour and the other agents' tours. A STE models the congestion at each goal vertex via a time table. The picking stations are usually

Algorithm 1 ROSETTA

1: **Input:** A DA-MATP instance I (a graph G, agents $\mathcal{A}$ with start vertices, goal vertices V_{G}, a time horizon $[T]$ and tasks $\mathcal{T}$,).
2: Initialize service time estimator STE on G and V_{G}.
3: $P = \{p_i : i \in [k]\} \leftarrow \mathsf{prioritizedPlanning}(I, \mathcal{A}, \mathsf{STE})$
4: Initialize the weight $\boldsymbol{w}$ of the destroy heuristics
5: **while** runtime limit not exceeded **do**
6: $\quad \mathcal{H} \leftarrow \mathsf{selectDestroyHeuristic}(\boldsymbol{w})$
7: $\quad \mathcal{A}' \leftarrow \mathsf{selectAgentSet}(I, \mathcal{H}, \mathsf{STE})$
8: $\quad P^- \leftarrow \{p_i : a_i \in \mathcal{A}'\}$
9: $\quad \mathsf{STE}' \leftarrow \mathsf{STE}$ ▷ Backup the STE
10: $\quad P^+ \leftarrow \mathsf{prioritizedPlanning}(I, \mathcal{A}', \mathsf{STE})$
11: $\quad$ Update the weights $\boldsymbol{w}$ of the destroy heuristics
12: $\quad$ **if** P^+ leads to a higher estimated score than P^- **then**
13: $\quad\quad P \leftarrow (P \setminus P^-) \cup P^+$
14: $\quad$ **else**
15: $\quad\quad \mathsf{STE} \leftarrow \mathsf{STE}'$ ▷ Restore the STE
16: **return** P

Algorithm 2 Estimate the service time and score of a tour

1: **Input:** a STE STE, a picking action duration δ and a tour $p_i = (v_1, \ldots, v_m)$ of agent a_i.
2: Retrieve γ, $\mathcal{A}^{\mathsf{STE}}$ and time tables $\{Q_v : v \in V_{\mathsf{G}}\}$ from STE
3: **if** $a_i \in \mathcal{A}^{\mathsf{STE}}$ **then**
4: $\quad$ **return** the estimated service time and score for agent a_i
5: $v_0 \leftarrow s_i$
6: $\hat{t}_i \leftarrow 0, \hat{R}_i \leftarrow 0$
7: **for** $j \leftarrow 1$ to m **do**
8: $\quad \hat{t}_i \leftarrow \hat{t}_i + \lceil \mathsf{dist}(v_{j-1}, v_j)/\gamma \rceil$
9: $\quad$ Find the minimum $\hat{t}_{i,v_j} \geq \hat{t}_i$ s.t. $[\hat{t}_{i,v_j}, \hat{t}_{i,v_j} + \delta)$ are not occupied in Q_{v_j}
10: $\quad \hat{R}_i \leftarrow \hat{R}_i +$ the number of tasks completed on time at v_j at $\hat{t}_{i,v_j}$ by a_i
11: $\quad \hat{t}_i \leftarrow \hat{t}_{i,v_j} + \delta - 1$
12: Save the estimated service time $\hat{t}_i$ and score $\hat{R}_i$ in STE and return them

Algorithm 3 $\mathsf{prioritizedPlanning}(I, \mathcal{A}^{\mathsf{PP}}, \mathsf{STE})$

1: **Input:** a DA-MATP instance I, a set of agents $\mathcal{A}^{\mathsf{PP}} \subseteq \mathcal{A}$, a STE STE.
2: Retrieve $\mathcal{A}^{\mathsf{STE}}$
3: **for** $a_i \in \mathcal{A}^{\mathsf{PP}} \cap \mathcal{A}^{\mathsf{STE}}$ **do**
4: $\quad$ Remove the tour of a_i from STE
5: Randomly shuffle $\mathcal{A}^{\mathsf{PP}}$ ▷ Obtain a random priority order
6: $P^{\mathsf{PP}} \leftarrow \emptyset$
7: **for** $a_i \in \mathcal{A}^{\mathsf{PP}}$ in descending order of their priorities **do**
8: $\quad p_i \leftarrow \mathsf{findSingleAgentOptimalTour}(I, a_i, \mathsf{STE})$
9: $\quad$ Add tour p_i into STE
10: $\quad P^{\mathsf{PP}} \leftarrow P^{\mathsf{PP}} \cup \{p_i\}$
11: **return** P^{PP}

the most congested area in a warehouse, where we need to worry about the interactions between the agents the most. As for the travel times from one picking station to another, they are more stable and thus we approximate them using the distance divided by a constant travel speed. More importantly, the STE enables multi-agent planning for DA-MATP. In contrast, LT-SAP uses a STE but estimates the travel times of agents without considering the tours of the other agents, and such a STE is not suitable for ROSETTA.

In the following, we first describe the implementation of the STE. We then introduce prioritized planning with a STE (called on Lines 3 and 10 in Algorithm 1) and the two destroy heuristics of the LNS of ROSETTA.

5.1 Service Time Estimator

In this subsection, we describe the implementation of the STE. The STE STE maintains a set of agents $\mathcal{A}^{\mathsf{STE}}$ and the set of already-planned tours $\{p_j : a_j \in \mathcal{A}^{\mathsf{STE}}\}$. It also maintains a time table Q_v for each $v \in V_{\mathsf{G}}$, which is an array of length T. Q_v records the index of the agent that performs a picking action at goal vertex v at each time step (or zero if no such agent exists). Given a tour $p_i = (v_1, \ldots, v_m)$ of an agent a_i, a STE supports three types of operations: (1) estimate the service time and score for a_i given tour p_i and the tours of the agents in $\mathcal{A}^{\mathsf{STE}}$; (2) add the tour of agent a_i into STE; and (3) remove the tour of agent a_i from STE. Given the graph G and goal vertices V_{G}, we initialize a STE (Line 2 in Algorithm 1) by setting $\mathcal{A}^{\mathsf{STE}} = \emptyset$ and Q_v to zeros for all $v \in V_{\mathsf{G}}$. We also pre-compute the all-pair distances between vertices in V_{G} on graph G to speed up the computation later.

Estimating the Service Time and Score for a_i Given Tour p_i and the Tours of the Agents in $\mathcal{A}^{\mathsf{STE}}$. To estimate the service time for a_i, the STE separately estimates (1) the travel time to get from one goal vertex to the next one and (2) the time the agent waits after "arriving at" a goal vertex (it does not precisely arrive at the goal vertex; instead, it waits at some vertices as close to the goal vertex as possible and slowly moves towards it) before it performs the picking action, i.e., the delay due to queuing at the goal vertex. As shown in Algorithm 2, we first retrieve the information from STE (Line 2) and initialize both the estimated service time $\hat{t}_i$ and score $\hat{R}_i$ to 0 (Line 6). We then iterate through all goal vertices v_j in order of their visits according to tour p_i (Line 7). We then estimate the "arrival" time $\hat{t}_i$ of agent a_i at v_j based on the pre-computed distance $\mathsf{dist}(v_{j-1}, v_j)$ from v_{j-1} to v_j and a constant speed $0 < \gamma \leq 1$ (Line 8). γ represents the average number of vertices an agent traverse per time step in the presence of other agents. Setting $\gamma = 1$ assumes no interference from other agents. In practice, however, agents wait in place to let other agents pass, slowing them down. We determine γ experimentally.

Once agents arrive at their next goal vertex, they queue up behind other agents that visit the vertex as well. To capture this queueing delay, we look for the next available δ consecutive time steps starting at time step $\hat{t}_i$, i.e., time steps that are marked as zero (unoccupied) in time table Q_{v_j}. Let $[\hat{t}_{i,v_j}, \hat{t}_{i,v_j} + \delta)$ be the time interval found (Line 9), where $\hat{t}_{i,v_j}$ is the estimated time step when agent a_i starts the picking action at v_j. We then update $\hat{t}_i$ and $\hat{R}_i$ accordingly (Lines 10-11). Finally, we return $\hat{t}_i$ and $\hat{R}_i$ as the estimated service time and score of agent a_i (Line 12).

Adding a Tour into STE. To add a tour $p_i = (v_1, \ldots, v_m)$ of agent a_i into STE, we can use Algorithm 2. The only changes we need to make are to update $\mathcal{A}_{\mathsf{STE}} \leftarrow \mathcal{A}_{\mathsf{STE}} \cup \{a_i\}$

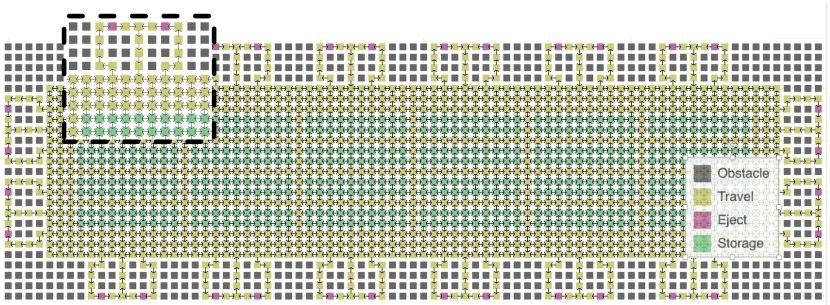

Figure 2: An empty warehouse map adapted from [Wurman *et al.*, 2008] with 32 picking stations spread around the perimeter. Grey cells are obstacles. Yellow, pink or green cells correspond to vertices of graph G that the agents can traverse and occupy. Every arrow pointing from one cell to an adjacent one indicates the direction of the corresponding edge. Pink cells are the eject cells of picking stations and correspond to goal vertices of graph G.

and mark the time intervals of picking actions as occupied by agent a_i in the time table (on Line 9 in Algorithm 2).

Removing a Tour from STE. To remove a tour $p_i = (v_1, \dots, v_m)$ of agent a_i from STE, we update $\mathcal{A}_{\mathsf{STE}} \leftarrow \mathcal{A}_{\mathsf{STE}} \setminus \{a_i\}$. We also mark the time steps that are occupied by agent a_i in the time table Q_{v_j} of each goal vertex v_j as zeros (on Line 9 in Algorithm 2).

5.2 Prioritized Planning with STE

In this subsection, we introduce prioritized planning (PP) with a STE in ROSETTA. Prioritized planning is a popular algorithm for cooperative multi-agent path planning [Silver, 2005] that can efficiently find suboptimal solutions. ROSETTA first uses PP to find an initial solution and then replans tours repeatedly (Lines 3 and 10 in Algorithm 1). We show the pseudocode of PP with STE in Algorithm 3. PP first removes the tours of all agents that it will replan from STE (Lines 3-4). (When PP is used to find an initial solution, these two lines do nothing since $\mathcal{A}^{\mathsf{STE}}$ is empty.) It then randomly shuffles the agents in $\mathcal{A}^{\mathsf{PP}}$ to obtain a random priority ordering (Line 5) and initializes the solution P^{PP} to $\emptyset$ (Line 6). Next, it iterates over all agents in $\mathcal{A}^{\mathsf{PP}}$ in descending order of their priorities (Line 7). For each agent a_i, it plans a (sub)optimal tour p_i that maximizes the estimated score $\hat{R}_i$ w.r.t. STE (Line 8) and updates both STE (Line 9) and P^{PP} (Line 10) accordingly. Finally, PP returns P^{PP} once it has (re)planned a tour for all agents in $\mathcal{A}^{\mathsf{PP}}$ (Line 11).

The final missing piece of Algorithm 3 is how to plan a (sub)optimal tour p_i for agent a_i (Line 8). This subproblem is a variant of the (single-agent) traveling salesman problem with deadlines [Bansal *et al.*, 2004] on a complete directed graph $G_{i,\mathsf{TSP}}$, where the vertices are the start vertex s_i of a_i and the goal vertices relevant for the agent. The cost of each edge in $G_{i,\mathsf{TSP}}$ is dynamic and depends on the time it starts traversing (which can be estimated with the STE). We use a local search algorithm to solve this subproblem.

5.3 Destroy Heuristics

In this subsection, we describe two destroy heuristics that generate the subset of agents $\mathcal{A}'$ to replan in each iteration of LNS (Line 7 in Algorithm 1), namely a random heuristic and a goal-based heuristic. We impose an upper bound k' on the number of agents. The random heuristic obtains $\mathcal{A}'$ by sampling k' agents from $\mathcal{A}$ uniformly at random without replacement. The goal-based heuristic randomly samples a goal vertex $v \in V_{\mathsf{G}}$ and a time step $t \in [T]$, looks up the time table Q_v maintained by the STE and obtains $\mathcal{A}'$ by including k' agents that perform picking actions at v at the time steps closest to t. If there are not a sufficient number of agents, it includes all agents that perform picking actions at v.

6 Empirical Evaluation

In this section, we demonstrate the effectiveness of ROSETTA through extensive experiments. We implement ROSETTA in C++ and conduct our experiments on 2.4 GHz CPUs with 16 GB RAM. We compare ROSETTA to both ST-SAP and LT-SAP on simulated warehouses inspired by KIVA systems [Wurman *et al.*, 2008]. We also compare it to prioritized planning with a STE (denoted by ROSETTA-init), which is the initial solution found by ROSETTA (Line 3 of Algorithm 1). We show the empty warehouse map in Figure 2, which is a 79×25 grid map with 32 picking stations surrounding a 71×17 empty area.

To generate a DA-MATP instance, we construct a graph $G = (V, E)$ whose vertices V correspond to the non-obstacle cells and whose edges E correspond to the arrows in Figure 2. The goal vertices V_{G} correspond to the picking stations (shown in pink). There are k agents, and the start vertex of each agent is randomly sampled from the storage cells (shown in green). We use a time horizon of $T = 600$ time steps, and each picking action takes $\delta = 5$ time steps. For each task $\tau = (a_\tau, v_\tau, d_\tau)$, a_τ, v_τ and d_τ are drawn uniformly at random from $\mathcal{A}, V_{\mathsf{G}}$ and Δ_i, respectively, where we use the deadline distributions $\Delta_1 = \{50, 100, 150, \dots, 600\}$, $\Delta_2 = \{100, 200, \dots, 600\}$, $\Delta_3 = \{200, 400, 600\}$ and $\Delta_4 = \{300, 600\}$. We generate 50 instances for each Δ_i and each number of agents $k \in \{250, 300, 350\}$. For $k = 250, 300$ and 350, we generate $|\mathcal{T}| = 3{,}333, 4{,}000$ and $4{,}666$ tasks, respectively.

For the evaluation, we use RHCR [Li *et al.*, 2021], a lifelong MAPF simulator. RHCR plans collision-free paths for agents with given tours on a rolling-horizon basis. For LT-SAP and ROSETTA, RHCR uses the tours that they generate; and for ST-SAP, RHCR uses the greedy rule to determine the

	algorithms	250 agents		300 agents		350 agents	
		#on-time	throughput	#on-time	throughput	#on-time	throughput
Δ_1	ST-SAP	1,880	2,787	2,102	3,252	2,213	3,622
	LT-SAP	2,379 (+26.5%)	3,199 (+14.8%)	2,478 (+17.9%)	3,447 (+06.0%)	2,501 (+13.0%)	3,507 (-03.2%)
	ROSETTA-init	2,435 (+29.5%)	**3,249 (+16.6%)**	2,736 (+30.1%)	3,708 (+14.0%)	2,866 (+29.5%)	3,956 (+09.2%)
	ROSETTA-30	2,441 (+29.8%)	3,244 (+16.4%)	2,742 (+30.4%)	3,720 (+14.4%)	2,891 (+30.4%)	3,991 (+10.2%)
	ROSETTA-60	**2,445 (+30.1%)**	3,246 (+16.5%)	**2,760 (+31.3%)**	**3,744 (+15.1%)**	**2,952 (+33.3%)**	**4,109 (+13.4%)**
Δ_2	ST-SAP	2,190	2,801	2,489	3,334	2,638	3,639
	LT-SAP	2,508 (+14.5%)	3,211 (+14.6%)	2,677 (+07.6%)	3,571 (+07.1%)	2,542 (-03.6%)	3,452 (-05.1%)
	ROSETTA-init	2,549 (+16.3%)	**3,255 (+16.2%)**	2,880 (+15.7%)	3,719 (+11.5%)	3,007 (+14.0%)	3,952 (+08.6%)
	ROSETTA-30	2,558 (+16.8%)	3,246 (+15.9%)	2,895 (+16.3%)	3,740 (+12.2%)	3,066 (+16.2%)	3,982 (+09.4%)
	ROSETTA-60	**2,562 (+17.0%)**	3,250 (+16.0%)	**2,905 (+16.7%)**	**3,768 (+13.0%)**	**3,100 (+17.5%)**	**4,117 (+13.1%)**
Δ_3	ST-SAP	2,502	2,814	2,883	3,286	3,141	3,648
	LT-SAP	2,708 (+08.2%)	3,228 (+14.7%)	2,822 (-02.1%)	3,453 (+05.1%)	3,093 (-01.5%)	3,506 (-03.9%)
	ROSETTA-init	2,739 (+09.5%)	**3,262 (+15.9%)**	3,085 (+07.0%)	3,703 (+12.7%)	3,214 (+02.3%)	3,940 (+08.0%)
	ROSETTA-30	**2,749 (+09.9%)**	3,260 (+15.8%)	3,094 (+07.3%)	3,715 (+13.1%)	3,227 (+02.7%)	3,932 (+07.8%)
	ROSETTA-60	2,747 (+09.8%)	3,253 (+15.6%)	**3,119 (+08.2%)**	**3,764 (+14.5%)**	**3,347 (+06.6%)**	**4,126 (+13.1%)**
Δ_4	ST-SAP	2,633	2,815	3,045	3,281	3,350	3,643
	LT-SAP	2,882 (+09.5%)	3,235 (+14.9%)	3,047 (+00.0%)	3,495 (+06.5%)	3,051 (-09.0%)	3,562 (-02.2%)
	ROSETTA-init	2,900 (+10.6%)	**3,267 (+16.1%)**	3,264 (+07.2%)	3,703 (+12.9%)	3,425 (+02.2%)	3,941 (+08.2%)
	ROSETTA-30	2,913 (+10.6%)	3,266 (+16.0%)	3,280 (+07.7%)	3,725 (+13.5%)	3,429 (+02.4%)	3,938 (+08.1%)
	ROSETTA-60	**2,914 (+10.7%)**	3,264 (+15.9%)	**3,309 (+08.7%)**	**3,777 (+15.1%)**	**3,571 (+06.6%)**	**4,141 (+13.7%)**

Table 1: Performance metrics on the empty warehouse map averaged over 50 instances. "#on-time" is the number of on-time tasks (higher is better) and "throughput" is the number of completed tasks (higher is better). The numbers in parentheses are the improvements (in percent) over ST-SAP. The entries with the best performance are shown in bold.

next goal vertex on the fly during execution. We evaluate the number of on-time tasks, the throughput (i.e., the number of completed tasks) and the station utilization (i.e., the number of time steps where a picking station is occupied) to evaluate the algorithms. We use ROSETTA with a runtime limit of 30 and 60 seconds, denote by ROSETTA-30 and ROSETTA-60. We use of $k' = 8$ an upper bound on the cardinality of the agent sets generated by the destroy heuristics.[1] We use $\lambda = 0.8$ as travel speed of the agents for both LT-SAP and ROSETTA.[2]

Results. Table 1 shows the number of on-time tasks and the throughput for the empty warehouse map shown in Figure 2 for different numbers of agents k and deadline distributions Δ_i averaged over 50 instances. The initial solution of ROSETTA (denoted by ROSETTA-init) already delivers better performance than ST-SAP and LT-SAP in all cases w.r.t. both metrics. Given additional time to run LNS, ROSETTA improves the solution quality further. For example, ROSETTA-60 completes 6.6% to 33.3% more tasks on time than ST-SAP on the empty warehouse map. and increases the throughput by at least 13.0% in all cases, even though it does not attempt to optimize this metric.

We compare ROSETTA to ST-SAP and LT-SAP further on the empty warehouse map with deadline distribution Δ_1. Table 2 shows the number of time steps that an agent needs to travel from its current goal vertex to the next one, the station utilization measured by the number of time steps that a goal vertex is occupied and the number of picking actions performed by the agents. ROSETTA shortens the agents' travel time between picking stations, coordinates agents to visit more of them and thus increases the station utilization compared to both ST-SAP and LT-SAP.

algorithms	task time	utilization	#picking action
250 agents			
ST-SAP	49.2	351	2,246
LT-SAP	33.8 (-31.2%)	410 (+16.8%)	2,626 (+16.9%)
ROSETTA-60	**33.3 (-32.2%)**	**416 (+18.5%)**	**2,649 (+17.9%)**
300 agents			
ST-SAP	50.2	408	2,607
LT-SAP	36.5 (-27.3%)	435 (+06.6%)	2,778 (+06.6%)
ROSETTA-60	**36.1 (-28.0%)**	**469 (+15.0%)**	**3,000 (+15.1%)**
350 agents			
ST-SAP	51.8	450	2,876
LT-SAP	39.7 (-23.4%)	436 (-03.1%)	2,787 (-03.1%)
ROSETTA-60	**39.0 (-24.6%)**	**491 (+09.1%)**	**3,139 (+09.1%)**

Table 2: Performance metrics on the empty warehouse map averaged over 50 instances with deadline distribution Δ_1. "Task time" is the number of time steps that an agent needs to travel from its current goal vertex to the next one (lower is better), "utilization" is the station utilization (higher is better), and "#picking action" is the total number of picking actions performed by the agents (higher is better). The numbers in parentheses are the improvements (in percent) over ST-SAP. The entries with the best performance are shown in bold.

7 Conclusion

Motivated by same-day delivery promises of e-commerce companies, we proposed the new challenge of coordinating multiple agents in warehouses to visit multiple goal locations by given deadlines in form of the DA-MATP problem. We designed the novel multi-agent planning algorithm ROSETTA that uses travel and queuing time estimates to decouple multi-agent tour planning from multi-agent path finding, allowing it to scale to realistic problem sizes. Empirical evaluations on realistic warehouse maps showed that ROSETTA outperformed two baselines significantly in terms of the resulting number of tasks completed by their deadlines, the throughput and the station utilization.

[1] We experiment with $k' \in \{4, 8, 16\}$ on a separate set of instances for validation, and $k' = 8$ performed the best.

[2] We experimented with $\lambda \in \{0.6, 0.7, 0.8, 0.9, 1\}$ on a separate set of instances for validation, and $\lambda = 0.8$ performed the best. Also, $\lambda = 0.8$ aligns with the travel speed of agents derived from data from RHCR's simulations.

Acknowledgments

This paper reports on research done while Taoan Huang was an intern at Amazon Robotics. The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1409987, 1724392, 1837779, 1935712, 2121028 and 2112533 as well as a gift from Amazon Robotics. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, or the U.S. government.

References

[Ahuja *et al.*, 2002] Ravindra K. Ahuja, Özlem Ergun, James B. Orlin, and Abraham P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1-3):75–102, 2002.

[Alexander, 2020] Jon Alexander. Same-day delivery just got faster. https://www.aboutamazon.com/news/operations/same-day-delivery-just-got-faster, 2020. Accessed: 2021-11-13.

[Bansal *et al.*, 2004] Nikhil Bansal, Avrim Blum, Shuchi Chawla, and Adam Meyerson. Approximation algorithms for deadline-TSP and vehicle routing with time-windows. In *the Annual ACM Symposium on Theory of Computing*, pages 166–174, 2004.

[Čáp *et al.*, 2015] Michal Čáp, Jiří Vokřínek, and Alexander Kleiner. Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures. In *the International Conference on Automated Planning and Scheduling*, pages 324–332, 2015.

[Farinelli *et al.*, 2020] Alessandro Farinelli, Antonello Contini, and Davide Zorzi. Decentralized task assignment for multi-item pickup and delivery in logistic scenarios. In *the International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1843–1845, 2020.

[Helsgaun, 2009] Keld Helsgaun. General k-opt submoves for the Lin–Kernighan TSP heuristic. *Mathematical Programming Computation*, 1(2):119–163, 2009.

[Li *et al.*, 2021] Jiaoyang Li, Andrew Tinka, Scott Kiesel, Joseph W. Durham, T. K. Satish Kumar, and Sven Koenig. Lifelong multi-agent path finding in large-scale warehouses. In *the AAAI Conference on Artificial Intelligence*, pages 11272–11281, 2021.

[Lin and Kernighan, 1973] Shen Lin and Brian W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.

[Liu *et al.*, 2019] Minghua Liu, Hang Ma, Jiaoyang Li, and Sven Koenig. Task and path planning for multi-agent pickup and delivery. In *the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2019.

[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 *the International Conference on Autonomous Agents and Multi-Agent Systems*, pages 837–845, 2017.

[Ma *et al.*, 2018] Hang Ma, Glenn Wagner, Ariel Felner, Jiaoyang Li, T.K. Kumar, and Sven Koenig. Multi-agent path finding with deadlines. In *the International Joint Conference on Artificial Intelligence*, 2018.

[Ma *et al.*, 2019] Hang Ma, Wolfgang Hönig, T.K. Satish Kumar, Nora Ayanian, and Sven Koenig. Lifelong path planning with kinematic constraints for multi-agent pickup and delivery. In *the AAAI Conference on Artificial Intelligence*, pages 7651–7658, 2019.

[Moon, 2013] Mariella Moon. Eat your heart out: Chinese e-commerce firm delivers packages within hours. https://www.engadget.com/2013-06-01-jingdong-china-same-day-delivery.html, 2013. Accessed: 2021-11-13.

[Ren *et al.*, 2021] Zhongqiang Ren, Sivakumar Rathinam, and Howie Choset. MS*: A New Exact Algorithm for Multi-agent Simultaneous Multi-goal Sequencing and Path Finding. *arXiv preprint arXiv:2103.09979*, 2021.

[Ropke and Pisinger, 2006] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455–472, 2006.

[Silver, 2005] David Silver. Cooperative pathfinding. In *the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pages 117–122, 2005.

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

[Surynek, 2021] Pavel Surynek. Multi-goal multi-agent path finding via decoupled and integrated goal vertex ordering. In *the International Symposium on Combinatorial Search*, pages 197–199, 2021.

[Wen *et al.*, 2012] Xingang Wen, Yinfeng Xu, and Huili Zhang. Online traveling salesman problem with deadline and advanced information. *Computers & Industrial Engineering*, 63(4):1048–1053, 2012.

[Wu *et al.*, 2021] Xiaohu Wu, Yihao Liu, Xueyan Tang, Wentong Cai, Funing Bai, Gilbert Khonstantine, and Guopeng Zhao. Multi-agent pickup and delivery with task deadlines. In *the International Symposium on Combinatorial Search*, pages 206–208, 2021.

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

[Yu and LaValle, 2013] Jingjin Yu and Steven M. LaValle. Multi-agent path planning and network flow. In *Algorithmic Foundations of Robotics X*, pages 157–173. Springer, 2013.