Solving Real-Time Distribution of Pickup and Delivery Tasks with Multiple Robotic Agents

Heder Soares Bernardino, Alex Borges Vieira, José Ronaldo Mouro

https://doi.org/10.1590/SciELOPreprints.13912

Submitted on: 2025-10-30
Posted on: 2025-11-10 (version 1)
(YYYY-MM-DD)

SOLVING REAL-TIME DISTRIBUTION OF PICKUP AND DELIVERY TASKS WITH MULTIPLE ROBOTIC AGENTS

José Ronaldo Mouro[1], Alex Borges Vieira[2], and Heder Soares Bernardino[3*]

*Corresponding author
[1]Universidade Federal de Juiz de Fora, Juiz de Fora, MG, BRAZIL / jronaldomouro@gmail.com / https://orcid.org/0009-0002-8571-3167
[2]Universidade Federal de Juiz de Fora, Juiz de Fora, MG, BRAZIL / alex.borges@ufjf.br / https://orcid.org/0000-0003-0821-126X
[3]Universidade Federal de Juiz de Fora, Juiz de Fora, MG, BRAZIL / heder.bernardino@ufjf.br / https://orcid.org/0000-0003-2012-7802

ABSTRACT. The advancement of automation technology has driven productivity and quality in industrial and logistics processes, reducing operational costs. In automated environments, such as warehouses and ports, fleets of robots continuously move loads without human intervention. The Multi-agent Pickup and Delivery (MAPD) problem involves multiple agents that attend to a continuous flow of pickup and delivery tasks in a known environment. Tasks arrive and are assigned to idle agents, which must move –free from collisions– from their current positions to the pickup location and then to the delivery location. Considering the combinatorial nature of the problem and the need for real-time responses, we propose three scalable heuristics: Nearest Pickup (NP), Threshold Task Path (TTP), and Split Delivery Task (SDT). The proposed heuristics were tested on instances from the literature with up to 500 agents and 1000 tasks, and compared to state-of-the-art approaches. NP performed better in terms of computational time, while TTP and SDT concerning the path quality. The use of path quality goals in TTP and SDT resulted in improvements of 3% to 10% compared to NP, with an increase in computational cost of 10% to 500%, respectively.

Keywords: multi-agent, real-time, heuristic

1 Introduction

Multi-agent systems (MAS) are essential components for automated material transport in large-scale environments such as warehouses, distribution centers, offices, and airports (Wurman et al., 2008; Boysen et al., 2019). In these domains, a large number of agents must continuously coordinate to execute transport tasks, optimizing resource use under demanding and dynamic conditions. The scale of these problems requires that high-quality solutions be computed rapidly to prevent operational bottlenecks. Indeed, a key constraint in real-time systems is computational agility, with task scheduling and path planning often subject to tight deadlines, exemplified by the one-second limit considered by Ma et al. (2017).

The Multi-Agent Pickup and Delivery (MAPD) problem, a lifelong extension of the Multi-Agent Pathfinding (MAPF) problem (Stern et al., 2019), directly addresses this challenge. In MAPD, agents operate collaboratively in a shared environment, managing a continuous flow of tasks—each involving a pickup and a delivery location—while avoiding collisions with other agents and obstacles (Lau and Sengupta, 2022). Solving MAPD encompasses two interdependent subproblems: task scheduling (deciding which agent will perform which task) and collision-free path generation. The task scheduling component is known for its high computational complexity (being NP-hard, as it generalizes problems like the Vehicle Routing Problem) (Ma et al., 2017; Goel et al., 2019), rendering optimal solutions intractable for large-scale instances.

The existing literature presents a central dilemma: methods that pursue optimality, typically through coupled approaches (solving scheduling and pathfinding simultaneously), are computationally expensive and do not scale to real-time scenarios (Goel et al., 2019; Chen et al., 2021). Conversely, decentralized methods and greedy heuristics, such as Token Passing (TP) (Ma et al., 2017), are scalable but can lead to significantly suboptimal solutions due to being "myopic"—that is, they make decisions based on local or incomplete information. Many simple centralized heuristics, while fast, also suffer from this myopia by using simplistic cost metrics (e.g., Euclidean distance), which are poor indicators of the true travel cost in congested environments (Chen et al., 2024). This work explores an alternative that seeks a balance between solution quality and computational speed: centralized, yet decoupled algorithms designed for real-time MAPD applications.

We approach the problem sequentially, processing task scheduling and path calculation for idle agents within defined time intervals. We propose three task scheduling heuristics of increasing complexity. The first, Nearest Pickup (NP), serves as a baseline that prioritizes tasks based on agent proximity to a central point of pending tasks. The other two introduce a novel path quality metric, which relates the actual path cost (considering the environment) to an ideal path (without obstacles or other agents), enabling an assignment criterion based on a minimum quality threshold. These heuristics are: Threshold Task Path (TTP), which extends NP by requiring tasks to meet minimum quality thresholds for both pickup and delivery paths; and Split Delivery Task (SDT), which enhances TTP by allowing task fractionation based on delivery path quality—a novel approach in MAPD literature that typically treats tasks as atomic. The proposals were evaluated and compared against the approaches introduced by Ma et al. (2017). To ensure a robust and direct comparison, the experiments were conducted on the same benchmark instances used in their work. The study also investigated the applicability and efficiency of the minimum path quality thresholds used in TTP and SDT concerning makespan and service time metrics in real-time contexts.

2 Problem Definition

The Multi-Agent Pickup and Delirery (MAPD) problem consists of a set $\mathcal{A} = a_1, a_2...a_m$ of m agents and a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ represents the locations of the agents and $\mathcal{E}$ represents the connections between these locations. Let $l_i(t) \in \mathcal{V}$ denote the location of agent a_i at discrete time step t. Agent a_i starts at location $l_i(0)$. At each time step

t, an agent can either stay at its current location $l_i(t)$ or move to an adjacent location, i.e., $l_i(t+1) = l_i(t)$ ou $(l_i(t), l_i(t+1)) \in \mathcal{E}$.

Two agents cannot occupy the same location at the same time step t, i.e., $l_i(t) \neq (l_{i'}(t))$. Two agents cannot move through the same edge at the same time step t, i.e., $l_i(t) \neq (l_{i'}(t+1))$ ou $l_{i'}(t) \neq (l_i(t+1))$. A path $p_i = [l_i(s), l_i(s+1), ..., l_i(g))]$ is a sequence that represents the movement of agent a_i, starting at time s and ending at time g. The length of a path is denoted by $|p|$. The concatenation of two paths is denoted by $p_i \oplus p_i'$. Two paths collide if two agents walking through them collide. The quality of a path p_i is expressed by $Q(p_i) = |IP(p_i)|/|p_i|$, where $0 < Q(p_i) \leq 1$, and $IP(p_i)$ is the ideal path calculated for an agent between $p_i[l_i(s)]$ and $p_i[l_i(g)]$, ignoring the presence of other agents in the environment.

A Minimum Path Quality Threshold (MPQT) m can be established for the paths p_i, such that a path is considered admissible if $Q(p_i) \geq m$.

Consider a set of tasks $\mathcal{T}$ that contains unexecuted tasks. At each time step, the system adds new tasks to the set. Each task $\tau_j \in \mathcal{T}$ is characterized by a pickup location $s_j \in V$ and a delivery location $g_j \in V$, called task endpoints. An agent is considered free if it is not currently executing a task, i.e., it does not have a task assigned to it or has already visited the pickup location. A free agent can be assigned to a task $\tau_j \in \mathcal{T}$ and must move to the pickup location s_j. After reaching s_j, the agent starts executing the task by moving to g_j, and the task is removed from the set $\mathcal{T}$. After reaching g_j, the task is considered executed, and the agent is free to be assigned to a new task. Before reaching s_j, the agent can be reassigned to another task $\tau_j \in \mathcal{T}$. However, if the agent has already reached s_j, it must complete the task τ_j.

Not every instance of the MAPD problem is solvable, according to the author Ma et al. (2017). The minimum conditions are that agents must be able to stay at locations called endpoints without blocking the passage of other agents, there must be a path between each pair of endpoints, and there must be at least as many endpoints as there are agents that are not task endpoints (pickup and delivery locations of tasks). The objective of the MAPD problem is to complete the execution of tasks quickly.

3 Related Work

Solving the Multi-Agent Pickup and Delivery (MAPD) problem inherently involves two core challenges: Task Allocation (TA) and Multi-Agent Path Finding (MAPF). The literature can be broadly categorized by how it addresses the interdependence of these two subproblems, most notably through coupled or decoupled methods.

3.1 Coupled vs. Decoupled Approaches

Coupled (or integrated) approaches attempt to solve task allocation and path planning simultaneously, aiming for global optimality. In Chen et al. (2021), a new coupled method was proposed where task assignment choices are informed by actual delivery costs instead

of lower-bound estimates, using a marginal-cost assignment heuristic and a meta-heuristic improvement strategy based on Large Neighborhood Search. Their experiments showed difficulty in scaling to larger instances.

Methods seeking formal optimal solutions exemplify the complexity of coupled approaches. Branch-and-Cut-and-Price (BCP-MAPD) and Branch-and-Cut-and-Price Benders (BCPB-MAPD) were introduced by Goel et al. (2019) to find optimal solutions for MAPD using mathematical programming. While successful in small environments (with average run times around 2700 seconds), the authors demonstrated that finding optimality in MAPD is computationally expensive and challenging.

The computational intractability of optimal and coupled methods for real-time applications motivates the use of decoupled approaches, which address scheduling and planning in sequential steps. While this decomposition may sacrifice global optimality, it allows for the development of much faster heuristics. This work adopts a centralized and decoupled approach, focusing on the development of high-speed scheduling heuristics.

3.2 Task Allocation Strategies in MAPD

Given a decoupled framework, the core challenge becomes the efficient allocation of tasks. These strategies can be decentralized or centralized.

Decentralized approaches distribute the decision-making process among the agents. Token Passing (TP), proposed by Ma et al. (2017), is a widely-used decentralized on-line algorithm. In TP, agents pass a "token" to greedily assign the nearest task and plan a path. Although its extension, Token Passing with Task Swapping (TPTS), achieved better performance metrics (makespan and service time), only TP could scale to larger problems, making it a common baseline for real-life examples. The primary drawback of TP is its myopic nature, which can lead to globally inefficient assignments.

Centralized approaches, conversely, attempt to optimize assignments from a global perspective. The simplest form of this is greedy heuristics, such as "Nearest Agent" (NA) or "Nearest Task" (NT), which often serve as initial baselines in the literature (Gerkey and Matarić, 2004). Our NP heuristic is inspired by these approaches. However, a critical flaw of these simple heuristics is the use of spatial distance (e.g., Euclidean or Manhattan) as a proxy for travel cost. In congested environments, spatial distance is a poor predictor of actual travel time (Chen et al., 2024).

To overcome the limitations of greedy heuristics, other centralized approaches use meta-heuristics. For instance, Queiroz et al. (2024) proposed an integer-encoded genetic algorithm to solve the task allocation part of the MAPD, combined with known path planning algorithms (PP and ICBS). The authors, however, did not present detailed execution times, limiting the analysis of their real-time applicability.

Our proposals, TTP and SDT, are positioned as fast, centralized heuristics that directly address the myopia of simple distance metrics. By introducing a path quality metric, they make scheduling decisions based on a more realistic estimate of the path cost, considering the current state of the environment. Furthermore, most MAPD literature (Lau and Sengupta, 2022) treats tasks as atomic (indivisible). Our SDT proposal challenges this

assumption by introducing task fractionation as a viable strategy to improve real-time throughput.

3.3 Lifelong Path Planning

Once tasks are assigned, agents must compute collision-free paths. In a lifelong setting, these plans must be continuously updated as new tasks arrive and the environment changes.

The Rolling-Horizon Collision Resolution (RHCR) framework, proposed by Li et al. (2021), was developed to manage large-scale agent coordination in dynamic environments, which they define as a lifelong MAPF problem. RHCR works by breaking the problem into smaller, time-limited "Windowed MAPF" instances, allowing for periodic re-planning.

RHCR represents a state-of-the-art approach to re-planning. The fundamental building blocks for the pathfinding subproblem (MAPF) include prioritized methods, such as Prioritized Planning (PP) (Erdmann and Lozano-Pérez, 1987), and optimal, conflict-based methods like Conflict-Based Search (CBS) (Sharon et al., 2015). The choice of a path planner in a decoupled, real-time system is a trade-off between path optimality and computation speed. In our work, the path calculation component utilizes a Space-Time A* (STA*) planner (Silver, 2005). This planner finds collision-free paths by treating the planned routes of other agents as dynamic obstacles in a shared reservation table, which provides an effective balance for the rapid evaluation required by our scheduling heuristics.

4 Proposed Approaches

Three approaches for MAPD problems are proposed here. Although the algorithms share common general steps, they are distinguished by the use of heuristics to update agent paths during task execution. The agents' paths are obtained using A* proposed by Nilsson (1971) with the Manhattan distance heuristic function.

4.1 General Steps

Algorithm 1 outlines the general steps of the proposals. First, agents' paths are initialized with their starting locations (Line 3). Then, in each successive iteration, new tasks entering the system are added to the set of unexecuted tasks $\mathcal{T}$ (Line 5). Next, the ***sortAgents*** function (Line 6) returns a sorted list of all agents, establishing a precedence among them for updating their paths. The precedence criterion is defined by the agent's proximity to the centroid of pickup endpoints s_j of tasks in $\mathcal{T}$, using the Manhattan distance metric. If an agent isn't assigned to a task (Line 8) and pending tasks exist in $\mathcal{T}$ (Line 10), the heuristic defined as a parameter is executed (Line 11). Subsequently, the ***updateRestPath*** function (Line 12) updates a path for the agent to move to the

Algorithm 1 General Flow

```
1: Data:  A, 𝒯, heuristic(h), nTask, pQT, dQT
2: s ← 0
3: foreach a_i ∈ A do initializePath(a_i)
4: while true do
5:     for every new task t do 𝒯 ← 𝒯 ∪ {t}
6:     al ← sortAgents(𝒯, A)
7:     from first to last a_i ∈ al do
8:         if isAssignedTask(a_i) ≠ true then
9:             p ← ∅
10:            if 𝒯 ≠ ∅ then
11:                p ← npSdtTtp(𝒯, a_i, s, nTask, pQT, dQT, h)
12:                updateRestPath(p, a_i, s)
13:            end if
14:            updateTrivialPath(p, a_i, s); Path_{a_i} ← Path_{a_i} ⊕ p ;
15:        end if
16:    s ← s + 1
17: end while
```

nearest waiting endpoint where it can wait without obstructing pending tasks. The path is modified only if the agent is obstructing a task by remaining at its delivery endpoint. If the previously called heuristic function returns an empty path for a task, ***updateRestPath*** ensures this agent doesn't obstruct pending tasks. Finally, if the agent is unassigned and not obstructing any tasks, the ***updateTrivialPath*** function (Line 14) assigns a trivial path where the agent remains at its current location, and the obtained path (p) is joined to the agent's path $Path_{a_i}$.

4.2 Agent Path Update Flows for Task Execution

We propose three task assignment heuristics. The first is a greedy approach, assigning the task to the nearest pickup location. The second enhances this by requiring that both pickup and delivery paths satisfy a Minimum Path Quality Threshold (MPQT). The third, a path-splitting heuristic, is more flexible: it requires the MPQT only for the pickup path, allowing the delivery path to be split at an intermediate waypoint to meet the threshold. This strategy brings an object progressively closer to its final destination, generating a new sub-task for the remaining journey to avoid costlier displacements.

4.2.1 *Nearest Pickup (NP)*

Algorithm 2 describes NP for task assignment and path calculation, which involves choosing the task closest to the agent - specifically, the one whose task pickup endpoint is nearest to the agent's current location. The proximity measure between the task and the agent is the length of the ideal path between the agent's current location and the task's pickup location. These distances are known through precalculation since the environment is known. First, $nTask$ unexecuted tasks in $\mathcal{T}$ will be sorted into a list tl, prioritizing those closest to the agent (Line 2). Each task t_j from the list tl, following the priority order, will be checked for obstruction by other agents (Line 4). The ***isUnobstructed*** function checks if another agent is stationed at task t_j 's endpoints. If obstruction is verified, the task will be disregarded; otherwise, the ***getTaskPath*** function will be used

Algorithm 2 Nearest Pickup (NP)

```
1: Data: 𝒯, agent, step, nTask
2: tl ← sortTasks(𝒯, agent, nTask)
3: from first to last t_j ∈ tl do
4:     if isUnobstructed(t_j, agent) = true then
5:         p ← getTaskPath(t_j, agent, step)
6:         agent ⇐ t_j; 𝒯 ← 𝒯 − t_j;
7:         return p
8:     end if
9: return ∅
```

to return a path for task t_j (Line 5). Next, task t_j is assigned to the agent, removed from the set of unexecuted tasks $\mathcal{T}$ (Line 6), and the task path p is returned by the algorithm (Line 7). In Line 9, the algorithm returns an empty path when no task is assigned to the agent.

4.2.2 *Threshold Task Path (TTP)*

Algorithm 3 describes TTP for task assignment and path calculation, which involves choosing the task closest to the agent that meets a minimum quality threshold (MPQT) for pickup and delivery. If this isn't possible, the task with the best average MPQT performance may be chosen. As in Algorithm 2, the $nTask$ tasks closest to the agent will be sorted, and an ordered iteration over each task will be performed, checking for obstruction (Lines 3, 4, and 5). If a task t_j is unobstructed, its actual path p is calculated (Line 6). In Line 7, the ***acceptedPath*** function checks if path p meets the minimum quality thresholds pQT and dQT. If task t_j meets the thresholds, it will be assigned to the agent, removed from the set of unexecuted tasks $\mathcal{T}$, and p will be returned by the algorithm (Lines 8 and 9). In Line 11, the ***saveTaskPath*** function is responsible for storing the best task that didn't meet the quality goals. The comparison metric used in the ***saveTaskPath*** function is an average value between the quality of the pickup path and the delivery path. If the iterative search process for a task that meets the quality thresholds is unsuccessful, but there is at least one unobstructed task, the task $auxTask$ will be assigned to the agent, removed from the set of unexecuted tasks $\mathcal{T}$, and the path $auxPath$ will be returned (Lines 13, 14, and 15). Otherwise, the algorithm returns an empty path (Line 17).

4.2.3 *Split Delivery Task (SDT)*

As in Algorithm 2, the $nTask$ tasks closest to the agent will be sorted, and an ordered iteration over each task will be performed, checking for obstruction [Lines 3, 4, and 5]. If the task t_j is not obstructed, the ***getPickupTaskPath*** function calculates the pickup path pp [Line 6]. Next, the ***acceptedPickupPath*** function checks if pp meets the pickup MPQT $pckpQT$ [Line 7]. If the pickup path is accepted, the ***getDeliveryTaskPath*** function calculates the delivery path dp [Line 8]. Next, The procedures defined in lines [10 to 16] are intended to find a fractional path fp corresponding to path dp. The loop [line 10] iterates in reverse over the sequence of locations l_g of path dp in order to find a

Algorithm 3 Threshold Task Path (TTP)

```
1: Data: 𝒯, agent, step, nTask, pQT, dQT
2: auxTask ← null; auxPath ← null
3: tl ← sortTasks(𝒯, agent, nTask)
4: from first to last t_j ∈ tl do
5:     if isUnobstructed(t_j, agent) = true then
6:         p ← getTaskPath(t_j, agent, step)
7:         if acceptedPath(p, pQT, dQT) = true then
8:             agent ⇐ t_j; 𝒯 ← 𝒯 − t_j
9:             return p
10:        end if
11:        auxPath, auxTask ← saveTaskPath(p, t_j)
12:    end if
13: if auxTask ≠ null then
14:    agent ⇐ auxTask; 𝒯 ← 𝒯 − auxTask
15:    return auxPath
16: end if
17: return ∅
```

task endpoint along dp whose path between the pickup location and the endpoint meets delivery MPQT $dlvrQT$, using the function ***acceptedSplitPath*** [line 12]. Note that the fractional path can be path dp itself or another larger possible path. In addition to the quality threshold, the ***acceptedSplitPath*** function [line 12] can also check if location l_g is capable of receiving the fractionation based on other criteria. In the experiments conducted, the only criterion adopted was the quality threshold. If f_p is not empty [Line 17], the task t_j will be executed, removed from the set $\mathcal{T}$, and assigned to the agent, and the algorithm returns the concatenation of the pickup path pp and fp [Line 22]. If the sizes of paths fp and dp are different [Line 18], this means that fp is a partial path contained within dp by which the transport for task t_j will be split. A complementary task t'_j will be added to the set $\mathcal{T}$ at the time interval $step + |fp|$ [Line 19]. In line 25, The ***saveTaskPath*** function is responsible for storing the best task and path that did not meet the quality goals. The comparison metric used in the ***saveTaskPath*** function is an average value between the quality of the pickup path and the delivery path. If the iterative search process for a task that meets the quality thresholds is unsuccessful, but there is at least one unobstructed task, the task $auxTask$ will be assigned to the agent, removed from the set of unexecuted tasks $\mathcal{T}$, and the path $auxPath$ will be returned [Lines 27, 28 and 29]. Otherwise, the algorithm returns an empty path [Line 31].

5 Computational Experiments

In this section, we analyze the results obtained by NP, TTP, and SDT. We used an Intel Core i7-4790M 3.60 GHz desktop with 16 GB of RAM, and the same instances used by Ma et al. (2017), with two-dimensional warehouse environments and sets of tasks. Figure 1 illustrates one of these environments with 21×35 locations and 10 agents (black circles). Task sets vary in number of tasks and frequency (number of tasks added per time interval). Each experiment runs until all tasks in the set ($\mathcal{T}$) are complete.

The comparative results are shown using the metrics of makespan, service time, and run-time. Makespan is the total number of time intervals required to execute all tasks; service

Algorithm 4 Split Delivery Task (SDT)

```
1: Data: 𝒯, agent, step, nTask, pckpQT, dlvrQT
2: auxTask ← null; auxPath ← null
3: tl ← sortTasks(𝒯, agent, nTask)
4: from first to last t_j ∈ tl do
5:     if isUnobstructed(t_j, agent) = true then
6:         pp ← getPickupTaskPath(t_j, agent, step)
7:         if acceptedPickupPath(pp, pckpQT) = true then
8:             dp ← getDeliveryTaskPath(t_j, agent, pp)
9:             fp ← ∅
10:            from last to first + 1 l_g ∈ dp do
11:                if isTaskEndpoint(l_g) = true then
12:                    if acceptedPath(dp[..., l_g], dlvrQT) = true then
13:                        fp ← dp[..., l_g]
14:                        break
15:                    end if
16:                end if
17:            if fp ≠ ∅ then
18:                if |fp| ≠ |dp| then
19:                    𝒯 ← 𝒯 ∪ {t'_j(fp_s, dp_g)} at (step + |pf|)
20:                end if
21:                𝒯 ← 𝒯 − t_j; agent ⇐ t_j
22:                return pp ⊕ fp
23:            end if
24:        end if
25:        auxPath, auxTask ← saveTaskPath(pp ⊕ dp, t_j)
26:    end if
27: if auxTask ≠ null then
28:    agent ⇐ auxTask; 𝒯 ← 𝒯 − auxTask
29:    return auxPath
30: end if
31: return ∅
```

time is the average time in time intervals for a task to be completed after its entry into the system; and runtime is the average actual computational time, in milliseconds, spent by the system per time interval. When comparing with the approaches from Ma et al. (2017), the runtime values of the proposals were multiplied by 1.74 to balance the difference in processor performance of the computational systems involved. This is the ratio between the Average CPU Mark (Single Thread Rating) of the CPU used here (2229) and that used by Ma et al. (2017) (1288) according to `https://www.cpubenchmark.net`.

5.1 Analysis of the use of quality goals

First, the heuristics were tested in a small environment (21×35 locations) with numbers of agents from {10, 20, 30, 40, 50}, 500 tasks with frequency from {0.2, 0.5, 1, 2, 5, 10}, totaling 30 scenarios. TTP and SDT were tested by combining 20 minimum quality path thresholds (0, 0.5, 1.0, ..., 0.9, 0.95). The values obtained were normalized with respect to the NP results and grouped by the average for each minimum quality path threshold. Figure 2 shows curves with regular growth and decay trends that rule out the occurrence of randomness and, thus, the judicious choice of these minimum quality path thresholds can direct the results to the objectives of interest.

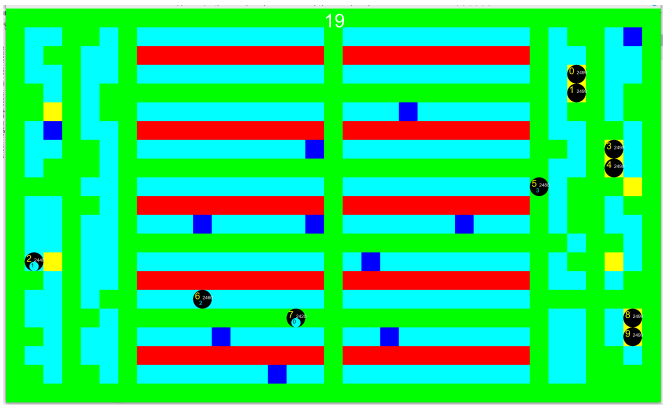

Figure 1: Example of environment used in the experiments. Light blue squares represent task endpoints, yellow squares represent agents' initial endpoints, red squares represent locations with obstacles, dark blue squares represent unused task endpoints, and green squares represent agent transit locations.

5.2 Comparative Results

Based on the service time values in Figure 2, a pair of minimum quality path thresholds were chosen for TTP (0.50, 0.75) and one for SDT (0.50, 0.50). The results of these heuristics, parameterized as such, along with NP, are shown in Table 1. The graphs in Figure 3 show the performance of the three proposed heuristics in relation to TP, TPTS, and Central. Through normalization of each result obtained by the heuristics for a pair of environment and task instances in relation to TP by Ma et al. (2017), grouping these normalized values by the average and for a 95% confidence interval, the results in makespan and service time of SDT and TTP do not differ significantly to those of TPTS by Ma et al. (2017), and they present a significantly better runtime value.

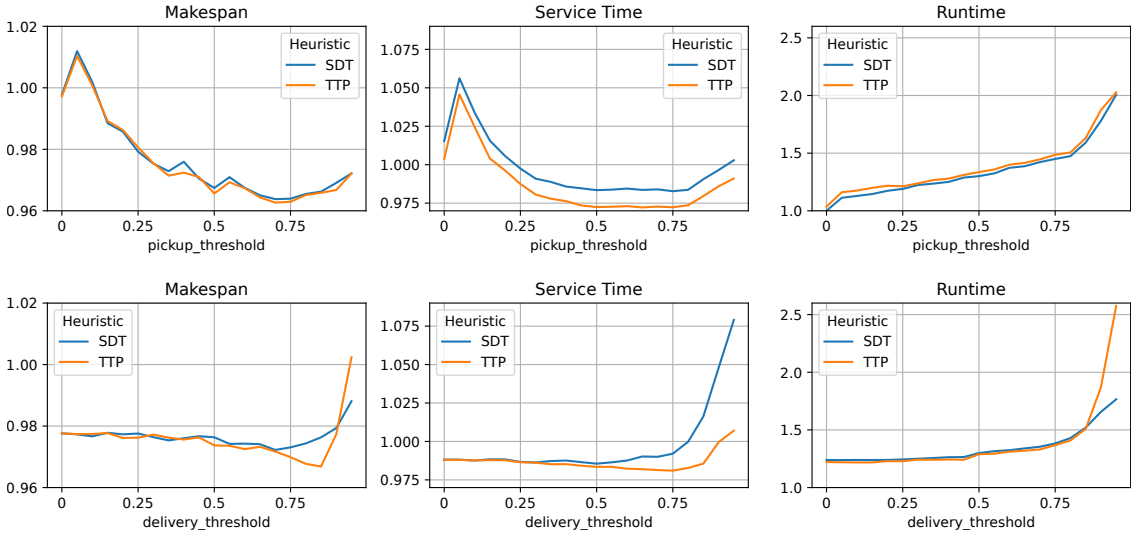

Figure 2: Impact on the performance of SDT and TTP of using collection and delivery thresholds compared to NP.

Table 1: Results of the NP, SDT, and TTP heuristics for the small-scale instance set. The best results are highlighted. The results for the TP and TPTS heuristics were extracted from Ma et al. (2017) and adopted as a baseline.

Fr.	Ag.	NP(500)			SDT(0.5:0.5:500)			TTP(0.5:0.75:500)			TP by Ma et al. (2017)			TPTS by Ma et al. (2017)		
		m.span	s.time	r.time(ms)	m.span	s.time	r.time(ms)	m.span	s.time	r.time(ms)	m.span	s.time	r.time(ms)	m.span	s.time	r.time(ms)
0.2	10	2532	29.89	0.02	2532	30.04	0.02	2532	30.04	0.02	2532	38.54	0.13	2532	29.33	1.86
	20	2532	26.58	0.04	2532	26.58	0.03	2532	26.58	0.03	2540	39.77	0.26	2520	25.36	9.82
	30	2532	25.40	0.03	2532	25.40	0.03	2532	25.40	0.03	2546	38.71	0.25	2527	23.88	21.57
	40	2532	25.77	0.04	2532	25.77	0.04	2532	25.77	0.04	2540	38.88	0.24	2524	23.50	27.49
	50	2532	25.63	0.05	2532	25.64	0.05	2532	25.63	0.04	2540	40.03	0.32	2524	23.11	47.33
0.5	10	1319	135.56	0.05	1249	120.58	0.06	1256	122.31	0.05	1309	132.79	0.24	1274	131.15	0.33
	20	1070	38.00	0.11	1064	39.93	0.11	1068	40.69	0.10	1094	42.69	1.16	1038	30.74	12.39
	30	1070	33.86	0.09	1072	34.95	0.11	1071	34.94	0.10	1069	43.97	1.51	1035	27.14	34.04
	40	1074	33.09	0.11	1073	32.63	0.10	1073	32.63	0.09	1090	43.01	0.70	1038	25.98	19.85
	50	1074	33.00	0.14	1073	32.99	0.10	1073	32.99	0.10	1083	43.66	1.36	1036	25.22	71.48
1	10	1207	306.82	0.10	1173	298.37	0.12	1161	298.07	0.13	1198	311.78	0.20	1182	301.03	0.37
	20	714	97.70	0.14	671	86.90	0.19	679	84.87	0.20	757	95.98	1.03	706	88.25	2.80
	30	604	45.53	0.16	604	43.91	0.23	625	43.27	0.25	607	53.80	2.81	561	42.84	8.45
	40	600	44.91	0.27	594	45.05	0.39	609	46.45	0.40	624	48.80	2.14	563	31.99	39.54
	50	595	43.18	0.48	594	47.59	0.38	587	46.54	0.52	597	49.14	3.76	554	30.27	128.13
2	10	1175	420.55	0.17	1164	414.53	0.19	1170	412.89	0.19	1167	407.62	0.19	1168	407.24	0.37
	20	665	178.22	0.25	646	177.24	0.29	637	172.89	0.28	683	190.76	0.96	667	181.03	2.38
	30	553	121.86	0.30	463	102.25	0.42	471	97.45	0.36	529	114.39	2.31	496	102.69	8.39
	40	467	80.12	0.27	416	71.94	0.61	399	73.75	0.64	464	95.32	3.43	425	72.59	7.32
	50	417	68.91	0.53	374	62.14	0.77	379	60.04	0.79	432	75.63	6.28	383	58.06	126.47
5	10	1196	486.40	0.20	1160	473.49	0.25	1143	473.00	0.24	1162	473.78	0.20	1165	473.18	0.41
	20	638	252.37	0.40	652	243.17	0.52	658	241.08	0.47	655	247.08	1.02	645	238.02	1.68
	30	485	173.63	0.47	493	166.81	0.58	448	163.48	0.77	478	170.78	2.22	474	167.66	8.58
	40	398	137.38	0.53	393	129.28	0.94	373	128.91	1.08	418	155.33	4.15	396	131.36	12.31
	50	373	120.41	0.75	349	112.67	1.41	316	107.74	1.51	395	124.59	5.92	343	104.86	59.64
10	10	1158	509.56	0.23	1158	499.63	0.26	1148	500.60	0.27	1163	495.93	0.22	1172	505.26	0.40
	20	646	270.69	0.49	654	264.60	0.51	624	260.19	0.52	643	275.24	1.09	645	258.36	1.87
	30	481	197.43	0.59	451	189.34	0.90	459	189.84	0.93	526	192.01	1.98	491	198.30	10.82
	40	478	159.21	0.64	415	150.09	1.01	406	148.19	1.14	407	154.63	1.65	389	152.49	12.62
	50	413	135.92	0.88	325	125.05	1.24	316	127.15	1.51	333	131.42	5.62	319	126.96	25.32

Table 2: Results of the NP, SDT, and TTP heuristics for the large-scale instance set. The best results are highlighted. The results for the TP heuristic were extracted from Ma et al. (2017) and adopted as a baseline.

Ag.	NP(50)		BTT(0.5:0.5:50)		TTP(0.5:0.75:50)		TP by Ma et al. (2017)	
	s.time	r.time(ms)	s.time	r.time(ms)	s.time	r.time(ms)	s.time	r.time(ms)
100	416.53	3.10	391.53	5.77	389.59	5.20	463.25	90.83
200	289.87	14.62	256.91	54.35	258.30	85.72	330.19	538.22
300	281.81	82.14	226.72	258.55	226.40	256.50	301.97	1854.44
400	244.10	70.10	220.01	488.01	222.87	615.96	289.08	3881.11
500	266.58	145.68	238.19	1188.78	230.19	1094.05	284.24	6121.6

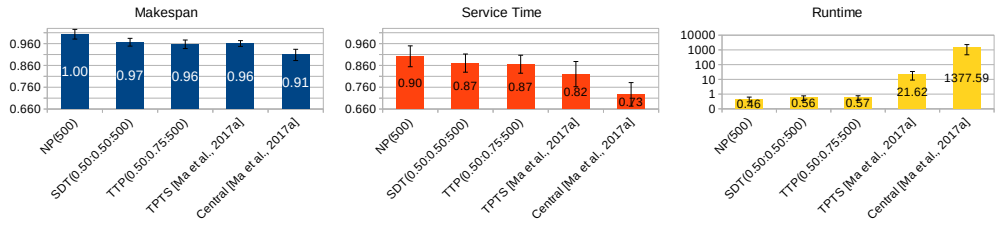

Figure 3: Comparative performance at average (CI:95%) rate of the results achieved by the heuristics in relation to TP [Ma et al., 2017a] on small instances.

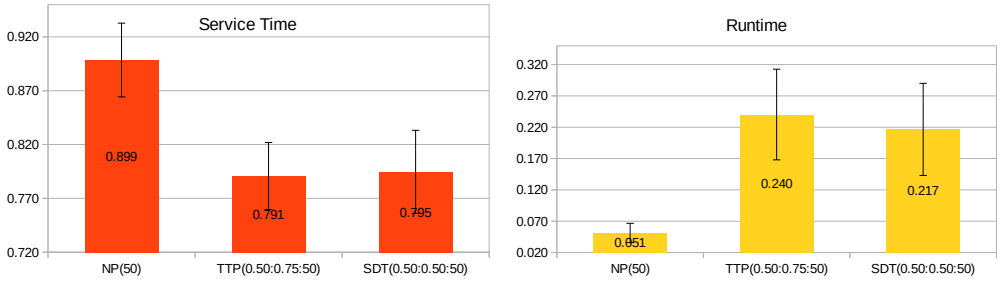

Figure 4: Comparative performance at average (CI:95%) rate of the results achieved by the heuristics in relation to TP [Ma et al., 2017a] on large instances.

The heuristics were also tested in a large environment (81×101 locations) with the number of agents from $\{100, 200, 300, 400, 500\}$, and 1000 tasks with frequency of 50, totaling 5 scenarios. A maximum task search threshold of 50 tasks was used, and the same minimum quality path thresholds chosen for the experiments in the small environment were used. The results can be verified in Table 2. The graphs in Figure 4 show the comparative performance of the three proposed heuristics normalized. In relation to the TP Ma et al. (2017) approach, the performance in service time of the TTP and SDT heuristics is significantly better than. . It can be seen that the average performance in service time of the TTP and SDT heuristics is significantly better than the performance of the NP heuristic, and that the three proposed heuristics outperform the TP Ma et al. (2017).

6 Conclusion

This article introduced three novel, deterministic, and centralized heuristics (NP, TTP and SDT) designed to address the MAPD problem in real-time systems. They efficiently manage the continuous flow of pickup and delivery tasks, ensuring collision-free agent movements within strict time constraints. Our experiments demonstrated that all three heuristics scale effectively to larger problem sizes, consistently achieving processing times under one second, crucial for real-time applications. Notably, the TTP and SDT heuris-

tics, which incorporate path quality as a task scheduling criterion, exhibited superior average performance in terms of makespan and service time, underscoring the benefits of integrating path quality considerations into multi-agent task assignment.

The TTP and SDT heuristics establish path quality criteria for selecting which tasks to execute. This leads to an increase in path calculations and, consequently, an increase in runtime. The *ntask* parameter passed to the approaches is used to reduce the computational effort involved in searching for the ideal task. A higher value for this parameter allows for a larger number of chosen tasks that meet the path quality criteria, at the expense of a higher computational cost.

Building on this work, we will now study the behavior of the proposed heuristics in a scenario with an energy constraint that requires agents to recharge in order to continue executing tasks.

7 Contribuição de Autoria

José Ronaldo Mouro: Investigação, Software, Redação – Rascunho Original. Heder Soares Bernardino: Conceituação, Análise Formal, Metodologia, Supervisão, Validação, Redação – Revisão e Edição. Alex Borges Vieira: Conceituação, Análise Formal, Metodologia, Supervisão, Validação, Redação – Revisão e Edição.

8 Conflito de Interesses

Os autores declaram não possuir interesses financeiros ou pessoais concorrentes que possam ser percebidos como influenciando a imparcialidade dos resultados reportados neste artigo.

9 Disponibilidade de Dados de Pesquisa

Os dados e o código-fonte que suportam os resultados deste estudo estão disponíveis publicamente no repositório GitHub em `https://github.com/jrmouro/MAPD`.

References

Boysen, N., de Koster, R., and Weidinger, F. (2019). Warehousing in the e-commerce era: A survey. *European Journal of Operational Research*, 277(2):396–411.

Chen, Y., Yu, Y., Ma, L., Huang, Z., Xu, L., and Zhang, K. (2021). Integrated task assignment and path planning for capacitated multi-agent pickup and delivery. In *AAAI Conf. on Artificial Intelligence (AAAI)*, volume 35, pages 13320–13327.

Chen, Z., Harabor, D., Li, J., and Stuckey, P. J. (2024). Traffic flow optimisation for lifelong multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 367–375.

Erdmann, M. and Lozano-Pérez, T. (1987). On multiple moving objects. *Algorithmica*, 2(1-4):477–521.

Gerkey, B. P. and Matarić, M. J. (2004). A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954.

Goel, A., Otten, N., Ma, H., and Koenig, S. (2019). Optimal multi-agent pickup and delivery using branch-and-cut-and-price algorithms. *Journal of Artificial Intelligence Research (JAIR)*, 66:725–776.

Lau, T. T.-K. and Sengupta, B. (2022). The multi-agent pickup and delivery problem: MAPF, MARL and its warehouse applications. *arXiv preprint arXiv:2203.07092*.

Li, J., Tinka, A., Kiesel, S., Durham, J., Kumar, A., and Koenig, S. (2021). Lifelong multi-agent path finding in large-scale warehouses. In *Intl. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 555–564.

Ma, H., Li, J., Kumar, T. K. S., and Koenig, S. (2017). Lifelong multi-agent path finding for online pickup and delivery tasks. In *Intl.Conf. on Auton. Agents and Multiagent Systems (AAMAS)*, pages 837–845.

Nilsson, N. J. (1971). A formal basis for the heuristic determination of minimum cost paths. In *IFIP Congress*, volume 71, pages 1–8. North-Holland Publishing Company.

Queiroz, A. C., Vieira, A., and Bernardino, H. (2024). Solving multi-agent pickup and delivery problems using multiobjective optimization. *Journal of Intelligent & Robotic Systems*, 114(1):1–19.

Sharon, G., Stern, R., Felner, A., and Sturtevant, N. R. (2015). Conflict-based search for optimal multi-agent pathfinding. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 1804–1810.

Silver, D. (2005). Cooperative pathfinding. In *Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 117–122. AAAI Press.

Stern, R., Sturtevant, N. R., Felner, A., Koenig, S., Ma, H., Walker, T. T., Li, J., Atzmon, D., Cohen, L., Kumar, T. K. S., Barták, R., and Boyarski, E. (2019). Multi-agent path finding: Definitions, variants, and benchmarks. In *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, pages 151–159.

Wurman, P. R., D'Andrea, R., and Mountz, M. (2008). Coordinating hundreds of cooperative, autonomous vehicles in distribution centers. *AI Magazine*, 29(1):9–20.

This preprint was submitted under the following conditions:

- The authors declare that the necessary Terms of Free and Informed Consent of participants or patients in the research were obtained and are described in the manuscript, when applicable.
- The authors declare that the preparation of the manuscript followed the ethical norms of scientific communication.
- The authors declare that they are aware that they are solely responsible for the content of the preprint and that the deposit in SciELO Preprints does not mean any commitment on the part of SciELO, except its preservation and dissemination.
- The authors declare that the data, applications, and other content underlying the manuscript are referenced.
- The deposited manuscript is in PDF format.
- The authors declare that the research that originated the manuscript followed good ethical practices and that the necessary approvals from research ethics committees, when applicable, are described in the manuscript.
- The authors declare that once a manuscript is posted on the SciELO Preprints server, it can only be taken down on request to the SciELO Preprints server Editorial Secretariat, who will post a retraction notice in its place.
- The authors agree that the approved manuscript will be made available under a Creative Commons CC-BY license.
- The submitting author declares that the contributions of all authors and conflict of interest statement are included explicitly and in specific sections of the manuscript.
- The authors declare that the manuscript was not deposited and/or previously made available on another preprint server or published by a journal.
- If the manuscript is being reviewed or being prepared for publishing but not yet published by a journal, the authors declare that they have received authorization from the journal to make this deposit.
- The submitting author declares that all authors of the manuscript agree with the submission to SciELO Preprints.