Deadline-Aware Multi-Agent Tour Planning

Taoan Huang[1], Vikas Shivashankar[2], Michael Caldara[2], Joseph Durham[2], Jiaoyang Li[3], Bistra Dilkina[1], Sven Koenig[1]

[1]University of Southern California
[2]Amazon Robotics
[3]Carnegie Mellon University
taoanhua@usc.edu, {vikshiv, caldaram, josepdur}@amazon.com, jiaoyangli@cmu.edu, {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, D'Andrea, and Mountz 2008). Instead of human operators walking around in storage areas to fetch inventory items, mobile *agents* transport them from storage areas 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 carefully to ensure that the order deadlines are met. Another challenge is *resource contention*.

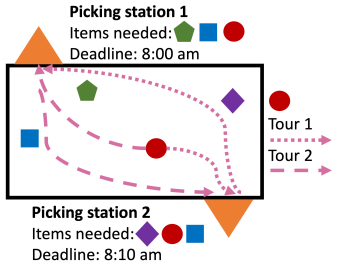

Figure 1: An illustrative example of the planning problem in warehouses for same-day delivery. The agents are represented by the shapes that correspond to the items they carry. There are two picking stations on the perimeter of the warehouse that each need certain items by a given deadline. The red-circle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

A single agent might be needed at multiple stations. For example, 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. 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 the blue square, need to visit more than one station, requiring the system to decide on the order in which the agents visit them. 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.

DA-MATP is a generalization of multi-agent path finding (MAPF) (Stern et al. 2019), the popular problem of planning collision-free paths for a team of agents. 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), an efficient and effective algorithm for solving the DA-MATP problem. Solving it 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. 2021b). 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 a MAPF algorithm (Li et al. 2021b) 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, D'Andrea, and Mountz 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, Contini, and Zorzi 2020). Multi-goal planning has been studied in the context of lifelong MAPF (Li et al. 2021b) and lifelong MAPD (Ma et al. 2017, 2019), where the goal orderings are either not part of the decision space or determined online. Optimal MAPF algorithms (Surynek 2021; Ren, Rathinam, and Choset 2021) have been proposed 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 collision-free paths during execution. Ma et al. (2018) study a variant of MAPF with a single deadline where they maximize the number of agents that reach their given goal vertices before the deadline. MAPD with deadlines have also been studied where each task has its own deadline. The state-of-the-art is a greedy algorithm (Wu et al. 2021) that assigns tasks to agents greedily on the fly to maximize the number of tasks completed by their 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 (Wu et al. 2021), introduced in Section 4, and compare it to ROSETTA.

LNS is a popular algorithm to solve planning and routing problems, such as vehicle routing (Ropke and Pisinger 2006; Azi, Gendreau, and Potvin 2014) and traveling salesman (Smith and Imeson 2017) problems. It has also recently been applied to solve the classical MAPF problem (Li et al. 2021a, 2022; Huang et al. 2022).

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 $\mathcal{A} = \{a_1, \ldots, a_k\}$ that allows them to finish a set of tasks $\mathcal{T}$ cooperatively 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 $t + \delta - 1$ 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 and performs a picking action at or before time step T, even if the task is late. Multiple tasks can be completed by an agent at a goal vertex simultaneously with one picking action.

Warehouses typically have a separate *stow* process to top up agents with new inventory (Wurman, D'Andrea, and Mountz 2008). 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. However, ROSETTA can also be applied to the case where agents carry heterogeneous items and/or the picking time is a linear function of the number of items.

Path Planning and Execution. The movement of the agents in DA-MATP 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 same edge in the opposite direction (if it exists) at the same time step. A *collision* occurs when either one of the movement constraints is violated. A *tour* p_i or, synonymously, 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 MAPF solver, 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, Vokřínek, and Kleiner 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, Vokřínek, and Kleiner 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 $R_i = \sum_{\tau \in \mathcal{T}_i} \mathbf{1}[t_{i,v_\tau} \leq d_\tau]$ 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 goal vertex. Suppose that agent a_i finishes a 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 but has at least a task for it, ST-SAP first finds the task with the soonest deadline $\tilde{t}_{v'}$ whose goal vertex is v' and 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.

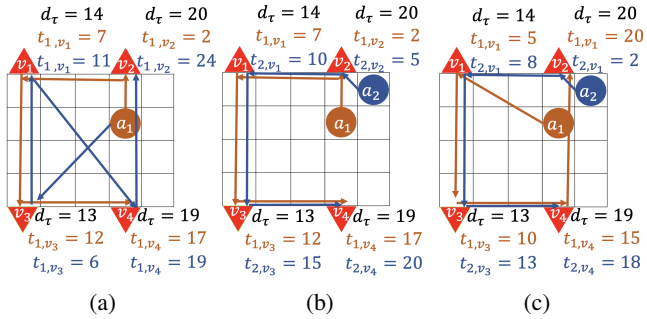

Figure 2: A DA-MATP instance. (a) shows the optimal tour (shown in brown) and a suboptimal tour (shown in blue) for one agent a_1. With two agents a_1 and a_2, (b) shows the suboptimal tours of LT-SAP (both agents follow their shortest paths to complete their tours, except that a_2 waits at its start vertex for 3 time steps) and (c) shows the optimal tours (both agents follow their shortest paths to complete their tours, except that a_2 waits at the second cell in the top row at time step 5 to avoid a collision with a_1).

ST-SAP runs in polynomial time but is myopic. It optimizes only for the short term, i.e., it constructs the tours 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 plans an individually optimal tour for each agent as if it were the only agent on the graph. Since it is NP-hard to plan the individually optimal tour even for a single agent, we instead use the same local search that finds a suboptimal tour used in ROSETTA in our experiments, which is introduced in Section 6.2.

5 A Motivating Example of DA-MATP

In this section, we motivate ROSETTA by showing an example of a DA-MATP instance that demonstrates the ineffectiveness of both SAP algorithms.

Figure 2 shows a DA-MATP instance on a grid map with a 4×5 empty area. The four cells off the map perimeter labeled $v_1, \ldots, v_4$ are goal vertices. Agents can move from a cell to any of its four adjacent cells. Consider time horizon $T = 25$, and assume that picking actions are executed in $\delta = 1$ time step. Each agent has to execute one task at each goal vertex, and the deadlines at v_1, v_2, v_3 and v_4 are $14, 20, 13$ and 19, respectively, for both agents. Figure 2a shows a DA-MATP instance with one agent a_1. The path planner finds paths that minimize the sum of the travel time of the agents. The optimal tour for a_1 (shown in brown) is (v_2, v_1, v_3, v_4). The agent arrives at v_1, v_2, v_3 and v_4 at time step $7, 2, 12$ and 17, respectively, and completes all tasks on time. This is the tour of LT-SAP. The tour of ST-SAP (shown in blue) is (v_3, v_1, v_4, v_2). It arrives at v_1, v_2, v_3 and v_4 at time step $11, 24, 6$ and 19, respectively, and misses the deadline at v_2.

Figures 2b and 2c show the same DA-MATP instance with

one more agent a_2. The tours of LT-SAP for both agents are the same (v_2, v_1, v_3, v_4). Agent a_1 arrives at v_1, v_2, v_3 and v_4 at time step $7, 2, 12$ and 17, respectively, and completes all tasks on time. Agent a_2 arrives at v_1, v_2, v_3 and v_4 at time step $10, 5, 15$ and 20, respectively, and misses the deadline at v_3 and v_4. Figure 2c shows an optimal solution on the same DA-MATP instance with two agents, demonstrating that, if we could coordinate both agents smartly, it is possible to complete all tasks on time.

6 ROSETTA

To overcome the weaknesses of SAP algorithms for DA-MATP, we propose ROSETTA, 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 their start vertices, the set of goal vertices V_G, the time horizon $[T]$ and the set of tasks $\mathcal{T}$ (Line 1). 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 local search algorithm that, in each iteration, destroys and re-optimizes a part of the solution. We propose two destroy heuristics for ROSETTA to determine which part of the solution to destroy, 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 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) from the solution and uses PP to replan them (Line 10). If the replanned tours P^+ improve the solution (Line 12), then ROSETTA replaces P^- with P^+ (Line 13). Finally, it passes the set of tours P (Line 16) to a MAPF solver to generate collision-free paths w.r.t. P in real-time. In experiments, we use RHCR (Li et al. 2021b), a lifelong MAPF algorithm to plan such paths. ROSETTA is designed to produce high-quality tours fast (typically in under a minute) (Lines 1-15) and then collision-free paths during execution (Line 16) in a longer time, upper bounded by the length of time horizon $[T]$ (typically $[T]$ spans 30-60 minutes).

ROSETTA relies on a service time estimator during planning (Lines 2, 3, 7 and 10). To determine the exact 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. However, planning such paths for hundreds of agents and hundreds of time steps on large graphs can take several 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) to estimate the score. The novelties of the STE are three-fold: (1) In a warehouse, the stations are usually the most congested areas on a warehouse floor. Thus, the STE does not precisely consider all collisions. Instead, it only considers collisions at the stations. (2) As for the travel times from one station to another, they usually have smaller variances. Thus, the STE approximates them using the shortest distance from one to the other divided by a constant speed. (3) More importantly, the STE enables multi-agent planning for DA-MATP since it implicitly considers the interaction between agents. In contrast, LT-SAP estimates the travel times of agents without considering the other agents. Specifically, ROSETTA's 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.

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: $\quad P^+ \leftarrow \mathsf{prioritizedPlanning}(I, \mathcal{A}', \mathsf{STE})$ ▷ Update STE with P^+
11: $\quad$ Update the weights $\boldsymbol{w}$ of the destroy heuristics
12: $\quad$ **if** P^+ has a higher estimated score than P^- **then**
13: $\quad\quad P \leftarrow (P \setminus P^-) \cup P^+$ ▷ Accept P^+
14: $\quad$ **else**
15: $\quad\quad \mathsf{STE} \leftarrow \mathsf{STE}'$ ▷ Restore STE
16: Pass P to a MAPF solver for execution

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

6.1 Service Time Estimator

In this subsection, we describe the implementation of the STE. The STE, denoted by 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

Algorithm 2: Estimate the service time and score of a tour

1: **Input:** 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: **return** the estimated service time and score for 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: $\hat{t}_i \leftarrow \hat{t}_i + \lceil \mathsf{dist}(v_{j-1}, v_j)/\gamma \rceil$
9: Find the minimum $\hat{t}_{i,v_j} \geq \hat{t}_i$ s.t. $[\hat{t}_{i,v_j}, \hat{t}_{i,v_j} + \delta)$ is not occupied in Q_{v_j}
10: $\hat{R}_i \leftarrow \hat{R}_i +$ the number of tasks completed on time at v_j at $\hat{t}_{i,v_j}$ by a_i
11: $\hat{t}_i \leftarrow \hat{t}_{i,v_j} + \delta - 1$
12: Save the estimated service time $\hat{t}_i$ and score $\hat{R}_i$ in STE and return them

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

1: **Input:** a DA-MATP instance I, a set of agents $\mathcal{A}^{\mathsf{PP}} \subseteq \mathcal{A}$ and STE.
2: Retrieve $\mathcal{A}^{\mathsf{STE}}$
3: **for** $a_i \in \mathcal{A}^{\mathsf{PP}} \cap \mathcal{A}^{\mathsf{STE}}$ **do**
4: 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: $p_i \leftarrow$ findSingleAgentSuboptimalTour(I, a_i, STE)
9: Add tour p_i to STE
10: $P^{\mathsf{PP}} \leftarrow P^{\mathsf{PP}} \cup \{p_i\}$
11: **return** P^{PP}

service time for a_i, the STE separately estimates (1) the travel time needed to get from one goal vertex to the next one and (2) the time that the agent spends waiting 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) 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) and 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 traverses 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 vertices at $\hat{t}_i$, they queue up behind other agents

Algorithm 4: findSingleAgentSuboptimalTour(I, a_i, STE)

1: **Input:** a DA-MATP instance I, an agent a_i and STE
2: Generate a random tour $p_i = (v_1, \ldots, v_m)$ for agent a_i
3: $\hat{R}_i \leftarrow$ estimate the score of p_i with STE
4: improvementFound $\leftarrow$ True
5: **while** improvementFound **do**
6: improvementFound $\leftarrow$ False
7: **for** each feasible κ-opt operation $\sigma(\cdot)$ **do**
8: $p'_i \leftarrow \sigma(p_i)$
9: $\hat{R}'_i \leftarrow$ estimate the score of p'_i with STE
10: **if** $\hat{R}'_i > \hat{R}_i$ **then**
11: Replace $p_i, \hat{R}_i$ with $p'_i, \hat{R}'_i$
12: improvementFound $\leftarrow$ True
13: **break**
14: **return** p_i

that visit the vertices 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 for agent a_i (Line 12).

Adding a Tour to STE. To add a tour $p_i = (v_1, \ldots, v_m)$ of agent a_i to STE, we use Algorithm 2. The only changes we make are to update $\mathcal{A}_{\mathsf{STE}} \leftarrow \mathcal{A}_{\mathsf{STE}} \cup \{a_i\}$ 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 $p_i = (v_1, \ldots, v_m)$ of agent a_i from STE, we update $\mathcal{A}_{\mathsf{STE}} \leftarrow \mathcal{A}_{\mathsf{STE}} \setminus \{a_i\}$. We mark the time steps occupied by agent a_i in the time table Q_{v_j} of each goal vertex v_j as unoccupied.

6.2 Prioritized Planning with STE

In this subsection, we introduce Prioritized Planning (PP) with a STE in ROSETTA. Prioritized planning (Silver 2005) is a popular algorithm for MAPF that efficiently finds 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 order (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 suboptimal 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).

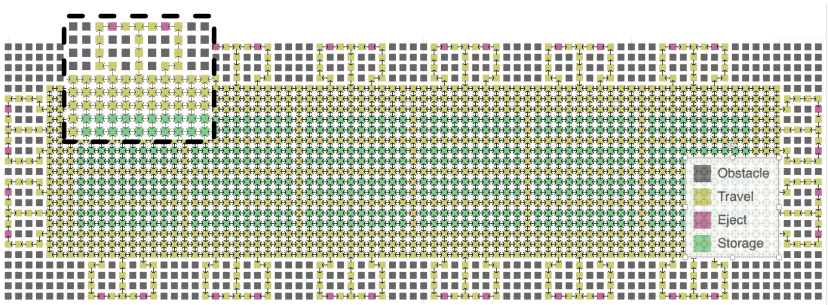

Figure 3: An empty warehouse map adapted from (Wurman, D'Andrea, and Mountz 2008) with 32 picking stations spread around the perimeter. Grey cells are obstacles. Yellow, pink and green cells correspond to vertices of graph G that the agents can traverse and occupy. Every arrow pointing from one cell to an adjacent one indicates the direction of the corresponding edge. Pink cells are the eject cells of picking stations where humans interact with the agents and correspond to goal vertices.

	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: The number of on-time tasks (denoted by "#on-time") and the throughput on the empty warehouse map averaged over 50 instances. The numbers in parentheses are the improvements (in percent) over ST-SAP. The entries with the best performance are shown in bold.

The missing piece of Algorithm 3 is how to plan a suboptimal tour p_i for agent a_i (Line 8). This subproblem is a variant of the traveling salesman problem with deadlines (Bansal et al. 2004) on a complete weighted 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 the agent starts traversing it (which can be estimated with the STE). We use a local search algorithm to solve this subproblem as follows.

Given a complete weighted directed graph, a start vertex on the graph and a deadline for each vertex, the deadline-TSP is to find a path starting at the start vertex that visits as many vertices by their deadlines as possible. The deadline-TSP has been studied theoretically, and approximation algorithms have been proposed for it (Bansal et al. 2004; Wen, Xu, and Zhang 2012). We adopt a local search approach with κ-opt operations, which is one of the most effective heuristics for solving the deadline-TSP and its variants (Lin and Kernighan 1973; Helsgaun 2009). Algorithm 4 uses this approach to find a suboptimal tour for each agent. It takes as input a DA-MATP instance, an agent a_i and STE. It first generates an initial tour p_i for the agent (Line 3) and estimates its score $\hat{R}_i$ with STE (Line 4). It then repeatedly applies κ-opt operations to improve the tour until no improvement can be found any longer (Lines 5-13). A κ-opt operation $\sigma(p_i)$ replaces κ edges of the tour with another κ edges to form a new tour. In our experiments, we use $\kappa = 3$.

6.3 Destroy Heuristics

In this subsection, we describe two destroy heuristics that generate the subset of agents $\mathcal{A}'$ to replan in LNS (Line 7 in Algorithm 1), namely a random heuristic and a goal-

	algorithms	250 agents		300 agents		350 agents	
		#on-time	throughput	#on-time	throughput	#on-time	throughput
Δ_1	ST-SAP	1,833	2,756	2,036	3,209	2,116	3,534
	LT-SAP	2,349 (+28.2%)	3,160 (+14.7%)	2,442 (+19.9%)	3,398 (+05.9%)	2,427 (+14.7%)	3,526 (-00.2%)
	ROSETTA-init	2,430 (+32.6%)	**3,244 (+17.7%)**	2,712 (+33.2%)	3,643 (+13.5%)	2,818 (+33.2%)	3,843 (+08.7%)
	ROSETTA-30	2,434 (+32.8%)	3,223 (+17.0%)	2,724 (+33.8%)	3,665 (+14.2%)	2,870 (+35.6%)	4,007 (+13.4%)
	ROSETTA-60	**2,439 (+33.1%)**	3,236 (+17.4%)	**2,750 (+35.1%)**	**3,733 (+16.3%)**	**2,912 (+37.6%)**	**4,034 (+14.1%)**
Δ_2	ST-SAP	2,138	2,764	2,419	3,233	2,557	3,581
	LT-SAP	2,485 (+16.2%)	3,188 (+15.3%)	2,562 (+06.0%)	3,380 (+04.5%)	2,619 (+02.4%)	3,537 (-01.2%)
	ROSETTA-init	2,539 (+18.7%)	**3,242 (+17.3%)**	2,846 (+17.7%)	3,669 (+13.5%)	2,967 (+16.0%)	3,881 (+08.4%)
	ROSETTA-30	**2,546 (+19.1%)**	3,232 (+16.9%)	2,874 (+18.8%)	3,703 (+14.5%)	3,010 (+17.7%)	4,033 (+12.6%)
	ROSETTA-60	2,544 (+19.0%)	3,226 (+16.7%)	**2,881 (+19.1%)**	**3,739 (+15.7%)**	**3,050 (+19.3%)**	**4,045 (+13.0%)**
Δ_3	ST-SAP	2,459	2,781	2,827	3,240	3,058	3,577
	LT-SAP	2,660 (+08.2%)	3,171 (+14.0%)	2,798 (-0.01%)	3,435 (+06.0%)	2,740 (-10.4%)	3,464 (-03.2%)
	ROSETTA-init	2,724 (+10.8%)	3,243 (+16.6%)	3,075 (+08.8%)	3,691 (+13.9%)	3,172 (+03.7%)	3,858 (+07.9%)
	ROSETTA-30	2,739 (+11.4%)	3,247 (+16.7%)	**3,089 (+09.3%)**	3,710 (+14.5%)	3,269 (+06.9%)	3,974 (+11.1%)
	ROSETTA-60	**2,740 (+11.4%)**	**3,249 (+16.8%)**	3,086 (+09.2%)	**3,714 (+14.6%)**	**3,304 (+08.0%)**	**4,048 (+13.2%)**
Δ_4	ST-SAP	2,578	2,767	2,991	3,231	3,271	3,566
	LT-SAP	2,851 (+10.6%)	3,204 (+15.8%)	2,982 (-00.3%)	3,423 (+05.9%)	2,971 (-09.2%)	3,482 (-02.4%)
	ROSETTA-init	2,886 (+11.9%)	3,236 (+16.9%)	3,211 (+07.4%)	3,634 (+12.5%)	3,339 (+02.1%)	3,822 (+07.2%)
	ROSETTA-30	2,891 (+12.1%)	3,254 (+17.6%)	3,240 (+08.3%)	3,663 (+13.4%)	3,442 (+05.2%)	3,960 (+11.0%)
	ROSETTA-60	**2,902 (+12.6%)**	**3,261 (+17.9%)**	**3,263 (+09.1%)**	**3,713 (+14.9%)**	**3,527 (+07.8%)**	**4,085 (+14.6%)**

Table 2: The number of on-time tasks (denoted by "#on-time") and the throughput on the warehouse map with 10% randomly blocked cells averaged over 50 instances. The numbers in parentheses are the improvements (in percent) over ST-SAP. The entries with the best performance are shown in bold.

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 goal vertex $v \in V_{\mathsf{G}}$ and time step $t \in [T]$, then looks up the time table Q_v maintained by the STE and obtains $\mathcal{A}'$ by including the k' agents that perform picking actions at v at the time steps closest to t. If there are not enough agents, it includes all agents that perform picking actions at v.

7 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, D'Andrea, and Mountz 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 3, which is a 79×25 grid map with 32 picking stations surrounding a 71×17 empty area. We also run experiments on the same warehouse map with 10% randomly blocked cells in the 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 (including eject, travel and storage cells that are all traversable) and whose edges E correspond to the arrows in Figure 3. The goal vertices V_{G} correspond to the eject cells (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 $\Delta_1 = \{50, 100, 150, \ldots, 600\}$, $\Delta_2 = \{100, 200, \ldots, 600\}$, $\Delta_3 = \{200, 400, 600\}$ 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 our evaluation, we use RHCR (Li et al. 2021b), a lifelong MAPF algorithm. 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 next goal vertex on the fly during execution. Once an agent completes all its tasks, RHCR routes it back to its start vertex to avoid staying, and thus causing congestion, at the picking station. 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, denoted by ROSETTA-30 and ROSETTA-60, respectively. We use $k' = 8$ as an upper bound on the cardinality of the agent sets generated by the destroy heuristics. We experiment with $k' \in \{4, 8, 16\}$ on a separate set of instances for validation, and $k' = 8$ performed the best. We use $\lambda = 0.8$ as travel speed of the agents for both LT-SAP and ROSETTA. We will also show that different choices of λ can outperform both ST-SAP and LT-SAP.

Results. Table 1 shows the number of on-time tasks and the throughput for the empty warehouse map shown in Fig-

algorithms	task time	utilization	#picking
250 agents			
ST-SAP	49.2	351	2,246
LT-SAP	33.8	410	2,626
ROSETTA-60	**33.3**	**416**	**2,649**
300 agents			
ST-SAP	50.2	408	2,607
LT-SAP	36.5	435	2,778
ROSETTA-60	**36.1**	**469**	**3,000**
350 agents			
ST-SAP	51.8	450	2,876
LT-SAP	39.7	436	2,787
ROSETTA-60	**39.0**	**491**	**3,139**

Table 3: The number of time steps that an agent takes to travel from one goal vertex to the next one (denoted by "task time"), station utilization (denoted by "utilization") and the total number of picking actions (denoted by "#picking") performed by the agents on the empty warehouse map with deadline distribution Δ_1 averaged over 50 instances. The entries with the best performance are shown in bold.

ure 3 for different numbers of agents k and deadline distributions Δ_i averaged over 50 instances. Table 2 shows the same results for the same warehouse map with 10% random obstacles generated in the 71×17 empty area. 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 7.8% to 37.6% more on the warehouse map with 10% randomly blocked cells. ROSETTA-60 also 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 in more details on the empty warehouse map with deadline distribution Δ_1. Table 3 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.

The runtimes for ST-SAP, LT-SAP and ROSETTA-init are 1.4, 1.8 and 2.8 seconds, respectively, averaged over instances with 350 agents across four deadline distributions (the runtime is not sensitive to different deadline distributions) tested on the empty warehouse map. ROSETTA-30 (ROSETTA-60) achieves a 30.4% (33.3%) improvement on the number of on-time tasks in 30 (60) seconds. This runtime is acceptable in practice since we need to run a DA-MATP algorithm only once every 30-60 minutes, depending on the time horizon. Given the output of ROSETTA, RHCR needs on average 90, 310 and 735 seconds to plan collision-free paths of 250, 300 and 350 agents, respectively. The runtime on the warehouse map with 10% randomly blocked

	algorithms	350 agents	
		#on-time	throughput
Δ_1	ST-SAP	2,116	3,534
	LT-SAP ($\gamma = 1$)	2,201	3,537
	ROSETTA-60 ($\gamma = 1$)	2,230	3,842
	LT-SAP ($\gamma = 0.8$)	2,427	3526
	ROSETTA-60 ($\gamma = 0.8$)	**2,912**	**4,034**
Δ_2	ST-SAP	2,557	3,581
	LT-SAP ($\gamma = 1$)	2,570	3,496
	ROSETTA-60 ($\gamma = 1$)	2,656	3,859
	LT-SAP ($\gamma = 0.8$)	2,619	3,537
	ROSETTA-60 ($\gamma = 0.8$)	**3,050**	**4,045**
Δ_3	ST-SAP	3,058	3,577
	LT-SAP ($\gamma = 1$)	2,634	3,477
	ROSETTA-60 ($\gamma = 1$)	3,122	3,801
	LT-SAP ($\gamma = 0.8$)	2,740	3,464
	ROSETTA-60 ($\gamma = 0.8$)	**3,304**	**4,048**
Δ_4	ST-SAP	3,271	3,566
	LT-SAP ($\gamma = 1$)	2,903	3,402
	ROSETTA-60 ($\gamma = 1$)	3,306	3,807
	LT-SAP ($\gamma = 0.8$)	2,971	3,482
	ROSETTA-60 ($\gamma = 0.8$)	**3,527**	**4,085**

Table 4: The number of on-time tasks (denoted by "#on-time") and the throughput on the warehouse map with 10% randomly blocked cells averaged over 50 instances. The entries with the best performance are shown in bold.

cells are similar and therefore omitted. We also observe that ROSETTA coordinates the movements of the agents more evenly across the areas around the picking stations than ST-SAP. For ST-SAP, the areas around some picking stations are on average more congested than those around the others.

Table 4 shows the numbers of on-time tasks and the throughput for $\gamma = 0.8$ and 1 on the warehouse map with 10% randomly blocked cells for 350 agents and different deadline distributions Δ_i. Setting $\gamma = 1$ corresponds to using the lengths of the shortest paths of agents as a proxy for their estimating travel times. The table shows that, while ROSETTA with $\gamma = 1$ still outperforms ST-SAP, ROSETTA with $\gamma = 0.8$ performs even better, demonstrating the advantage of tuning γ.[1]

8 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 also tried $\lambda \in \{0.6, 0.7, 0.8, 0.9, 1\}$ and $\lambda = 0.8$ still performed the best. It also aligns with the travel speed of agents derived 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, R. K.; Ergun, Ö.; Orlin, J. B.; and Punnen, A. P. 2002. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1-3): 75–102.

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

Azi, N.; Gendreau, M.; and Potvin, J.-Y. 2014. An adaptive large neighborhood search for a vehicle routing problem with multiple routes. *Computers & Operations Research*, 41: 167–173.

Bansal, N.; Blum, A.; Chawla, S.; and Meyerson, A. 2004. Approximation algorithms for deadline-TSP and vehicle routing with time-windows. In *the Annual ACM Symposium on Theory of Computing*, 166–174.

Čáp, M.; Vokřínek, J.; and Kleiner, A. 2015. Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures. In *the International Conference on Automated Planning and Scheduling*, 324–332.

Farinelli, A.; Contini, A.; and Zorzi, D. 2020. Decentralized task assignment for multi-item pickup and delivery in logistic scenarios. In *the International Conference on Autonomous Agents and Multi-Agent Systems*, 1843–1845.

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

Huang, T.; Li, J.; Koenig, S.; and Dilkina, B. 2022. Anytime Multi-Agent Path Finding via Machine Learning-Guided Large Neighborhood Search. In *the AAAI Conference on Artificial Intelligence*, 9368–9376.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2021a. Anytime Multi-Agent Path Finding via Large Neighborhood Search. In *the International Joint Conference on Artificial Intelligence*, 4127–4135.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2022. MAPF-LNS2: Fast Repairing for Multi-Agent Path Finding via Large Neighborhood Search. In *the AAAI Conference on Artificial Intelligence*, 10256–10265.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. K. S.; and Koenig, S. 2021b. Lifelong multi-agent path finding in large-scale warehouses. In *the AAAI Conference on Artificial Intelligence*, 11272–11281.

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

Liu, M.; Ma, H.; Li, J.; and Koenig, S. 2019. Task and path planning for multi-agent pickup and delivery. In *the International Joint Conference on Autonomous Agents and Multiagent Systems*, 1152–1160.

Ma, H.; Hönig, W.; Kumar, T. K. S.; Ayanian, N.; and Koenig, S. 2019. Lifelong path planning with kinematic constraints for multi-agent pickup and delivery. In *the AAAI Conference on Artificial Intelligence*, 7651–7658.

Ma, H.; Li, J.; Kumar, T. K. S.; and Koenig, S. 2017. Lifelong multi-agent path finding for online pickup and delivery tasks. In *the International Conference on Autonomous Agents and Multi-Agent Systems*, 837–845.

Ma, H.; Wagner, G.; Felner, A.; Li, J.; Kumar, T.; and Koenig, S. 2018. Multi-agent path finding with deadlines. In *the International Joint Conference on Artificial Intelligence*, 417–423.

Moon, M. 2013. 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. Accessed: 2021-11-13.

Ren, Z.; Rathinam, S.; and Choset, H. 2021. MS*: A new exact algorithm for multi-agent simultaneous multi-goal sequencing and path finding. *arXiv preprint arXiv:2103.09979*.

Ropke, S.; and Pisinger, D. 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4): 455–472.

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

Smith, S. L.; and Imeson, F. 2017. GLNS: An effective large neighborhood search heuristic for the generalized traveling salesman problem. *Computers & Operations Research*, 87: 1–19.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Boyarski, E.; and Bartak, R. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *the International Symposium on Combinatorial Search*, 151–159.

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

Wen, X.; Xu, Y.; and Zhang, H. 2012. Online traveling salesman problem with deadline and advanced information. *Computers & Industrial Engineering*, 63(4): 1048–1053.

Wu, X.; Liu, Y.; Tang, X.; Cai, W.; Bai, F.; Khonstantine, G.; and Zhao, G. 2021. Multi-agent pickup and delivery with task deadlines. In *the International Symposium on Combinatorial Search*, 206–208.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2008. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 29(1): 9–9.