Multi-Goal Multi-Agent Pickup and Delivery

by

Qinghong Xu

M.Sc., Simon Fraser University, 2019
B.Sc., Xiamen University, 2017

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

© Qinghong Xu 2022
SIMON FRASER UNIVERSITY
Spring 2022

Copyright in this work is held by the author. Please ensure that any reproduction or re-use is done in accordance with the relevant national copyright legislation.

Declaration of Committee

Name:	**Qinghong Xu**
Degree:	**Master of Science**
Thesis title:	**Multi-Goal Multi-Agent Pickup and Delivery**
Committee:	**Chair:** Angelica Lim Assistant Professor, Computing Science
	Hang Ma Supervisor Assistant Professor, Computing Science
	Mo Chen Committee Member Assistant Professor, Computing Science
	Satish Kumar Thittamaranahalli External Examiner Research Assistant Professor, Computer Science

Abstract

In this work, we consider the Multi-Agent Pickup and Delivery (MAPD) problem, where agents constantly engage with new tasks and need to plan collision-free paths to execute them. To execute a task, an agent needs to visit a pair of goal locations, namely the pickup location and the delivery location. To solve an MAPD instance, we need to decide which agent executes which tasks (task-assignment), and plan collision-free paths for agents to execute these tasks (path-finding). Existing MAPD methods either assign an agent's next task only, which can lead to bad schedules of the entire task set, or plan agents' paths segment by segment, which can lead to larger path costs. Therefore, we propose a method that improves the state-of-the-art MAPD methods in both aspects. Our method assigns a sequence of tasks to each agent using the anytime algorithm Large Neighborhood Search (LNS), and plans paths through a sequence of goal locations using the Multi-Agent Path Finding (MAPF) algorithm Priority-Based Search (PBS). Specifically, two variants of this method are proposed: LNS-PBS and LNS-wPBS. Theoretically, we prove that LNS-PBS is complete for well-formed MAPD, a realistic subclass of MAPD instances. Empirically, LNS-PBS produces better solutions than the existing complete method CENTRAL. The second variant LNS-wPBS is more efficient and stable (but has no completeness guarantee). Empirically, LNS-wPBS can scale to thousands of agents and thousands of tasks in a large warehouse, and produce better solutions than the existing scalable methods. Lastly, the proposed method applies to a more generalized variant of MAPD, the Multi-Goal MAPD (MG-MAPD) problem, where tasks can have a various number of goal locations.

Keywords: Multi-Robot Systems; Path Planning for Multiple Mobile Robots or Agents; Task and Motion Planning

Acknowledgements

First of all, I would like to thank my supervisor Prof. Hang Ma for offering me this great opportunity to explore a whole new field of research that is exciting and inspiring. I would like to thank him for offering me the freedom to discover my own research interest, and always being supportive, encouraging and patient whenever I'm faced with challenges.

I would like to thank my mentor Jiaoyang Li for her detailed comments and suggestions about my research. I would like to thank Prof. Satish Kumar Thittamaranahalli for providing profound insight and guidance on the project we have collaborated on. I also would like to thank him for dedicating time to examining this thesis and providing helpful comments. I would like to thank the other committee member Prof. Mo Chen and the committee chair Prof. Angelica Lim, for always kindly sharing their knowledge and experience in our Robotics meetings, and for their helpful comments about my research.

I would like to thank our group members Dingyi Sun, Xinyi Zhong, Dingcheng Hu, Baiyu Li, Danoosh Chamani, Qiushi Lin and group visitor Yudong Luo. I enjoy every discussion we have had during and after the group meeting, and I'm always motivated by their curiosity and enthusiasm for both research and life.

I also would like to thank my fellow students Xinwei Gu, Jiaqi Tan, Yue Ruan, Miao Liu, Xiang Xu and Fuyang Zhang; my friends Jie Jian, Tian Chen, Sui Yi, Pengyu Liu and Yongjun He. I would like to thank them for making this journey full of joy and unforgettable memories.

Last but not least, I would like to thank my parents Weixiong Xu and Bixuan Li, for their unconditional love and support throughout my life.

Table of Contents

Declaration of Committee **ii**

Abstract **iii**

Acknowledgements **iv**

Table of Contents **v**

List of Tables **vi**

List of Figures **vii**

1 Introduction **1**

2 Related Work **4**
- 2.1 Multi-Agent Task Assignment 4
- 2.2 Multi-Agent Path Finding 5
- 2.3 Multi-Agent Pickup and Delivery 7

3 Problem Definition **9**

4 Method **11**
- 4.1 Large Neighborhood Search 11
- 4.2 Priority-Based Search 13
- 4.3 Completeness of LNS-PBS 14
- 4.4 LNS-wPBS 15
- 4.5 Look-Ahead Horizons 16

5 Experiments **17**

6 Conclusions **24**

Bibliography **25**

List of Tables

Table 1.1 Research related to MAPD. "lifelong" represents that agents can constantly engage with new tasks. "online" represents that the entire task set is not known at the beginning, and new tasks can enter the system at any time. "assign tasks (seq. task)" represents that a solver can assign one task sequence (vs. assign only one task) to each agent. "find paths (seq. goals)" represents that a solver can plan a path through a sequence of goal locations (vs. plan a path segment by segment). "complete (well-formed)" represents that a solver is complete for well-formed instances. 2

Table 5.1 Compare CENTRAL and LNS-PBS, RMCA and LNS-wPBS in a small warehouse environment 18

Table 5.2 Compare HBH+MLA*, RMCA, LNS-wPBS and LNS-PBS in a large warehouse environment with 1000 tasks and 100, 200, 300, 400, 500 agents. 50 tasks are released in one timestep. 19

Table 5.3 Compare HBH+MLA* and LNS-wPBS in a large warehouse environment with 1000 agents and 1000, 2000, 3000, 4000, 5000 tasks. 100 tasks are released in one timestep. 20

Table 5.4 Service time of MG-MAPD instances produced by three task assignment methods: Hungarian, Greedy-LNS and Hungarian-LNS 23

Table 5.5 Service time for different look-ahead horizons: "LA1" represents look one batch of tasks ahead. 23

List of Figures

Figure 4.1 A MG-MAPD instance at timestep 0 and 2 (left and right). Solid circles represent the existing tasks from previous timesteps and dash circles represent the new task released at the current timestep. . . . 16

Figure 5.1 Two simulated warehouse maps borrowed from [18]. Left: a small warehouse environment with 50 agents. Right: a large warehouse environment with 180 agents. The blue cells represent task endpoints, the orange cells represent non-task endpoints and the black cells are blocked. 18

Figure 5.2 Total computation time per timestep. The charts are generated for 40 agents and task frequency 1, 2, 5 and 10 instances (from left to right and top to bottom). 20

Figure 5.3 Task planning time and path finding time per timestep. The charts are generated for 40 agents and task frequency 0.2, 0.5, 1 and 2 instances (from (a) to (d)). The orange dots are hidden by the green dots since the task planning time of LNS-PBS and LNS-wPBS are very close. 21

Figure 5.4 Throughput and the number of tasks added per timestep. The charts are generated for 50 agents and task frequency 0.2, 0.5, 1, 2, 5 and 10 instances (from left to right and up to bottom). 22

Chapter 1

Introduction

In many real-world multi-robot systems, robots have to constantly attend to new tasks and plan collision-free paths to execute them. Examples include autonomous aircraft-towing vehicles [24], delivery drones [6], manufacturing robots [4], video game characters [23], and robots for automated warehouses [32]. Take an automated warehouse as an example, robots need to move inventory shelves to workstations, where human workers can pick products from the shelves to fulfill customers' requests. This problem has been studied as Multi-Agent Pickup and Delivery (MAPD) [21]. In MAPD, each task has a release time and a sequence of two goal locations, namely the pickup location and the delivery location. The pickup location indicates the storage location of an inventory shelf in a warehouse, and the delivery location indicates the location of a workstation that requests a product stored on the inventory shelf. To execute a task, a warehouse robot needs to first visit its pickup location at or after its release time, and then visit its delivery location.

To solve a MAPD instance, agents need to decide which tasks they are going to execute, and plan collision-free paths to execute them efficiently. Most existing solvers isolate the task-assignment part and the path-finding part, i.e., they first assign tasks to the agents based on an estimation of the actual path cost, and then use a Multi-Agent Path Finding (MAPF) [29] solver to plan actual paths for agents. Such decoupled solvers can be further categorized into: (1) assigning only one task to each agent and planning paths for the agents segment by segment [21], i.e., each call of the path planner only computes a plan that moves agents from their current locations to their next goal location; (2) assigning only one task to each agent, but planning a path through a sequence of goal locations for each agent [10]; (3) assigning a sequence of tasks to each agent and planning paths for the agents segment by segment [18, 5]. Assigning only one task to each agent can lead to a bad task assignment, since it only optimizes the cost of a subset of the existing tasks, and planning paths segment by segment can cause a larger path cost [10].

In addition, there is some work that focuses only on the path-finding part of the problem. For instance, Surynek [31] proposes two optimal Multi-Goal MAPF solvers HCBS and SMT-HCBS that can plan paths for a sequence of goal locations (where the ordering of the goal

Table 1.1: Research related to MAPD. "lifelong" represents that agents can constantly engage with new tasks. "online" represents that the entire task set is not known at the beginning, and new tasks can enter the system at any time. "assign tasks (seq. task)" represents that a solver can assign one task sequence (vs. assign only one task) to each agent. "find paths (seq. goals)" represents that a solver can plan a path through a sequence of goal locations (vs. plan a path segment by segment). "complete (well-formed)" represents that a solver is complete for well-formed instances.

	lifelong	online	assign tasks (seq. tasks)	find paths (seq. goals)	complete (well-formed)
CENTRAL [21]	✓	✓	✗	✗	✓
TA-Hybrid [18]	✓	✗	✓	✗	✓
HBH+MLA* [10]	✓	✓	✗	✓	✓
RMCA [5]	✓	✓	✓	✗	✗
(SMT-)HCBS [31]	✗	N/A	✗	✓	✓
WHCR [17]	✓	✓	✗	✓	✗
LNS-PBS	✓	✓	✓	✓	✓
LNS-wPBS	✓	✓	✓	✓	✗

locations is not specified). However, these solvers can only solve one-shot problems where each agent has only one task, and their scalability is limited. Li et al. [17] propose an efficient lifelong MAPF solver Rolling-Horizon Collision Resolution (WHCR) to plan paths for a sequence of goal locations. It uses a rolling-horizon framework to repeatedly call a Windowed MAPF solver that resolves collisions only for a few timesteps ahead. Such Windowed MAPF solvers run significantly faster than regular MAPF solvers but usually lose the completeness guarantee as they can lead to deadlocks due to their shortsightedness.

The main contributions of this work are as follows: We propose a decoupled method that assigns a sequence of tasks to each agent using the anytime algorithm Large Neighborhood Search (LNS), and plans paths through sequential goals using the MAPF algorithm Priority-Based Search (PBS). More specifically, two variants of this method are proposed: LNS-PBS and LNS-wPBS. The first variant focuses on completeness and effectiveness. PBS is in general not complete. Combined with the idea of "reserving dummy paths" from [18], we prove that LNS-PBS is complete on the well-formed MAPD instances, a realistic subclass of MAPD instances. Empirically, LNS-PBS can produce better solutions than the existing complete method CENTRAL. The second variant focuses on efficiency and stability. LNS-wPBS reserves the Windowed MAPF solver in WHCR, therefore the computation time of LNS-wPBS is controlled by the user-specified time window [17] and the anytime task assignment algorithm. Empirically, LNS-wPBS can produce better solutions than the existing scalable method HBH+MLA*, and can scale to thousands of agents and thousands of tasks in a large warehouse.

As a further contribution, we study two extensions of MAPD. Firstly, our method can extend to a generalized variant of MAPD, namely the Multi-Goal MAPD (MG-MAPD) problem, where tasks can have a various number of goal locations. This problem models the scenario where a warehouse robot may need to deliver an inventory shelf to multiple

workstations because they all request products stored on the same inventory shelf. We prove that LNS-PBS is complete for the MG-MAPD problem. Secondly, our method can handle different MAPD settings. This includes the online setting [21], i.e., the entire task set is not known at the beginning and new tasks can enter the system at any time, the offline setting [18], i.e., the entire task set is known at the beginning, and the semi-online setting (which has not been studied before), i.e., the system has partial knowledge about future tasks that it can plan ahead for. We compare the existing MAPD-related works against our methods in Table 1.1.

Chapter 2

Related Work

Existing MAPD solvers consist of two components: task-assignment and path-finding. In this section, we discuss several research that relates to them.

2.1 Multi-Agent Task Assignment

In the task-assignment part, agents need to decide which tasks they are going to execute. This problem is related to the Multi-Robot Task Allocation (MRTA) literature. The problem of MRTA is: Given m robots, n tasks, and estimates of how well each robot can perform each task, we need to assign robots to tasks so that the overall expected performance is maximized [8].

Gerkey et al. [9] and Korsah et al. [15] provide a comprehensive taxonomy for this topic. According to Gerkey et al. [9], MRTA can be categorized into three axes: (1) single-task robots (ST) vs. multi-task robots (MT), (2) single-robot tasks (SR) vs. multi-robot tasks (MR), and (3) instantaneous assignment (IA) vs. time-extended assignment (TA). The first axis describes that each robot is capable of executing at most one task (vs. multiple tasks) at a time. The second axis describes that each task requires exactly one (vs. multiple robots) to execute it. In the third axis, IA means that the available information only allows an instantaneous allocation of tasks to robots, while TA means that more information (e.g., a model of how tasks will arrive) is available, which allows for planning for future allocation. Hungarian [16] is a combinatorial optimization algorithm that finds the maximum-weight matching in a bipartite graph with polynomial time; it finds the optimal allocation for the ST-SR-IA problem in $O(mn^2)$ time. The ST-SR-TA problem is to find a time-extended schedule of tasks for each agent, and this problem is NP-hard to solve optimally [9].

Consider the online MAPD, where the entire task set is not known at the beginning and new tasks can enter the system at any time [21]. How to assign tasks can be categorized as an online variant of the ST-SR-IA problem, i.e., the robot-task performance is revealed only when the task is released. This is known as an online assignment problem [14]. Under this setting, the Hungarian method can repeatedly find an optimal task allocation for new

tasks [21], or new tasks can be assigned to the most fit robot that is currently available in a greedy fashion [10]. If the entire task set is known at the beginning, i.e., offline MAPD [18], then the task assignment can be categorized as an ST-SR-TA problem. In this work, we also study a novel semi-online setting, i.e., the system has partial knowledge about future tasks that it can plan ahead for, and there are no theoretical studies about this setting as far as we know.

Some other related problems include the Traveling Salesman Problem (TSP), Vehicle Routing Problem (VRP) and Dial-a-Ride Problem (DARP) [1, 2]. For example, Liu et al. [18] formulate the task assignment problem as a TSP, and uses an anytime TSP solver LKH3 [12] to find one task sequence for each agent. The VRP is to find an optimal set of routes for a fleet of vehicles, such that all the customers are delivered to the same given location. Shaw [27] introduces a local search algorithm Large Neighborhood Search (LNS) to construct a customer schedule for VRP. The idea is to start with an initial schedule, and iteratively replace it with a better schedule. In every iteration, some customers are removed from the schedule based on a removal heuristic; these customers are then inserted back to the schedule with a simple greedy approach. An adaptive LNS heuristic is used in the Pickup and Delivery Problem with Time Windows [25] to construct a schedule such that the transport requests are fulfilled and the total travel distance is minimized. These classic problems are generally NP-hard to solve optimally, and therefore the existing MAPD solvers usually have limited scalability to work on large instances, e.g., thousands of tasks and thousands of robots.

This literature only considers how to assign tasks based on an estimate of the actual path cost (e.g., the cost of the shortest path without collision avoidance), in the following we will discuss how the actual paths are planned when multiple robots are present.

2.2 Multi-Agent Path Finding

The path-finding problem is related to the Multi-Agent Path Finding (MAPF) literature. The problem of MAPF is to find a set of collision-free paths, following which agents can move from their start locations to their pre-assigned goal locations. The quality of a MAPF solution is measured by the flowtime, i.e., the sum of the arrival times of all agents at their goal locations, or the makespan, i.e., the maximum of the arrival times of all agents at their goal locations. MAPF is NP-hard to solve optimally for both flowtime and makespan minimization [33, 22].

Many MAPF solvers exist, such as the complete and optimal solvers Conflict-Based Search (CBS) [26] and its improved variant Improved CBS (ICBS) [3], the prioritized solvers Cooperative A* (CA*) [28] and Cooperative Partial-Refinement A* [30]. The prioritized solvers are based on the idea of Prioritized Planning [7]: Given a total priority ordering, each agent computes its shortest path (in the order of priority) without colliding with

the paths of all higher priority agents. Prioritized Planning is very efficient, but a predefined total priority ordering can lead to a bad solution quality or result in the failure of finding solution for solvable MAPF instances. Priority-Based Search (PBS) [20] is a two-level prioritized algorithm that attempts to address this issue: The high level of PBS performs a depth-first search to construct a priority ordering and builds a priority tree (PT), and the low level of PBS calls space-time A* [28] to plan optimal paths for agents that are consistent with the priority ordering generated by the high level. Since PBS introduces a new ordered pair to the priority ordering of the child PT node whenever it splits a parent PT node, the depth of the PT is $O(M^2)$, where M is the number of agents and M^2 is the number of all possible ordered pairs [20].

A* [11] is a best-first search algorithm. The priority function of A* takes into account both the cost function, i.e., the cost of reaching the current state from the start state, and the heuristic function, i.e., the estimated cost of reaching the goal state from the current state. A* is guaranteed to return the cost-minimal path when the heuristic underestimates the real cost (i.e., the heuristic is admissible). For example, consider moving an agent from the start cell to the goal cell on a 2D four-neighbouring grid, one of the admissible heuristics is the Manhattan-distance heuristic.

When multiple agents are present, space-time A* [28] is a single-agent pathfinding solver that can plan a time-minimal path. Consider a 2D four-neighbouring grid world. Two agents collide if they are occupying the same cell at the same time step (called vertex collision), or they are moving to the cell of the other agent at the same time step (called edge collision). Space-time A* searches in the cell-time space, where the vertexes are pairs (x, t) of cell x and time step t. Vertex (x, t) has a directed edge to vertex $(x, t+1)$ if and only if the agent can wait at cell x at time step t, and vertex (x, t) has a directed edge to vertex $(y, t+1)$ if and only if the agent can move from cell x to cell $y \neq x$ from time step t to time step $t+1$. Space-time A* searches a collision-free path by removing the edges that will cause vertex or edge collisions with other agents.

Multi-Label A* (MLA*) [10] is an extension of space-time A* that can plan a time-minimal path for a pair of ordered goal locations. Traditional methods use two sequential calls of space-time A* to plan a path that goes through two goal locations. The drawback is, when A* computes an agent's path from the current location to the goal location, it assumes that the agent will stay at the goal indefinitely. Under this assumption, if the first goal location is another agent's second goal location, A* will fail; if the first goal location will be visited by another agent in the near future, A* will plan a longer path that allows the other agent to pass through this goal location first. In both cases, this agent can actually pass through the first goal location and immediately proceed to the second goal location before the other agent arrives [10]. To plan a path that goes through two goal locations, MLA* introduces a "label" dimension to indicate the current state of the vertex: If "label" is 1 then the agent is seeking a path to the first goal location, and if "label" is 2 then the

agent is seeking a path to the second goal location. The algorithm returns a solution only when the current vertex has a "label" 2 and it reaches the delivery location.

Li et al. [17] generalize MLA* to plan paths for agents with a varying number of goal locations. In addition to this, they propose Rolling-Horizon Collision Resolution (WHCR), an efficient solver for lifelong MAPF, where agents are constantly engaged with new goal locations and we need to plan collision-free paths for them. WHCR uses a rolling-horizon framework to repeatedly call a Windowed MAPF solver that resolves collisions only for a few time steps ahead. Such Windowed MAPF solvers run significantly faster than regular MAPF solvers but usually lose the completeness guarantee as they can lead to deadlocks due to their shortsightedness.

2.3 Multi-Agent Pickup and Delivery

In Multi-Agent Pickup and Delivery (MAPD) [21], agents are constantly engaged with new tasks and we need to assign tasks and plan collision-free paths for the agents. Each task in MAPD consists of two goal locations, namely the pickup location and the delivery location. To execute a task, an agent needs to first visit its pickup location and then visit its delivery location; a task is completed when the agent arrives at its delivery location. The quality of a MAPD solution is measured by the service time, i.e., the average time needed to finish executing each task after it was released to the system, or the makespan, i.e., the maximum of the completion times of all tasks. MAPD is NP-hard to solve optimally for both service time and makespan minimization [19].

Ma et al. [21] present a complete algorithm CENTRAL for well-formed MAPD, a subclass of MAPD instances that are widely used in many real-world applications. CENTRAL considers MAPD in a online setting. At each time step, it first uses the Hungarian [16] method to assign each agent one goal location, and then uses CBS to plan paths for all agents from their current locations to their assigned goal locations. Finally, all agents move along their planned paths for one time step and the procedure repeats. TA-Hybrid [18] considers the offline setting. It formulates the task assignment problem as a TSP and uses an anytime TSP solver LKH3 [12] to find one task sequence for each agent. At each time step, TA-Hybrid plans paths for two groups of agents from their current locations to their next goal locations.

Grenouilleau et al. [10] propose an H-value-Based Heuristic (HBH) that takes into account the distance between an agent's current location and the task's pickup location, and the distance between the task's pickup and delivery location; they then sort the list of agent-task pair in a non-decreasing order of HBH. For each agent-task pair, they use MLA* to plan an agent's path that goes through a pair of goal locations from their current location, and they remove the corresponding agent and task from the available agent set and the unassigned task set if MLA* returns a path successfully.

The above are decoupled MAPD solvers, i.e., they first assign tasks to the agents based on an estimate of the actual path cost, and then use a path finding solver to plan actual paths for agents. Chen et al. [5] propose a coupled MAPD solver RMCA that assigns tasks and plans paths simultaneously, therefore their task assignment decision can be informed by the actual path cost. The idea of RMCA is to maintain a task assignment heap that sorts all the potential assignment in a specific order. A potential assignment includes the updated path and cost after inserting an unassigned task into an agent's current task sequence. Whenever we select the top assignment and assign the task to the corresponding agent, we need to update all the remaining assignment in the heap that are affected by this new assignment. This includes the potential assignments on the selected agent, and the paths of other agents' potential assignments that are colliding with this agent's updated path. For the path finding part, RMCA uses the Prioritized Planning with sequential A* calls to plan a single agent's path that avoids collisions with the existing paths of other agents. Furthermore, Chen et al. [5] use an LNS anytime improvement strategy: They first use RMCA with a standard regret-based marginal-cost heuristic to construct the initial solution, then they iteratively remove a subset of tasks using a greedy heuristic, and reassign these tasks using RMCA. Henkel et al. [13] propose a Task Conflict-Based Search (TCBS) solver to solve the combined task allocation and multi-agent path finding problem optimally. However, their optimal solver can only scale to 4 robots and 4 tasks.

Chapter 3

Problem Definition

In this section, we formalize a generalized variant of the MAPD problem, namely the Multi-Goal MAPD problem. MAPD is a special case with only two goal locations (i.e., pickup and delivery location) in one task.

A Multi-Goal MAPD (MG-MAPD) instance consists of a set of M agents $\{a_1, a_2, ..., a_M\}$ and an undirected graph $G = (V, E)$, where vertices V represent the set of locations and edges E represent the connections between locations that agents can move along. Let $p_i(t)$ denote the location of a_i at time step t. Agent a_i starts at its start location $p_i(0)$; at each time step, it either moves to an adjacent location or waits at its current location. A vertex collision occurs between a_i and a_j if $p_i(t) = p_j(t)$; an edge collision occurs if $p_i(t) = p_j(t+1)$ and $p_i(t+1) = p_j(t)$.

At each time step, the system releases new tasks (if any). Each task τ_i is characterized by a sequence of goal locations and a release time r_i; we let s_i denote its first goal location and g_i denote its last goal location. To execute τ_i, an agent needs to visit all the goal locations in sequence. When an agent arrives at s_i, it starts to execute τ_i and cannot execute other tasks; the completion time of τ_i is the time when the agent arrives at g_i. We let $\mathcal{T}$ denote the set of all unexecuted tasks. Agents that are assigned tasks are called task agents; otherwise, they are called free agents.

Not all MG-MAPD instances are solvable; in this work, we consider the well-formed MG-MAPD, a realistic subclass of MG-MAPD instances [21, 18]. We specify two types of endpoints on our map: (1) all the goal locations of tasks are called task endpoints, and (2) the start locations of the agents are called non-task endpoints. A MG-MAPD instance is well-formed if each agent's start location is different from all the task endpoints, and for any two endpoints, there exists a path between them that traverses no other endpoints. In a well-formed MG-MAPD instance, agents can stay at the non-task endpoints indefinitely to avoid collisions with other agents.

The problem of MG-MAPD is to assign tasks to agents and plan collision-free paths for them such that the tasks are executed effectively and efficiently. The effectiveness of a MG-MAPD solver is measured by the average service time. The service time of a task is

the difference between its completion time and its release time, i.e., the waiting time a task spends in the system. The efficiency is measured by the computation time per time step.

Chapter 4

Method

Algorithm 1 without the blue parts (i.e., Lines [6-8]) shows how LNS-PBS works: when the system releases new tasks or when a task agent finishes executing its task sequence, LNS is triggered to (re)assign all the unexecuted tasks to agents. This will destroy agents' current task sequences (except for the tasks they are currently executing) and replan new task sequences for them. Then we update the goal sequence of each agent based on its task sequence constructed by LNS, and use PBS to (re)plan paths for the agents from their current locations. We will explain Line [3] in Section 4.1 and Lines [4-5] in Section 4.2. We will then prove the completeness of LNS-PBS for well-formed MG-MAPD in Section 4.3 and last introduce LNS-wPBS (i.e., Lines [6-8]) in Section 4.5.

4.1 Large Neighborhood Search

LNS is a local search algorithm that starts with an initial solution and iteratively replaces it with a better solution. In each iteration, LNS selects a subset of variables, re-optimize their values, and accept the resulting solution if it is better than the old one.

Hungarian-Based Insertion We apply the Hungarian algorithm [16] to construct agents' initial task sequences (i.e., the initial solution). Each call of the Hungarian algorithm adds one task to the end of the task sequence of each agent. We repeatedly call it until all the unexecuted tasks are assigned. In each call, the Hungarian algorithm takes a cost matrix as input and outputs an agent-task assignment with the minimum cost. Previous works define each element of the cost matrix as the estimated cost of the shortest path from an agent's current location to the first goal location of a task. This choice prioritizes those tasks whose first goal locations are near agents' current locations, without considering the tasks' release and completion time. Instead, we propose to define the cost between an agent a_i and a task τ_i as the estimated completion time of τ_i executed by a_i. To compute this, we assume τ_i is inserted at the end of the existing task sequence of a_i and estimate the completion time using the shortest path distance without considering collision avoidance.

Algorithm 1 LNS-PBS (with time window)

```
1: while true do
2:     if system releases new tasks or any task agents finish their tasks then
3:         Assign all unexecuted tasks using LNS;
4:         Update goal sequence;
5:         Plan paths for all agents using wPBS;
6:     else if agents have moved w timesteps then
7:         Update goal sequence;
8:         Plan paths for all agents using wPBS;
9:     end if
10:    Agents move one timestep following their paths;
11: end while
```

Shaw Removal After the construction of the initial task sequences, we use the Shaw removal operation [25] to remove a group of tasks from the current sequences. The idea is that when inserting related tasks back into the sequence, we are more likely to obtain different task sequences since related tasks are more easily to be exchanged. When the tasks are not related to each other, they are more likely to be inserted into their original position and therefore the sequence won't change. We let $d(u, v)$ represent the estimated cost of the shortest path from u to v without collision checking. The Shaw removal operation defines the relatedness between two tasks τ_i and τ_j as

$$r(\tau_i, \tau_j) = \omega_1(d(g_i, g_j) + d(s_i, s_j)) + \omega_2(|t(s_i) - t(s_j)| + |t(g_i) - t(g_j)|),$$

where $t(s_i)$ represents the estimated time when an agent starts to execute τ_i (i.e., when the agent reaches the first goal location s_i of τ_i), and $t(g_i)$ represents the estimated completion time of τ_i (i.e., when the agent reaches the last goal location g_i of τ_i). Both distance and time measurements are considered and weighted by ω_1 and ω_2 respectively. To compute the relatedness, we still use the shortest path distance without collision checking to estimate the actual path cost. The Shaw removal operation works as follows: we first choose a task τ^* randomly and compute the relatedness of all the other tasks with τ^*. We then remove a group of tasks (including τ^* itself) in decreasing order of the relatedness.

Regret-Based Re-insertion We then use a re-insertion operator to insert the removed tasks back to the task sequences. Specifically, we use the regret-based heuristic from [5][25]. Let f denote the estimated total service time of the current task sequences (that do not contain the removed tasks). Let $f^j_{i,k}$ denote the estimated total service time of the task sequences obtained by inserting task τ_i into the jth position of agent a_k's task sequence. Let $f^{(1)}_i$ denote the estimated total service time of the task sequences obtained by inserting task τ_i to its best position that increases the estimated total service time the least, i.e., $f^{(1)}_i = \min\{f^j_{i,k} \mid, k \in \{1, ..., M\}, j \in \{0, ..., l_k\}\}$ where l_k is the number of tasks in a_k's current task sequence. Let $f^{(2)}_i$ denote the estimated total service time of the schedule obtained

by inserting task τ_i to its second-best position that increases the estimated total service time the second-least. The regret value of a task τ_i is defined as $r_i = f_i^{(2)} - f_i^{(1)}$, i.e., the difference in the estimated total service time of inserting task τ_i to its best position and the second-best position. The regret-based re-insertion operation works as follows: we greedily choose the task with maximum regret value and insert this task to its best position. We also need to update the regret value for the remaining tasks based on this new assignment. Then we repeat the process until all the removed tasks are inserted back to the task sequences.

In each iteration, we remove a group of tasks from the sequences and insert them back to the sequences. If there is an improvement on the estimated total service time, we will replace the original solution with the new one; if there is no improvement, we will reserve the original solution. This procedure is repeated until we hit the time limit. We call the number of tasks we remove and re-insert in each iteration the size of the neighborhood.

4.2 Priority-Based Search

Priority-Based Search (PBS) [20] is a two-level MAPF algorithm. On the high level, PBS performs a depth-first search to construct a priority ordering and builds a priority tree (PT). PBS starts with a root node that contains an empty priority ordering and an individually optimal path for each agent. When resolving a collision between two agents, PBS generates two child nodes and adds an additional priority order to each of them that indicates one of the agents involved in the collision has a higher priority than the other agent. On the low level, PBS calls A* to plan optimal paths for agents that are consistent with the priority ordering generated by the high level (i.e., lower-priority agents do not collide with higher-priority agents). Li et al. [17] generalize the low level of PBS so that it can plan collision-free paths for agents with a sequence of goal locations. PBS is in general incomplete; we introduce dummy endpoints and modify the low level of PBS in a way such that it is complete for well-formed MG-MAPD instances.

Formally, when LNS is triggered to (re)assign tasks at timestep t, we need to specify a goal sequence for each agent, and plan agents' paths starting from their current locations at timestep t. To avoid planning a path for a long task sequence, we will truncate the task sequence with a user-specified variable C. For a task agent, its goal sequence consists of all the goal locations of the first C tasks in this agent's task sequence, and a dummy endpoint in the end; for a free agent, its goal sequence contains no goal locations except the dummy endpoint.

Dummy Endpoints Whenever we plan new paths for agents to go through their goal sequence, we also plan paths from agents' last goal locations to their dummy endpoints. We assign one dummy endpoint to each agent and require the agents to hold their dummy endpoints, i.e., agents can stay at their dummy endpoints indefinitely. We assign the dummy endpoints in two steps: (1) assign dummy endpoints to task agents one by one, and these

dummy endpoints should be pairwise different and different from all the task endpoints in $\mathcal{T}$; (2) assign dummy endpoints to free agents one by one, and these dummy endpoints should be pairwise different and different from all the endpoints assigned in first step and all the task endpoints in $\mathcal{T}$. When choosing a dummy endpoint for an agent, we consider the task endpoints in increasing order of their estimated distance to agents' last goal location. If there are no available task endpoints to assign, we will the closet non-task endpoint as its dummy endpoint. In the next iteration, we do not assign those tasks whose goal location is already used as a dummy endpoint in this iteration. The paths that start from agents' last goal locations to their assigned dummy endpoints are never executed by the agents, but we still need to reserve these old paths (including the dummy paths) so that in the worst case agents can follow their old paths and stay at their dummy endpoints as long as needed without colliding with other agents.

PBS Low-Level Search Before PBS starts, we save the old paths computed from last iteration. For the first iteration, the old paths are the paths that agents stay at their start locations indefinitely. These old paths might not visit the goal locations in the current goal sequences, but they are guaranteed to be collision-free. When PBS generates the root node, it plans a shortest path for each agent that avoids the other M-1 agents' old paths. When PBS resolves a collision between two agents, for each child node generated, PBS plans a shortest path for an agent that avoids collisions with the new paths of all higher priority agents and the old paths of the other agents. When the given MG-MAPD instance is well-formed, this modification enables every high-level node to always successfully find a feasible path for each agent (which we will prove later), and therefore, the number of PT node expanded is no larger than the maximum depth of the PT, which is $\mathcal{O}(M^2)$ [20].

4.3 Completeness of LNS-PBS

Now, we prove the completeness of LNS-PBS for well-formed MG-MAPD.

Theorem 1. *Given a well-formed MG-MAPD instance with a finite number of tasks, LNS-PBS is guaranteed to plan collision-free paths for the agents to execute all the tasks in finite timesteps.*

Proof. We first prove that all tasks will be assigned to the agents in finite time. Assume in the Tth iteration, a new task τ_i is added to $\mathcal{T}$. Task τ_i is not assigned to any agent since one of its goal locations is used as a dummy endpoint in the $(T-1)$th iteration. Then task τ_i will be assigned in the $(T+1)$th iteration. This is because the dummy endpoints assigned in the Tth iteration are different from all the task endpoints in $\mathcal{T}$. So in the $(T+1)$th iteration, task τ_i will be assigned to an agent since none of its goal locations are used as a dummy endpoint in the Tth iteration.

We then prove that, LNS-PBS will return paths successfully for all agents such that their assigned tasks are executed in finite time. In the first iteration, for the root node, PBS

can plan a path for each agent that does not collide with the other $M - 1$ agents' old paths. We can do this since our instances are well-formed, which implies that there exists a path between any two endpoints that avoids the start locations. For each child node, PBS can plan a path for an agent since, in the worst case, this agent can stay at its start location, and let all higher priority agents move along their new paths and stay at their dummy endpoints. Then this agent can plan a path through its goal sequence without using the dummy endpoints of higher-priority agents and the start locations of the other agents. We can do this since the assigned dummy endpoints are pairwise different and different from all the task endpoints.

In the next iteration, LNS-PBS is also guaranteed to return paths successfully for all agents. For the root node, in the worst case, an agent can move along its old path and wait at its old dummy endpoint until all other agents move along their old paths and arrive at their old dummy endpoints. Then this agent plans a path to visit its goal sequence and stay at its new dummy endpoint. We can do this since, in this iteration, we don't assign those tasks whose goal locations are used as dummy endpoints in the previous iteration. For each child node, PBS can successfully plan a path since, in the worst case, the planned agent can first move along its old path and stay at its old dummy endpoint, and let higher-priority agents move along their new paths and stay at their new dummy endpoints, and let the other agents move along their old paths and stay at their old dummy endpoints, then this agent plans a path through its goal sequence and stay at its new dummy endpoint. □

4.4 LNS-wPBS

In this section, we consider a variant of LNS-PBS, called LNS-wPBS, that is more efficient but is not complete. Compared with LNS-PBS, LNS-wPBS is different from three perspectives:

First of all, LNS-wPBS reserves the "time window" in WHCR, i.e., wPBS plans collision-free paths of length w timesteps, and paths are planned again when agents have moved for w timesteps. By comparison, LNS-PBS always plans the entire paths, ensuring that agents will fully execute the first C tasks in their task sequences. Secondly, the low level of wPBS is the same as the original PBS (that do not need to avoid collisions with the old paths of agents): for the root node, PBS plans an individual optimal path for each agent without any constraints; for the child node, PBS plans a shortest path that does not collide with the paths of higher priority agents. Thirdly, the dummy endpoints assigned to the agents only need to satisfy the condition that they are pairwise different from each other. The blue parts in Algorithm 1 are the differences between LNS-PBS and LNS-wPBS.

LNS-wPBS is not complete. This is because wPBS only plans a path of length w timesteps, so there is no guarantee that agents can reach their goal locations in finite timesteps. Nevertheless, LNS-wPBS always successfully finds solutions in our experiments.

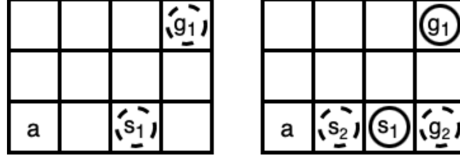

Figure 4.1: A MG-MAPD instance at timestep 0 and 2 (left and right). Solid circles represent the existing tasks from previous timesteps and dash circles represent the new task released at the current timestep.

4.5 Look-Ahead Horizons

In this section, we study the semi-online setting, i.e., the system has partial knowledge about the future tasks that it can plan ahead for. To handle this, LNS needs to consider all the known tasks when generating task sequences. We divide tasks into batches, where the tasks in one batch are released at the same timestep. Here we define a look-ahead horizon as the number of batches of tasks we can know in advance. For example, if the system releases 0.2 tasks per timestep (i.e., releases one task per five timestep), a look-ahead horizon of 1 means that, at timestep 0, we know the tasks that will be released at timestep 0 and 5. The problem becomes offline if the horizon is infinite.

If the system knows the incoming tasks ahead of their release time, we can send an agent to the pickup location and wait for the task to be released. For example in Figure 4.1, we assume that task τ_1 is released at timestep $t = 0$ with pickup location s_1 and delivery location g_1. At timestep $t = 2$, the system releases another task $\tau_2 = (s_2, g_2)$. If the system has no knowledge about this incoming task, we will first assign τ_1 to the agent at timestep 0 and then assign τ_2 when it is released at timestep 2. Thus, the completion time of τ_1 is timestep 5 and the completion time of τ_2 is timestep 11. Since τ_1 is released at timestep 0 and τ_2 is released at timestep 2, the average service time is $(5-0+11-2)/2 = 7$. However, if our system has a look-ahead horizon of 1, we can let our agent first move to s_2 and wait for one timestep. It then starts to execute τ_2 at timestep 2 and then execute τ_1 at timestep 5. Thus, the completion time of τ_2 is timestep 4 and the completion time of τ_1 is timestep 8. This produces an average service time of $(8 - 0 + 4 - 2)/2 = 5$.

Chapter 5

Experiments

In this section, we compare our method with other MAPD methods on MAPD instances. We also compare different task assignment heuristics on MG-MAPD instances and test our method in a semi-online setting. Our experiments are performed on a macOS 2.3 GHz Intel Core i5, with an 8 GB RAM environment. All the methods are implemented in C++, and the implementation of our method is based on the codebase provided by WHCR [17].

In all experiments, we choose a neighborhood size of 2 for LNS procedure and set the optimization time to 1 second. We use the same parameters $\omega_1 = 9$ and $\omega_2 = 3$ as in [25] to compute the Shaw removal heuristic. Our experiments are performed on two 2d 4-neighbouring grids, as shown in Figure 5.1. For MAPD instances, the pickup and delivery locations are randomly generated from all the task endpoints; for MG-MAPD instances, we sample tasks from 1 to 5 goal locations, and each goal location is randomly generated from all the task endpoints. We set $C = 2$ in small warehouse, which allows PBS to plan paths for agents to execute two tasks at a time; we set $C = 1$ in large warehouse, which allows PBS to plan paths for agents to execute only one task at a time. We set the time window of wPBS to be 5 timesteps in a small warehouse environment and 10 timesteps in a large warehouse environment. Since the method is deterministic, a single execution is sufficient. For each instance, we report the average service time and the runtime (ms) per timestep in a small warehouse:

Small Warehouse In a small warehouse environment, we test seven task frequencies in the online setting: 0.2 (system releases 0.2 tasks every timestep), 0.5, 1, 2, 5, 10 (10 tasks are released every timestep) and 500 (all 500 tasks are released at the beginning). For each task frequency, we test with 10, 20, 30, 40 and 50 agents. We compare LNS-PBS with CENTRAL, since they are both complete methods for well-formed MAPD, and we compare LNS-wPBS with RMCA, which is the state-of-the-art MAPD method that has no completeness guarantees.

Table 5.1 shows that LNS-PBS produces a smaller service time than CENTRAL in almost every instance. When task frequency increases ($\geq$2), the average improvement is above 10% and the largest improvement is 26% (for a task frequency 2 and 50 agents

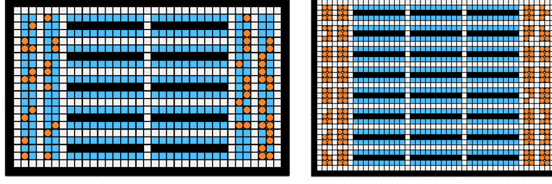

Figure 5.1: Two simulated warehouse maps borrowed from [18]. Left: a small warehouse environment with 50 agents. Right: a large warehouse environment with 180 agents. The blue cells represent task endpoints, the orange cells represent non-task endpoints and the black cells are blocked.

Table 5.1: Compare CENTRAL and LNS-PBS, RMCA and LNS-wPBS in a small warehouse environment

		CENTRAL		LNS-PBS ($C = 2$)		RMCA		LNS-wPBS ($w = 5$)	
f	agents	service time	runtime	service time	runtime	service time	runtime	service time	runtime
0.2	10	29.77	28.16	27.92	313.45	26.74	200.08	27.87	316.08
	20	26.70	136.21	25.33	294.94	24.28	200.74	25.67	316.50
	30	25.56	305.78	25.10	292.04	23.27	201.88	24.69	298.29
	40	25.46	415.25	24.30	286.58	22.62	202.98	24.58	291.88
	50	25.05	757.40	24.09	277.72	22.37	205.00	24.39	292.96
Avg. Gap				-4.2%	+206.2%			+6.7%	+50.0%
0.5	10	109.71	51.23	116.59	400.44	101.62	438.57	117.44	382.50
	20	27.99	172.36	26.91	646.19	25.44	496.28	27.52	617.43
	30	26.23	512.04	25.26	667.61	23.66	501.24	25.72	635.44
	40	25.39	1017.49	24.65	667.25	22.73	503.49	24.84	657.18
	50	24.94	1736.7	23.94	666.01	22.44	508.45	24.76	645.86
Avg. Gap				-1.6%	+178.1%			+10.4%	+19.1%
1	10	285.75	65.70	273.48	448.81	269.766	464.67	266.77	419.59
	20	75.13	266.76	67.21	880.60	59.12	851.97	67.20	762.55
	30	31.41	492.12	28.82	1030.93	25.59	974.76	28.05	947.47
	40	28.33	1381.56	25.28	1042.05	23.67	987.04	25.62	960.76
	50	27.38	3238.17	24.42	1055.77	23.01	995.00	25.30	958.01
Avg. Gap				-8.8%	+166.1%			+8.0%	-5.8%
2	10	388.21	81.35	361.59	258.75	371.272	231.32	356.90	229.33
	20	162.00	424.18	140.27	477.73	146.816	444.33	140.22	420.59
	30	85.89	702.22	75.45	749.36	77.75	635.75	74.30	597.24
	40	57.53	1440.2	44.55	1307.76	43.49	798.31	44.70	752.98
	50	41.43	2206.7	30.46	1249.02	28.88	927.25	31.01	893.02
Avg. Gap				-16.2%	+36.9%			-0.6%	-4.3%
5	10	455.16	85.32	412.75	157.27	435.70	99.57	408.77	109.84
	20	229.55	422.41	197.28	244.39	209.558	184.11	197.51	187.50
	30	147.76	1012.82	126.41	373.18	132.06	268.07	123.95	272.18
	40	108.28	1745.05	90.01	627.75	96.81	362.65	91.01	364.22
	50	86.90	2686.08	70.31	914.49	74.32	425.26	72.25	422.82
Avg. Gap				-14.7%	-30.1%			-5.3%	+2.7%
10	10	478.17	92.96	438.71	117.76	458.23	56.68	431.76	65.62
	20	242.18	375.23	217.33	168.63	228.9	101.20	215.74	110.00
	30	165.13	869.85	146.56	254.68	154.28	152.34	144.11	163.94
	40	128.39	1723.1	110.41	381.89	115.04	208.32	109.10	203.33
	50	106.70	7442.20	88.75	602.85	94.29	246.96	89.33	243.87
Avg. Gap				-12.0%	-5.7%			-5.7%	+5.6%
500	10	501.11	76.03	447.59	76.60	N/A	N/A	456.41	15.18
	20	263.55	374.87	234.72	92.18	N/A	N/A	234.88	31.80
	30	187.31	19471.88	165.09	127.70	N/A	N/A	162.52	47.03
	40	N/A	N/A	128.87	180.36	N/A	N/A	129.72	61.30
	50	N/A	N/A	110.04	246.96	104.01	259.34	109.40	65.82
Avg. Gap				-11.5%	-57.9%			+5.1%	-74.6%

Table 5.2: Compare HBH+MLA*, RMCA, LNS-wPBS and LNS-PBS in a large warehouse environment with 1000 tasks and 100, 200, 300, 400, 500 agents. 50 tasks are released in one timestep.

	HBH+MLA*		RMCA		LNS-wPBS ($w = 10$)		LNS-PBS ($C = 1$)	
agents	service time	runtime	service time	runtime	service time	runtime	service time	runtime
100	362.70	1.99	329.58	565.76	300.90 (-17.0%, -8.7%)	87.35	301.78	345.36
200	207.76	6.75	192.67	2072.98	176.81 (-14.8%, -8.2%)	220.28	176.13	3065.95
300	157.11	14.89	147.42	4734.94	139.33 (-11.3%, -5.4%)	465.78	137.97	8844.98
400	136.40	32.59	126.44	9906.40	123.32 (-9.5%, -2.4%)	806.54	N/A	N/A
500	125.42	65.79	N/A	N/A	113.78 (-9.2%)	1385.9	N/A	N/A

instance). The only instance in which our method produces a larger service time than CENTRAL (+6.2%) is when the task frequency is 0.5 and the agent number is 10. In terms of the runtime comparison, we observe the limited scalability of CENTRAL. For the task frequency 500 instances, CENTRAL requires almost 20 seconds per timestep to solve for 30 agents, and it takes even longer to solve for 40 and 50 agents, while the runtime for LNS-PBS is still less than 1 second per timestep. However, when the task frequency is small (≤ 2) and the agent number is small (≤ 30), CENTRAL is more efficient than LNS-PBS.

When task frequency is small (≤ 1), RMCA produces a solution with smaller service time and runtime than LNS-wPBS. However, LNS-wPBS outperforms RMCA when there are more tasks to plan, e.g., when task frequency is 5 and 10, the average improvement produced by LNS-wPBS is about 5%. We also test RMCA in the task frequency 500 instances, but their program only succeeded in one instance (50 agents). The service time produced by LNS-wPBS is very close to the service time produced by LNS-PBS, however, the runtime required for LNS-wPBS are all below 1000 ms per timestep. We will observe a larger gap in the runtime when running both methods in the large warehouse.

Large Warehouse We perform two set of experiments in the large warehouse. In first experiment, the task set is fixed: the system releases 50 tasks every timestep and there are 1000 tasks in total. We test 5 instances: 100, 200, 300, 400, 500 agents. In second experiment, the agent set is fixed: the system releases 100 tasks every timestep and there are 1000 agents in total. We test 5 instances: 1000, 2000, 3000, 4000, 5000 tasks. The runtime limit is 1.5 hours for each instance.

Table 5.2 shows that LNS-wPBS produces a solution of smaller service time than HBH+MLA* for every instance (more than 9%). However, HBH+MLA* runs faster since it assigns tasks in a greedy way and uses prioritized planning algorithm to plan paths for agents sequentially. Both RMCA and LNS-PBS has timeout issues when the number of agents increases (≥ 500 and ≥ 400 respectively). LNS-wPBS outperforms RMCA when the agent number decreases, and the maximum gap between their service time is 8%. Even though LNS-wPBS produces comparable results with RMCA when the agent number is large, LNS-wPBS still runs faster and has better scalability.

Table 5.3: Compare HBH+MLA* and LNS-wPBS in a large warehouse environment with 1000 agents and 1000, 2000, 3000, 4000, 5000 tasks. 100 tasks are released in one timestep.

	HBH+MLA*		LNS-wPBS ($w = 10$)	
tasks	service time	runtime	service time	runtime
1000	162.98	373.10	155.00 (-4.8%)	9288.45
2000	209.89	468.74	193.22 (-7.9%)	7792.24
3000	258.74	346.44	233.99 (-10.5%)	8173.96
4000	307.59	400.26	274.54 (-10.7%)	7730.04
5000	356.60	487.90	314.42 (-11.8%)	5038.37

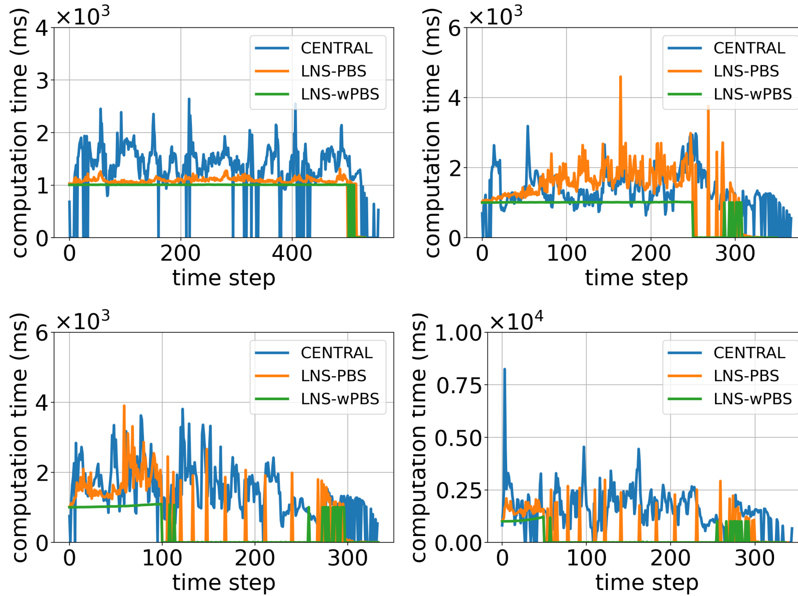

Figure 5.2: Total computation time per timestep. The charts are generated for 40 agents and task frequency 1, 2, 5 and 10 instances (from left to right and top to bottom).

A similar trend is observed in Table 5.3, where we only compare LNS-wPBS with HBH+MLA* since the other two methods cannot scale to thousand of agents and thousands of tasks. HBH+MLA* still runs faster, however, LNS-wPBS produces a solution with smaller service time, and the gap increases when there are more tasks to plan; the maximum gap we achieve is 11%. Notice that the runtime of LNS-PBS is decreasing when the number of tasks is increasing. This is because here we are computing the runtime per timestep, which is the total runtime divided by the makespan, and the makespan is longer when there are more tasks to plan.

Runtime and Throughput Figure 5.2 shows that the computation time of LNS-wPBS is more stable than the computation time of LNS-PBS and CENTRAL. This is because the computation time of LNS-wPBS can be controlled by the user-specified time window of wPBS and the user-specified time limit of LNS. Notice that for all three methods, the computation (of task assignment and path finding) is only triggered under certain

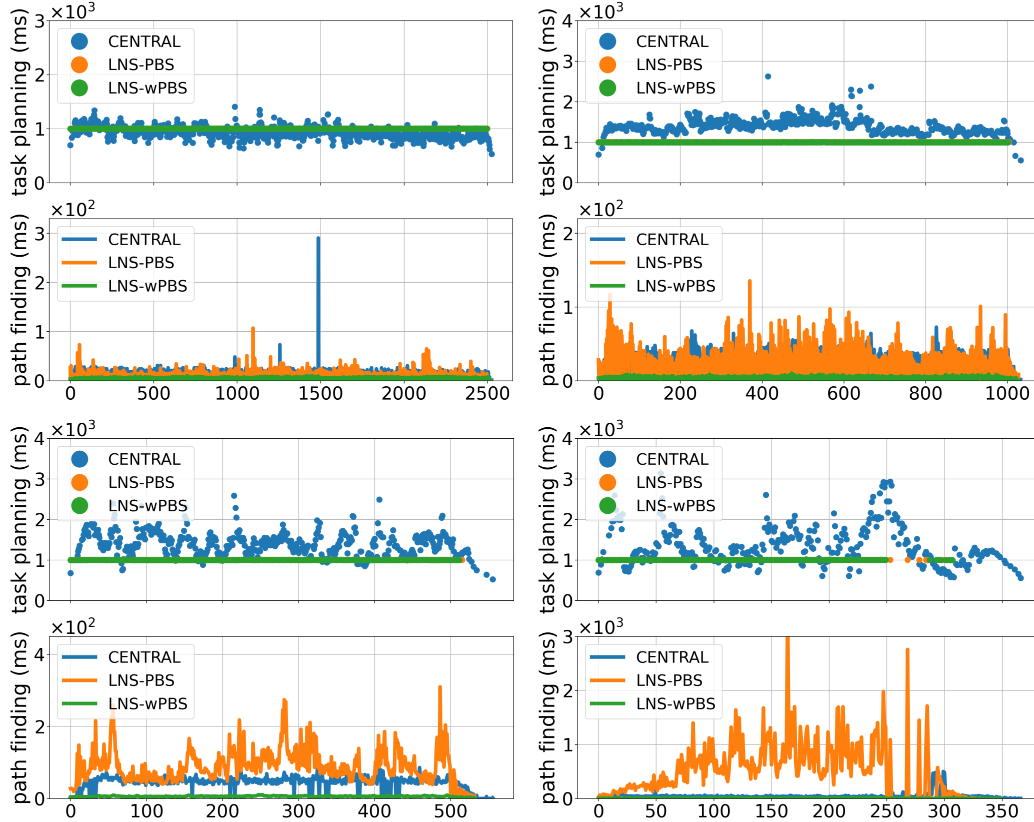

Figure 5.3: Task planning time and path finding time per timestep. The charts are generated for 40 agents and task frequency 0.2, 0.5, 1 and 2 instances (from (a) to (d)). The orange dots are hidden by the green dots since the task planning time of LNS-PBS and LNS-wPBS are very close.

circumstances, e.g., when new tasks are released, so the computation time is close to 0 for some timesteps and the variance is very high in the charts. However, we can still see that the computation time of LNS-wPBS is always around 1 second (which is the time limit of we set for LNS), while the computation time of the other two methods deviate significantly over the timeline.

We further divide the total computation time into two parts: task planning and path finding. In Figure 5.3. We can see that LNS-PBS and LNS-wPBS both take about 1 second to assign tasks, whereas the computation time of task planning in CENTRAL is unpredictable. We also notice that the path planning time for LNS-wPBS is more evenly distributed than LNS-PBS and CENTRAL. Therefore, the total computation time for LNS-wPBS is under control, whereas it is difficult to predict the total computation time for the other two methods.

Figure 5.4 shows the number of tasks added and completed by 50 agents in a time window $[t-99, t]$ as a function of timestep t. When task frequency is 0.2, 0.5 and 1, these

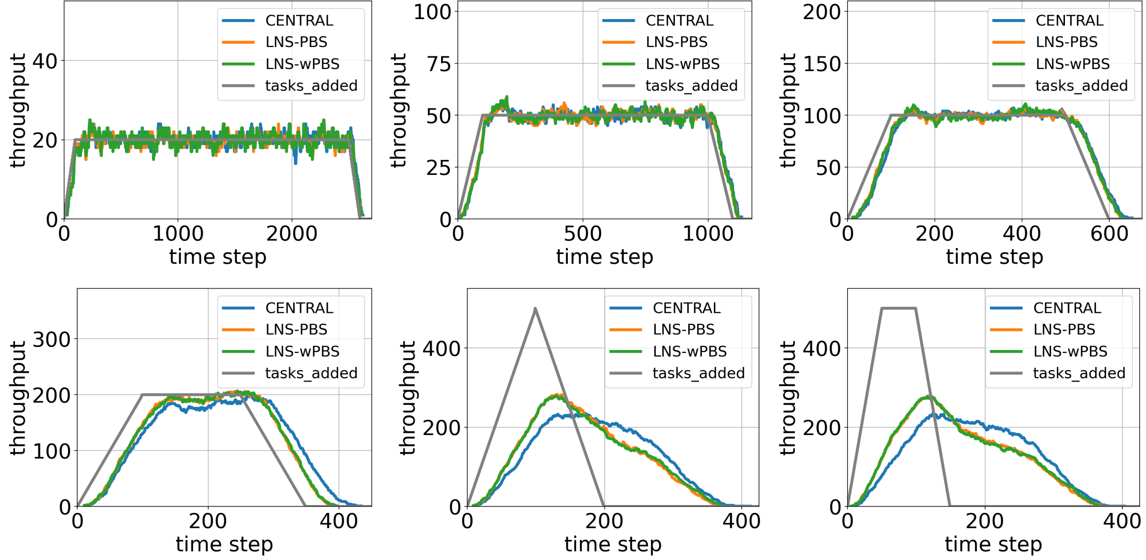

Figure 5.4: Throughput and the number of tasks added per timestep. The charts are generated for 50 agents and task frequency 0.2, 0.5, 1, 2, 5 and 10 instances (from left to right and up to bottom).

three methods perform equally well, and they all produce a throughput that is close to the number of tasks added. When task frequency increases, i.e., more tasks are added at each timestep, LNS-PBS and LNS-wPBS behave similarly and produce higher throughput than CENTRAL.

Task-Assignment Heuristics In this experiment, we compare the performance of Hungarian-LNS with two other task assignment heuristics in assigning multi-goal tasks. Hungarian heuristic assigns at most one task to an agent, whereas Greedy-LNS and Hungarian-LNS can assign a sequence of tasks to an agent. Hungarian uses the completion time of a task executed by an agent as the cost of each agent-task pair, instead of the distance between an agent's current location and the first goal location of a task. Greedy-LNS uses a standard regret-based heuristic to construct the initial solution (as in [25, 5]), whereas Hungarian-LNS uses a sequence of Hungarian calls to construct the initial solution; they both iteratively optimize the solution using LNS as we describe in the previous section.

The results are summarized in Table 5.4. The gap is calculated based on the solution produced by Hungarian and Hungarian-LNS. We can see that for a few instances, Hungarian outperforms the other two with a small margin, e.g., for the instance with task frequency 0.5 and 30 agents; in most cases, Hungarian-LNS produces smaller service time than the other two heuristics. We also notice that Greedy-LNS produces a solution with smallest service time when there are fewer agents to plan, e.g., for the instances with task frequency 2, 5 and 10 agents.

Look-Ahead Horizons In this experiment, we explore the semi-online setting when a look-ahead horizon is applied and we test in MG-MAPD instances. The results are sum-

Table 5.4: Service time of MG-MAPD instances produced by three task assignment methods: Hungarian, Greedy-LNS and Hungarian-LNS

f	agents	Hungarian	Greedy-LNS	Hungarian-LNS	Gap (%)
0.5	10	382.95	370.84	**369.47**	-3.5
	20	88.74	90.43	**87.57**	-1.3
	30	**47.04**	47.4	47.11	+0.1
	40	45.87	46.32	**45.82**	-0.1
	50	45.81	45.67	**45.40**	-0.8
1	10	527.51	521.25	**514.93**	-2.3
	20	204.63	210.22	**197.31**	-3.5
	30	111.57	114.17	**108.80**	-2.4
	40	70.67	68.37	**67.98**	-3.8
	50	**48.32**	49.07	48.69	+0.7
2	10	623.87	**590.54**	597.05	-4.2
	20	283.83	277.65	**269.87**	-4.9
	30	173.15	177.21	**170.07**	-1.7
	40	**122.21**	129.57	122.24	+0.0
	50	98.10	99.58	**95.94**	-2.6
5	10	683.50	**637.66**	645.21	-5.6
	20	332.23	327.57	**318.55**	-4.1
	30	224.21	221.70	**218.27**	-2.6
	40	168.11	173.47	**164.8**	-1.9
	50	141.61	141.95	**139.86**	-1.2
10	10	683.60	654.26	**650.63**	-4.8
	20	343.88	341.56	**336.97**	-2.0
	30	239.39	239.20	**229.76**	-4.0
	40	184.94	187.55	**178.19**	-3.6
	50	157.01	156.07	**149.91**	-4.5

Table 5.5: Service time for different look-ahead horizons: "LA1" represents look one batch of tasks ahead.

f	agents	LNS-wPBS	LA1	LA5	LA10
0.2	10	58.93	54.07 (-8.2%)	48.89 (-17.0%)	65.25 (+10.6%)
	20	46.52	41.85 (-10.0%)	37.94 (-18.4%)	38.12 (-18.0%)
	30	45.36	41.59 (-8.3%)	38.30 (-15.5%)	38.41 (-15.3%)
	40	45.17	41.99 (-7.0%)	38.80 (-14.1%)	38.84 (-14.0%)
	50	45.00	41.57 (-7.6%)	39.11 (-13.0%)	39.31 (-12.6%)
0.5	10	369.47	372.88 (+0.9%)	367.23 (+0.6%)	388.17 (+5.0%)
	20	87.57	86.22 (-1.5%)	81.77 (-6.6%)	80.35 (-8.2%)
	30	47.11	44.88 (-4.7%)	39.92 (-15.2%)	38.76 (-17.7%)
	40	45.82	44.60 (-2.6%)	39.76 (-13.2%)	38.82 (-15.2%)
	50	45.40	44.13 (-2.7%)	39.97 (-11.9%)	39.01 (-14.0%)

marized in Table 5.5. "LA1" represents look one batch of tasks ahead and we test three look-ahead horizons: 1, 5 and 10. We can see that knowing future tasks allows us to plan in advance and therefore obtain a smaller service time, but this benefit diminishes as we extend the horizon. For example, for instances whose task frequency is 0.2, extending the horizon from 1 to 5 can almost double the improvement on the service time. However, the improvements are very close (or even worse) when we further extend the horizon from 5 to 10. For example, for instances whose task frequency is 0.2, planning 10 batches of tasks ahead leads to a solution with a larger service time (+10%). We suspect the reason is: in order to optimize the entire task sequences, we need to sacrifice the service time of the first few tasks, and since the task sequences change frequently, the agents will never execute the whole task sequences as planned.

Chapter 6

Conclusions

In this work, we propose a decoupled MAPD method that assigns task sequences to agents and plans agents' paths through a sequence of goal locations. Two variants of this method are proposed: The first variant LNS-PBS is complete for well-formed MAPD instances, and the second variant LNS-wPBS is more efficient and stable (but has no completeness guarantees). The computation time of LNS-wPBS can be controlled by the anytime LNS task assigner and the user-specified time window in the path finding solver. Empirically, both LNS-PBS and LNS-wPBS produce a solution with smaller service time than the state-of-the-art methods, and LNS-wPBS can scale to thousands of agents and thousands of tasks in a large warehouse. As a further contribution, our method extends to MG-MAPD, a generalized variant of MAPD where tasks can have a varying number of goal locations. We also study the performance of our method under the online with look-ahead setting.

In our experiments, RMCA shows its great success on small instances, which indicates the advantage of assigning tasks based on the actual path cost. Motivated by their work, a future improvement of our work is to use a more informed heuristic, e.g., the actual path cost to help LNS escape from the local minimum and continue to optimize the task sequence. Another future direction is to use a 3D physics simulator to demonstrate the potential to run our method in actual robots.

Bibliography

[1] Tolga Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, pages 209–219, 2006.

[2] Vincenzo Bonifaci, Maarten Lipmann, and Leen Stougie. Online multi-server dial-a-ride problems. *Computational Complexity - CC*, 01 2006.

[3] Eli Boyarski, Ariel Felner, Roni Stern, Guni Sharon, David Tolpin, Oded Betzalel, and Eyal Shimony. Icbs: Improved conflict-based search algorithm for multi-agent pathfinding. In *Proceedings of the 24th International Conference on Artificial Intelligence*, page 740–746, 2015.

[4] Kyle Brown, Oriana Peltzer, Martin A. Sehr, Mac Schwager, and Mykel J. Kochenderfer. Optimal sequential task assignment and path finding for multi-agent robotic assembly planning. In *IEEE International Conference on Robotics and Automation*, pages 441–447, 2020.

[5] Zhe Chen, Javier Alonso-Mora, Xiaoshan Bai, Daniel D. Harabor, and Peter J. Stuckey. Integrated task assignment and path planning for capacitated multi-agent pickup and delivery. *IEEE Robotics and Automation Letters*, 6(3):5816–5823, 2021.

[6] Shushman Choudhury, Kiril Solovey, Mykel J Kochenderfer, and Marco Pavone. Efficient large-scale multi-drone delivery using transit networks. *Journal of Artificial Intelligence Research*, 70:757–788, 2021.

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

[8] B.P. Gerkey and M.J. Mataric. Multi-robot task allocation: analyzing the complexity and optimality of key architectures. In *2003 IEEE International Conference on Robotics and Automation*, volume 3, pages 3862–3868 vol.3, 2003.

[9] Brian P Gerkey and Maja J Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, pages 939–954, 2004.

[10] Florian Grenouilleau, Willem-Jan van Hoeve, and J. N. Hooker. A multi-label a* algorithm for multi-agent pathfinding. *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 181–185, May 2021.

[11] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[12] Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. 2017.

[13] Christian Henkel, Jannik Abbenseth, and Marc Toussaint. An optimal algorithm to solve the combined task allocation and path finding problem. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4140–4146, 2019.

[14] B. Kalyanasundaram and K. Pruhs. Online weighted matching. *Journal of Algorithms*, 14(3):478–488, 1993.

[15] G. Ayorkor Korsah, Anthony Stentz, and M. Bernardine Dias. A comprehensive taxonomy for multi-robot task allocation. *The International journal of robotics research*, pages 1495–1512, 2013.

[16] H. W. Kuhn and Bryn Yaw. The hungarian method for the assignment problem. *Naval Res. Logist. Quart*, pages 83–97, 1955.

[17] Jiaoyang Li, Andrew Tinka, Scott Kiesel, Joseph W. Durham, T. K. Satish Kumar, and Sven Koenig. Lifelong multi-agent path finding in large-scale warehouses. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, 2021.

[18] Minghua Liu, Hang Ma, Jiaoyang Li, and Sven Koenig. Task and path planning for multi-agent pickup and delivery. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 2253–2255, 2019.

[19] Hang Ma. *Target Assignment and Path Planning for Navigation Tasks with Teams of Agents*. PhD thesis, University of Southern California, 2020.

[20] Hang Ma, Daniel Harabor, Peter J. Stuckey, Jiaoyang Li, and Sven Koenig. Searching with consistent prioritization for multi-agent path finding. In *AAAI Conference on Artificial Intelligence*, pages 7643–7650, 2019.

[21] Hang Ma, Jiaoyang Li, T. K. Satish Kumar, and Sven Koenig. Lifelong multi-agent path finding for online pickup and delivery tasks. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 837–845, 2017.

[22] Hang Ma, Craig Tovey, Guni Sharon, T. K. Satish Kumar, and Sven Koenig. Multi-agent path finding with payload transfers and the package-exchange robot-routing problem. In *AAAI Conference on Artificial Intelligence*, pages 3166–3173, 2016.

[23] Hang Ma, Jingxing Yang, Liron Cohen, T. K. Satish Kumar, and Sven Koenig. Feasibility study: Moving non-homogeneous teams in congested video game environments. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pages 270–272, 2017.

[24] Robert Morris, Corina Pasareanu, Kasper Luckow, Waqar Malik, Hang Ma, T. K. Satish Kumar, and Sven Koenig. Planning, scheduling and monitoring for airport surface operations. In *AAAI-16 Workshop on Planning for Hybrid Systems*, pages 608–614, 2016.

[25] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, pages 455–472, 2006.

[26] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, page 563–569, 2012.

[27] Paul Shaw. A new local search algorithm providing high quality solutions to vehicle routing problems, 1997.

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

[29] Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Eli Boyarski, and Roman Bartak. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *International Symposium on Combinatorial Search*, pages 151–159, 2019.

[30] Nathan Sturtevant and Michael Buro. Improving collaborative pathfinding using map abstraction. pages 80–85, 01 2006.

[31] Pavel Surynek. Multi-goal multi-agent path finding via decoupled and integrated goal vertex ordering. In *AAAI Conference on Artificial Intelligence*, pages 12409–12417, 2021.

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

[33] Jingjin Yu and Steven M. LaValle. Structure and intractability of optimal multi-robot path planning on graphs. AAAI'13, page 1443–1449. AAAI Press, 2013.