# Multi-agent Pickup and Delivery Planning with Transfers

Brian Coltin

CMU-RI-TR-14-05

*Submitted in partial fulfillment of the*
*requirements for the degree of*
*Doctor of Philosophy in Robotics*

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

May 2014

**Thesis Committee:**
Manuela M. Veloso, Chair
Howie Choset
Stephen Smith
Daniele Nardi (Sapienza University of Rome)

*To my parents*

# Abstract

In Pickup and Delivery Problems (PDPs), mobile vehicles retrieve and deliver a set of items. The PDP is a well-studied, NP-hard problem. Examples of the PDP include mail and courier services, taxis, ridesharing services, and robots such as our own CoBots and CreBots, which retrieve and deliver items for the occupants of a building, and are the motivation for this thesis. The goal of the PDP is to find a schedule that delivers as many items as possible at the lowest cost, under various constraints such as time windows and vehicle capacities.

We augment the PDP with *transfers* to form the PDP with Transfers (PDP-T). Instead of having a single vehicle retrieve and deliver each item, vehicles can transfer items to other vehicles (or chains of vehicles) for delivery. Allowing transfers makes lower cost solutions feasible, but increases the number of possible schedules exponentially.

In this thesis, we contribute a series of algorithms to form schedules for variants of the PDP-T. We introduce the Very Large Neighborhood Search with Transfers (VLNS-T) algorithm to plan schedules for the most general version of the PDP-T, with constraints including time windows, capacities, vehicle start and end locations, maximum item transport times, and maximum vehicle route durations. We also contribute algorithms for simplified variants of the PDP-T, which take advantage of the problem structure to find solutions more quickly and more effectively than the general algorithm for specific PDP-T variants, some with provable guarantees. We also study the challenges of deploying PDP-T schedules on physical robots, and execute PDP-T schedules on the CoBots. The robots reschedule their tasks in response to new requests, delays, failures, and shared information from other robots. We also introduce the CreBots, which transfer items fully autonomously. Our PDP-T algorithms are evaluated on benchmark problems, on the CoBots, and on problems on city maps sampled from real world taxi data, demonstrating that lower cost schedules can be found with transfers.

# Acknowledgements

First and foremost, I would like to thank my advisor, Manuela Veloso, for everything she has done. Her enthusiasm is contagious, and it has been a joy to work with her for the past seven years.

I would also like to thank Howie Choset, Stephen Smith, and Daniele Nardi, for serving on my thesis committee. Their advice and support has been invaluable. I would especially like to thank Howie, whom I worked with as an undergraduate and who helped convince me to pursue a PhD in the first place.

My family has always supported me, for which I am constantly grateful, especially for my mother, father, and brother.

I am grateful for the help and support of my many friends and colleagues who have been part of the CORAL group over the years. I should especially thank Joydeep Biswas and Stephanie Rosenthal. Without their work on the CoBots this thesis could not have been possible. I am also thankful for the many wonderful experiences I had with my RoboCup teammates, especially Ryan Cahoon, Somchaya Liemhetcharat, Jinsu Liu, Çetin Meriçli, Misha Novitzky, Junyun Tay, Mike Phillips, Doug Vail, and Feng Xue. Thank you to the other members of the CORAL group who have enriched my life as well, particularly Aijun Bai, Susana Brandão, Tom Charley, Ben Choi, Fatma Faruq, Tom Kollar, Max Korein, JP Mendoza, Tekin Meriçli, Michael Murphy, Vittorio Perera, Prashant Reddy, Mehdi Samadi, Yichao Sun, Rodrigo Ventura, Richard Wang, and Danny Zhu. Thanks as well to the participants in the NSF CoBot group.

I am grateful for the many people I have had the pleasure of working with outside of CMU. I would particularly like to thank everyone I worked with at my internships at Aldebaran in Boston, at Intel Research Pittsburgh, and at the NASA Ames Intelligent Robotics Group. Special thanks go to Dean Pomerleau and Ara Nefian. I am also thankful for the many people I have met at RoboCup and at conferences.

I would like to thank my many other friends who made my time working on my PhD infinitely more enjoyable, including, but not only: Pranay Agrawal, Jeevan Bandreddi, Austin Buchan, Justin Chang, Tai Chen, Nam-Phuong Cong-Huyen, Felix Duvallet, Aram Ebtekar,

Siyuan Feng, Rushane Hua, Rich Hong, Junchen Jiang, Andrew Ko, Nan Li, Mun-Thye Mak, Chris Mar, Matthew Marge, Eugene Marinelli, Shelah Moreno, David Mowrey, Thorvald Natvig, Brad Neuman, Mark Qiao, Tamir Sen, Mark Shim, Jane Sun, Ming Sun, Glenn Wagner, Xuezhi Wang, Kevin Woo, Andrew Yeager, Kaya Yuki, Jason Zaman, Fan Zhang, and Kevin Zhao.

I am also grateful for everyone in the Kiltie Band and at the LUC, who made my time in Pittsburgh much more pleasant. Thank you as well to my many other friends in Pittsburgh, in Austin (especially my friends from TLLC) and elsewhere. I could not have made it alone.

# Contents


**1 Introduction** **1**

1.1 Thesis Approach . . . 3
1.2 Contributions . . . 4
1.3 Evaluation . . . 5
1.4 Document Outline . . . 5

**2 Pickup and Delivery Problems with Transfers** **7**

2.1 Background: The Pickup and Delivery Problem . . . 7
2.1.1 PDP Properties and Constraints . . . 7
2.1.2 Actions in a Schedule . . . 10
2.1.3 Determining Schedule Validity . . . 11
2.1.4 Objective Function . . . 12
2.1.5 Example PDP and Solution . . . 14
2.1.6 Difficulty of the PDP . . . 15
2.2 The PDP with Transfers . . . 16
2.2.1 Additional Transfer Actions . . . 16
2.2.2 Schedule Validity with Transfers . . . 17
2.2.3 The Cost of Transfers . . . 20
2.2.4 Example PDP-T Solution . . . 21
2.2.5 The Benefit of Transfers . . . 21
2.3 Variations on the PDP-T . . . 25
2.4 Online and Distributed PDP-T . . . 27
2.5 Chapter Summary . . . 27

**3 Task Scheduling and Execution on the CoBot Robots** **29**

3.1 The CoBot Robots . . . 30
3.2 Tasks for the CoBots . . . 31

3.3 Requesting Tasks . . . 32
3.4 Scheduling Tasks . . . 36
3.4.1 MIP Scheduling . . . 37
3.4.2 Illustrative Scheduling Example . . . 40
3.5 Executing and Managing Schedules . . . 41
3.5.1 Executing a Schedule . . . 41
3.5.2 Managing Task Lists with Interruptible Autonomy . . . 42
3.6 Monitoring Schedule Execution . . . 43
3.6.1 The Telepresence Interface . . . 45
3.6.2 Monitoring Multiple CoBots . . . 48
3.7 Selected Deployment Results . . . 48
3.8 Chapter Summary . . . 50

**4 The Pickup and Single Delivery Problem with Transfers 51**
4.1 The PSDP-T Problem . . . 52
4.2 PSDP-T Algorithms . . . 54
4.2.1 Optimal Approach . . . 54
4.2.2 Minimum Length Approximation . . . 56
4.2.3 Improvement with Local Search . . . 58
4.3 Transfers at Any Location . . . 60
4.4 Experimental Results . . . 61
4.4.1 Simulation Results . . . 61
4.4.2 Illustrative Deployments on CoBots . . . 63
4.4.3 Autonomous Transfers with CreBots . . . 63
4.5 Chapter Summary . . . 65

**5 Heuristics for the nTPDP-T 67**
5.1 Heuristics for the nTPDP without Transfers . . . 68
5.1.1 The Greedy Approach . . . 68
5.1.2 The Auction Approach . . . 69
5.2 Heuristics with Transfers for the nTPDP-T . . . 70
5.2.1 Finding a Transfer Point . . . 70
5.2.2 Splitting an Item's Route . . . 71
5.2.3 Greedy Transfer Insertion . . . 73
5.2.4 Transfer Insertion with Auctions . . . 74
5.2.5 Graph Search . . . 75

5.3 Selected Experimental Results . . . 78
5.3.1 The Euclidean Plane . . . 78
5.3.2 San Francisco . . . 80
5.4 Chapter Summary . . . 81

**6 Online Rescheduling with Transfers 85**
6.1 Auctions to Revise Schedules with Transfers Online . . . 86
6.1.1 The Online Auction Algorithm . . . 86
6.1.2 Online Scheduling and Rescheduling . . . 90
6.1.3 Experiments in Online Rescheduling . . . 91
6.2 Rescheduling with Shared Information . . . 94
6.2.1 Multi-Agent Rescheduling with Rationale Graphs . . . 95
6.2.2 Multi-Agent Rationale Sharing . . . 97
6.2.3 Multi-Agent Rescheduling . . . 98
6.2.4 Rescheduling on the CoBot Robots . . . 99
6.3 Chapter Summary . . . 102

**7 Very Large Neighborhood Search with Transfers 105**
7.1 The VLNS-T Algorithm . . . 106
7.1.1 Very Large Neighborhood Search . . . 106
7.1.2 Greedy Insertion with Transfers . . . 109
7.1.3 Determining Action Execution Times . . . 113
7.2 Experiments . . . 116
7.2.1 Example Problems . . . 116
7.2.2 Benchmark Problems . . . 117
7.2.3 New York Taxi Problems . . . 118
7.3 Chapter Summary . . . 119

**8 Background and Related Work 121**
8.1 Scheduling: The Vehicle Routing Problem . . . 121
8.1.1 The Traveling Salesman Problem . . . 121
8.1.2 The Vehicle Routing Problem with Backhauls . . . 122
8.1.3 The Pickup and Delivery Problem . . . 123
8.1.4 Dynamic Pickup and Delivery Problems . . . 124
8.1.5 Pickup and Delivery Problems with Transfers . . . 124
8.2 Related Robotics Research . . . 125

8.2.1 Pickup and Delivery Robots . . . 125
8.2.2 Task Allocation . . . 126
8.2.3 Ridesharing . . . 126
8.2.4 Robot Rendezvous . . . 127
8.2.5 Distributed Algorithms . . . 127

**9 Conclusion** **129**
9.1 Contributions . . . 129
9.2 Future Research Directions . . . 130
9.3 Concluding Remarks . . . 132

# List of Figures


1.1 An example PDP-T problem, in which robots pick up and deliver items, using *(a)* a single robot, *(b)* two robots without transfers, and *(c)* two robots with transfers. Using transfers lowers both the total distance travelled by the robots and the delivery time. . . . . 2

1.2 We evaluate PDP-T solutions in three main domains: on simulated benchmark problems, *(a)* on the roads, with autonomous taxis and ridesharing; and *(b)* in the office, delivering items with the CoBot and CreBot robots. . . . . 5

2.1 An example PDP problem with five locations connected by corridors of length 1. The squares represent vehicles, and the circles represent items. . . . . 14

2.2 The paths taken by the vehicles in the optimal schedule given in Table 2.3. . . . . 16

2.3 The paths taken by the vehicles in the optimal schedule given in Table 2.5. . . . . 22

2.4 A hub and spoke PDP. All $n$ items start at the same location as the vehicle $v_0$, and end at the locations shown. The distance $D >> 1$. . . . . 25

3.1 CoBot-1, CoBot-2, and CoBot-4: Collaborative service robots deployed in our buildings. (An additional CoBot-3 is deployed off campus.) . . . . . 30

3.2 In a pickup and delivery task, *(a)* one user places an item in CoBot's basket and *(b)* another user removes it when the item is delivered. In *(c)*, CoBot finds and retrieves a cup of coffee from the kitchen. . . . . 31

3.3 The system architecture of the CoBots. Users place requests through the dialog manager or a graphical UI on the robots, or directly to the web scheduler, which forms schedules for all the robots. . . . . 33

3.4 Users schedule tasks for CoBot on CoBot's web site. The user inputs correspond to the task request parameters in Table 3.1. . . . . 34

3.5 Users schedule tasks for CoBot on CoBot's touchscreen. The user inputs correspond to the task request parameters in Table 3.1. . . . . 35

3.6 Robot starting positions, task locations, and the generated plan. While executing the plan, CoBot-2 *(d)* retrieves an envelope, *(b)* delivers a spoken message, and *(a)* delivers the envelope. CoBot-1 $c$ escorts a visitor to an office before delivering a message. . . . . 41

3.7 A CoBot robot was interrupted by a passerby while executing a visit a room task. A new task is assigned to the CoBot and successfully scheduled for immediate execution. CoBot begins executing the new task and resumes the previous task when the new one is finished. . . . . 44

3.8 CoBot's web interface, with the Control and Map tabs visible. In the map tab, CoBot's current position, path and LIDAR readings are shown. Users may click on the map to set CoBot's localization position or travel to a point. . . . . 46

3.9 At left, CoBot-2 views a scene at the default zoom level. At right, the user zooms in to the boxed area at the maximum level of 18X. The powerful zoom functionality of the robot allows users to inspect small or distant objects, and even read text remotely. Only CoBot-2 possesses such a powerful camera. . . . . 47

3.10 Execution times for, from left to right, Deliver Message tasks, Go to Room tasks, and Transport tasks. The breakdown includes 1) waiting for help to start the task, 2) riding the elevator, 3) navigating (not including time blocked by obstacles), 4) waiting blocked by an obstacle, and 5) waiting for help to end the task. . . . . 50

4.1 Two vehicles collect letters and deliver them, minimizing the total distance travelled to conserve energy. Each corridor has length $c$. a Without transfers, the optimal solution has cost $4c$. b With transfers, the optimal cost is $3c$. . . . . 52

4.2 The optimal solution to the PSDP-T can be formulated as a delivery tree. Edges represent the motion of a single vehicle. Where the tree branches, all the vehicles at that branch point transfer their entire load to a single vehicle which continues alone. . . . . 54

4.3 Robots and items are situated at the end of $n$ hallways of length one emanating from the delivery point $f$. $O_n = n$, where every vehicle travels to $f$. $O_1 = 2n-1$, where one vehicle retrieves every item. . . . . 56

4.4 An example of the approximation algorithm. Solid black lines indicate edges on the minimum spanning tree, and dashed lines show the generated delivery tree. . 58

4.5 *(a)* The initial state. *(b)* A neighbor found by swapping $l_1$ and $l_2$ (the edges linking them to other nodes are different). *(c)* A neighbor with $l_2$ grafted from $l_1$ to $l_3$. . . . . 59

4.6 Results for the PSDP-T algorithms with *(a)* three robots and up to twenty items to retrieve under the total distance objective function in the planar domain, and *(b)* 25 robots and up to 175 items to retrieve under the mixed objective function in the office building domain. Lines indicate the mean objective function cost, and the filled area shows the standard deviation across trials. Darker areas indicate overlap. . . . . 62
4.7 The paths planned by the approximation algorithm and taken by the robots, with and without transfers, for a Scenario 1, b Scenario 2, and c Scenario 3. . . . . 64
4.8 a A CreBot robot, with a Create base, laptop computer, Kinect RGB-D camera, tilting tray, and QR code for alignment. b One CreBot transfers an item to another during a collection and delivery task. . . . . 64

5.1 The solution cost for each method in the Euclidean domain with $|V| = 20$, $\mathit{cap}(v) = 5$, $\mathit{mni}(v) = 7$, $\mathit{bud}(v) = 6$, and $c_t(m, v_1, v_2) = 0$. . . . . 79
5.2 The computation time for each method in the Euclidean domain with $|V| = 20$, $\mathit{cap}(v) = 5$, $\mathit{mni}(v) = 7$, $\mathit{bud}(v) = 6$, and $c_t(m, v_1, v_2) = 0$. . . . . 79
5.3 The number of transfers found for each method in the Euclidean domain with $|V| = 20$, $\mathit{cap}(v) = 5$, $\mathit{mni}(v) = 7$, $\mathit{bud}(v) = 6$, and $c_t(m, v_1, v_2) = 0$. . . . . 80
5.4 The solution cost found for each method in the Euclidean domain with $|M| = 10$, $\mathit{cap}(v) = 5$, $\mathit{mni}(v) = 7$, $\mathit{bud}(v) = 6$, and $c_t(m, v_1, v_2) = 0$. . . . . 81
5.5 An example solution with transfers for a problem in San Francisco. Two passengers are transferred to other vehicles. . . . . 82
5.6 The solution cost found for each method in San Francisco with $|V| = 18$, $C_v = 4$, $M_v = 7$, $B_v = 1.5$ km, and $c_T = 0$. . . . . 82

6.1 Vehicle $r$'s pickup and delivery of item $m$ is split with vehicle $s$ using `insert_transfer`. First, a transfer point is chosen between two subsequent tasks on each vehicle's plan. Then the delivery point is removed from $r$'s plan and inserted into $s$'s, lowering the delivery cost. . . . . 88
6.2 An example temporal network with two vehicles, three items and a single transfer. The feasible time windows for each action are computed based on the item time windows and the edge durations. . . . . 90
6.3 Planned schedules to deliver four items, scheduled with the *(a)* MIP without transfers, and *(b)* auction with transfers. Items 1 and 2 are requested at time 0, Item 3 is requested after 150 s, and Item 4 is requested after 200 s. See Fig. 6.1 for symbol meanings. . . . . 92

6.4 *(a)* Deliveries are scheduled with three robots (including two transfers). *(b)* When one of the robots dies and fails to respond, the tasks are rescheduled. See Fig. 6.1 for symbol meanings. . . . . 93
6.5 The mean cost of the generated schedules to the number of items. The shaded regions depict the standard deviation across the fifty trials. . . . . 94
6.6 a CoBot-2 heads to Office A to make a delivery. b CoBot-4 passes Office A and observes the door is closed. It notifies the server. The server sees a violation in CoBot-2's rationale graph and reschedules its tasks. c CoBot-2 is scheduled to complete a task at Office B first, and turns around. d CoBot-2 completes its task at Office B. e CoBot-4 returns from its previous task and observes that the door to Office A is now open. f CoBot-2 completes its original task at Office A. . . . . 101
6.7 a CoBot-2 heads towards an office to make a delivery. b CoBot-4 detects that a hallway is blocked, and tells the server. The server realizes that CoBot-2's rationale has been violated and communicates this information to CoBot-2. c CoBot-2 uses the information to change its plan. d CoBot-2 takes an alternate round to avoid the blocked hallway. . . . . 102
6.8 CoBot-2 replans its route after CoBot-4 reports that a hallway is blocked. . . . . 103

7.1 a The cost to deliver the items is $800$ without transfers. b The cost is halved to $400$ if transfers are allowed. Note that the order of the *Transfer* and *Receive* actions are arranged such that no deadlocking cycles are formed. . . . . 117

# List of Tables


2.1 PDP Notation . . . . . 8
2.2 Notation for Action $a$ . . . . . 10
2.3 An Optimal PDP Schedule, $c(S) = 8$ . . . . . 15
2.4 PDP-T Notation (New Notation is Bolded) . . . . . 18
2.5 An Optimal PDP Schedule, $c(S) = 4$ . . . . . 22
2.6 Variations on the PDP . . . . . 26

3.1 The fields in a task request. . . . . 33
3.2 An example specification of a Pickup and Delivery Request. . . . . 33
3.3 An example schedule. . . . . 40
3.4 Total number of task requests per task type and the respective number that used the elevator. . . . . 49

4.1 Deployment Results for Selected Scenarios . . . . . 63

5.1 Schedule Before Route Splitting . . . . . 71
5.2 Schedule After Route Splitting, $b$ Executes Action $R$ . . . . . 72
5.3 Schedule After Route Splitting, $b$ Executes Action $D$ . . . . . 72

7.1 VLNS Parameters . . . . . 108
7.2 Cordeau benchmark results. All times are in hours. The "Tabu" column gives the best results for Tabu search [33], the "VLNS" section for one run of VLNS [93], and the "VLNS-T" section for one run of our algorithm. The "Tr." column is the number of transfers. Computation times are in hours. . . . . 118
7.3 New York City Taxi Experiment Results . . . . . 119

8.1 Selected variants of the Vehicle Routing Problem. . . . . 122
8.2 Selected variants of the Pickup and Delivery Problem (PDP). . . . . 123

# Chapter 1

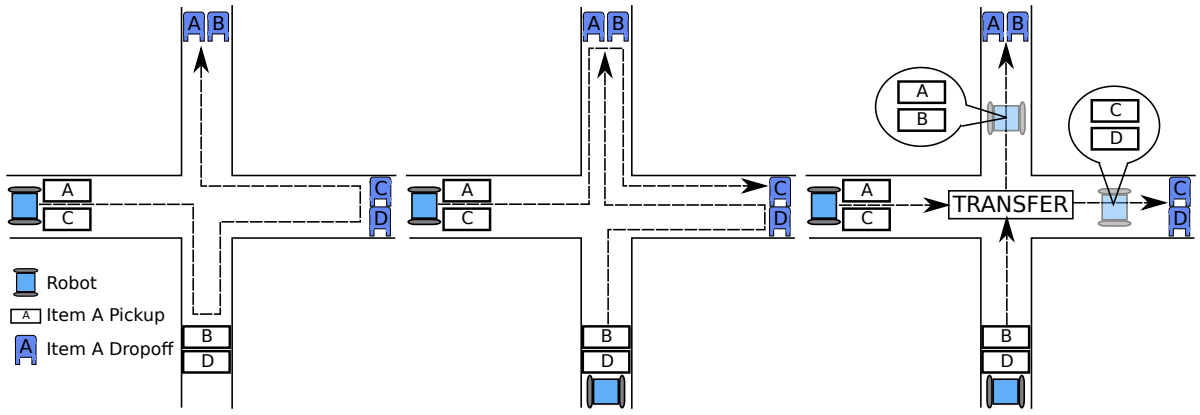
# Introduction

Many types of logistics problems require mobile vehicles to retrieve and deliver a set of objects. For example, taxis pick up and drop off passengers, mail services retrieve and deliver letters and packages, and airplanes deliver passengers and freight. All of these problems are instances of the Pickup and Delivery Problem (PDP), in which a set of mobile vehicles services a set of spatially-located requests. Given a set of object pickup and dropoff locations, along with a set of mobile vehicles, the goal is to find a schedule for the vehicles to minimize energy consumed or delivery time, under constrains such as time windows and vehicle capacities. The PDP is a well-studied, NP-hard problem, so approximation algorithms and heuristics have been developed to address variants of the PDP.

As autonomous mobile robots become increasingly capable and available, they offer new applications of the PDP. Our multiple CoBot robots currently deliver items, such as mail, autonomously to occupants of an office building in response to user requests. To deliver items as effectively as possible, algorithms are needed to address the PDP and form schedules of tasks for the robots. These schedules must account for robot capacities, time constraints, as well as additional constraints, and plans must be created and modified online in response to delays, cancellations, and new task requests.

While deploying the CoBots to pick up and deliver items for users, we realized that the robots' schedules could be improved if the robots **transferred** items between one another. For example, two robots could be assigned to pick up items from opposite ends of the seventh floor, and deliver them to the eighth floor. Both would have to go wait at the elevator before delivering their items on the eighth floor. But the schedule could be much more efficient if one robot passed its item to the other at the elevator, a single robot delivered both the items, and the other robot was free to do other tasks.

Inspired by the CoBots, we introduce and examine a novel extension to the PDP: allowing




(a) Distance: 8, Time: 8 (b) Distance: 12, Time: 6 (c) Distance: 4, Time: 4

Figure 1.1: An example PDP-T problem, in which robots pick up and deliver items, using *(a)* a single robot, *(b)* two robots without transfers, and *(c)* two robots with transfers. Using transfers lowers both the total distance travelled by the robots and the delivery time.

vehicles to *transfer* items to other vehicles before delivery. We call this problem the Pickup and Delivery Problem with Transfers (PDP-T). With transfers, one robot may pick up an object and transfer it a different robot (or a chain of robots) for delivery. Transferring items enables a reduction in fuel cost and delivery time for many problem instances.

Figure 1.1 shows an example of the benefits of transferring items. In Figure 1.1a, one robot picks up and then delivers all the items A, B, C and D. The robot travels six corridors of distance over a corresponding six units of time. In Figure 1.1b, a second robot is introduced. With two robots delivering the items, each picks up the items near its starting position and delivers one to each of the two delivery points. An increased eight units of distance are travelled, but delivery now only takes four units of time. Next, in Figure 1.1c we consider two robots that can transfer items. At the intersection of the two corridors, the robots swap items B and C. Now, each robot only needs to travel to the end of a single corridor to deliver both of its items. By transferring items, we reduce the total distance travelled to four units and the delivery time to two units, assuming the transfer itself takes negligible time. Transferring empowers us to make deliveries more efficiently because items heading in the same direction can be routed to the same vehicle, dividing the transportation cost between the items. While this simple example demonstrates the benefits of transferring items in the PDP-T, there are many algorithmic challenges that must be addressed to effectively plan for deliveries with transfers.

The principal question addressed in this thesis is:

> How can transfers be incorporated to improve schedules generated to solve Pickup and Delivery Problems, both offline and online, with multiple constraints, particu-

larly for deployment on robots?

This thesis builds upon the work of both the robotics community and the scheduling community, as we devise schedules with transfers and deploy them on actual robots. From a robotics perspective, the problem of transfers is especially interesting because, unlike in traditional multi-robot task allocation, the individual tasks are *tightly coupled*. The cost of a schedule with transfers depends highly on the combination of tasks assigned to all of the robots. Planning for transfers requires multi-robot coordination and cooperation in the fullest sense, rather than simply the execution of tasks independently in parallel. From a scheduling perspective, allowing transfers increases the space of possible schedules exponentially, but may lower the costs of the best solutions.

## 1.1 Thesis Approach

In practice in large scale logistics systems, such as the airline or shipping industries, items are transferred at central hubs which they are routed through in "hub and spoke" networks [24]. Our work differs in that we plan routes for each individual package rather than making use of fixed routes in economies of scale. Some previous approaches in the scheduling community have considered transferring items at a set of fixed "transshipment" points specified beforehand (e.g., [34, 80, 82, 78]). We allow individual items to be transferred multiple times at arbitrary locations generated based on the vehicles' routes, which increases the size of the search space but may allow for better solutions.

We contribute multiple novel algorithms to solve variants of the PDP-T, with numerous constraints, including capacities, time windows, maximum item transport times, and maximum vehicle route durations. We begin with simplifications of the full problem and develop heuristics, approximation algorithms and metaheuristics to solve the simplified problems. By developing specific algorithms for simplified problems, we are able to develop faster and more effective algorithms for the specific problems these algorithms address than the general algorithm which could solve any PDP-T which we do ultimately introduce. We build upon the earlier approaches to develop algorithms of increasing complexity and generality.

Since our work is motivated by a desire to execute schedules on robots, we address issues inherent to executing schedules on robots in the real world, which will inevitably differ from simulation. In most scheduling problems, it is assumed that a function exists which estimates the time taken to travel between two locations. In the real world, such a function is likely to both overestimate and underestimate the time in different situations, and exact execution of the schedule is not possible. Hence, we develop algorithms to replan with transfers in response

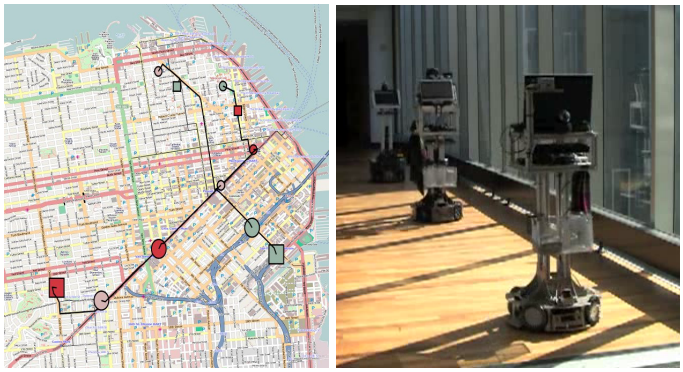
to delays. Vehicles will also sometimes fail, and our algorithms are able to reschedule failed vehicles items for delivery by other vehicles, with transfers. We also reschedule in response to shared information from other robots about traffic conditions and expected feasibility of task completion.

In this thesis, we mainly develop centralized algorithms, which rely on communication with a centralized server. We introduce a backup approach such that the robots can still execute locally requested tasks when communication fails. We also discuss distributed approaches which do not rely on a centralized server.

## 1.2 Contributions

Concretely, the main contributions of this thesis are:

- A formal definition of the PDP-T.
- An analysis of the potential benefits of transfers, including proofs bounding the maximum improvement in the objective function from transfers in certain cases.
- The online scheduling and execution of tasks on the CoBot robots, including interrupting the robots to modify schedules, and the remote monitoring and management of schedule execution.
- The introduction and implementation of the CreBot robots which autonomously transfer items.
- A two-approximate heuristic and a metaheuristic for the Pickup and Single Delivery Problem with Transfers (a simplified PDP-T variant), as well as a proven bound on the improvement from allowing transfers at any location.
- The introduction and comparison of multiple heuristics for a variant of the PDP-T without time windows.
- An auction-based heuristic to schedule PDP-Ts online. The heuristic reschedules in response to new requests, delays, and failures, and could be implemented in a distributed manner.
- The novel idea of multi-robot rescheduling by one robot from information observed and shared from other robots, and an algorithm to implement this idea.
- A metaheuristic to solve the general PDP-T, which outperforms state of the art PDP algorithms on benchmark problems.



(a) On the Roads (b) In the Office

Figure 1.2: We evaluate PDP-T solutions in three main domains: on simulated benchmark problems, *(a)* on the roads, with autonomous taxis and ridesharing; and *(b)* in the office, delivering items with the CoBot and CreBot robots.

## 1.3 Evaluation

We evaluate our algorithms for scheduling with transfers in three main domains:

1. **In simulation.** We evaluate our algorithms on generated problems with both Euclidean and Manhattan geometries. We show an improvement from using transfers on the Cordeau benchmark problems [33], which use Euclidean distance.
2. **On the roads.** Using maps of cities and publicly available taxi-cab data, we can recreate passenger demands in the city. With this data, we construct PDP-T problems and evaluate the effectiveness of solutions generated by our algorithms. On city maps, we have two particular use cases in mind: fleets of autonomous taxis without their own destinations, and a ridesharing service, such as Lyft [43], in which vehicles are driven by non-professional drivers with their own starting points and destinations who offer to pick up passengers along the way (see Fig. 1.2a).
3. **In the office.** We will deploy the CoBot and CreBot robots to execute schedules delivering mail or other items in the Gates-Hillman Center, and demonstrate that transferring items leads to faster delivery times at lower cost.

In each domain, we compare solutions with transfers to solutions without transfers, and demonstrate that our algorithms with transfers find better solutions.

## 1.4 Document Outline

The thesis is organized as follows:

- **Chapter 2** formally defines and introduces all aspects of the Pickup and Delivery Problem with Transfers, including our notation, the constraints and objectives of the PDP-T, and the actions and requirements of a valid schedule. The potential benefits of transfers and the difficulty of the PDP-T are analyzed. The variants of the PDP-T examined in this thesis our discussed.
- **Chapter 3** discusses how tasks are requested and executed on the CoBot robots. The idea of interruptible autonomy is introduced, with which the robots can be interrupted and new requests scheduled. This chapter also discusses how schedule execution can be monitored over the web and presents results from a public deployment of the CoBots.
- **Chapter 4** examines a simplified variant of the PDP-T, in which there are no time windows or capacities and all items are delivered to the same central location. A two-approximate heuristic and a metaheuristic are presented to solve this version of the problem, and results are presented from the CoBot and CreBot robots.
- **Chapter 5** presents three heuristics for a PDP-T with capacities and distinct item destinations but without time windows. The heuristics are compared on problems generated from taxi data on maps of San Francisco.
- **Chapter 6** examines the online PDP-T with time windows. An auction-based heuristic is introduced and evaluated on the CoBot robots. The heuristic is used to replan in response to new requests, delays, and robot failures, as well as shared information from other robots. The implementation of the heuristic in a distributed manner is discussed.
- **Chapter 7** presents a metaheuristic, VLNS-T, to solve the full PDP-T with all allowed constraints. The algorithm is shown to improve upon the best known results for PDP benchmark problems, and outperforms state-of-the-art PDP algorithms on problems generated from New York City taxi data.
- **Chapter 8** discusses areas of related research, including from the scheduling and robotics communities, and places this thesis into the context of that work.
- **Chapter 9** summarizes our major findings and suggests potential lines of future research.

# Chapter 2

# Pickup and Delivery Problems with Transfers

In this chapter, we contribute a formulation of the Pickup and Delivery Problem with Transfers (PDP-T), a novel generalization of the PDP. To do so, we first extensively discuss the space of pickup and delivery problems without transfers, explaining the many variants of PDPs and their different constraints. We specify how a solution to the PDP is formulated as a schedule, and discuss the objective functions a solution could seek to optimize. Then, we explain how adding transfers changes the problem and makes it more challenging. In this chapter, we do not introduce any algorithms or discuss *how* to solve the problem, we only describe the problem itself. The rest of the thesis discusses how to solve the problem.

## 2.1 Background: The Pickup and Delivery Problem

The goal in a PDP is to form a **schedule** for a set of vehicles $V$ to pickup and deliver a set of items $M$ while satisfying a set of **constraints** and minimizing an **objective function**. The constraints and objective function rely on a **distance function** which is estimated based on a map of the environment.

### 2.1.1 PDP Properties and Constraints

A PDP has numerous properties and constraints, both for the items and for the vehicles. A handy reference for our notation is given in Table 2.1.

Vehicle $v$ Properties

| Notation | Description |
|---|---|
| $\mathit{start}(v)$ | Start Location |
| $\mathit{end}(v)$ | End Location |
| $\mathit{cap}(v)$ | Capacity |
| $W_a(v)$ | Availability Window |
| $\mathit{mrd}(v)$ | Max. Route Duration |
| $d_v(a,b)$ | Distance Metric |

Item $m$ Properties

| Notation | Description |
|---|---|
| $\mathit{start}(m)$ | Start Location |
| $\mathit{end}(m)$ | End Location |
| $\mathit{dem}(m)$ | Demand |
| $W_p(m)$ | Pickup Time Window |
| $W_d(m)$ | Delivery Time Window |
| $\delta_{\mathtt{p}}(m)$ | Pickup Duration |
| $\delta_{\mathtt{d}}(m)$ | Delivery Duration |
| $\mathit{mtt}(m)$ | Max. Transport Time |
| $\mathit{AV}(m)$ | Allowed Vehicles |
| $\mathit{pri}(m)$ | Priority |

Table 2.1: PDP Notation

### Vehicle Properties and Constraints

For each vehicle $v \in V$, we are given a:

- **Start Location**: Vehicle $v$ must start at the location $\mathit{start}(v)$. This location may be a centralized depot shared by multiple vehicles, such as a charging station or taxi stand. Or the vehicles could each have different starting locations, such as vehicle drivers' homes or the current position of vehicles during operation.
- **End Location**: Vehicle $v$ must end at the location $\mathit{end}(v)$. End locations $\mathit{end}(v)$ always exist, but may be either a normal location or a special "wildcard" location $\emptyset$, in which case the vehicle may end at any location. The ending location may be a central depot the vehicle must start from and return to, or a different location than its starting point, such as the destination of the vehicle's driver.
- **Capacity**: Vehicle $v$ has capacity $\mathit{cap}(v)$. At any time, the sum of the demand from all items the vehicle is currently carrying cannot exceed $\mathit{cap}(v)$. Note that we can model uncapacitated problem formulations by setting $\mathit{cap}(v) = \infty$.
- **Vehicle Availability Time Window** Vehicle $v$ is only available for use during the specified time window $W_a(v)$. No actions may be scheduled outside this window. If the problem specifies no window and the vehicle is always available, then $W_a(v) = [-\infty, \infty]$.
- **Maximum Vehicle Route Duration** The maximum length of vehicle $v$'s route, from when it leaves its starting point until it returns to its ending point, cannot exceed the maximum route duration $\mathit{mrd}(v)$. This constraint could represent limited energy supply of a car or robot, or a limited availability of the vehicle's driver. If the constraint does not apply to a

problem, then $\mathit{mrd}(v) = \infty$.

### Item Properties and Constraints

The vehicles pickup and deliver a set of items $M$. Each item $m \in M$ is given a:

- **Pickup and Dropoff Location**: Item $m$ must be retrieved from the location $\mathit{start}(m)$ and delivered to the location $\mathit{end}(m)$.
- **Demand**: Item $m$ has demand $\mathit{dem}(m)$, which represents the amount of a vehicle's capacity taken up by an item. The sum of the demands of all items carried by the vehicle $v$ cannot exceed the capacity $\mathit{cap}(v)$ of the vehicle at any time. For example, in a transportation domain, an individual could have $\mathit{dem}(m) = 1$, while a family of four could have $\mathit{dem}(m) = 4$.
- **Pickup and Delivery Time Window**: Item $m$ must be delivered within the time window $W_p(m)$ and delivered within the time window $W_d(m)$. For time windows, we use the notation $W_p(m) = [ep(m), lp(m)]$. This time window means that item $m$ cannot be delivered before the time $ep(m)$ or delivered after the time $lp(m)$. The time windows may either be *hard* or soft. Hard time windows may never be violated, but for soft time windows, the delivery can take place after the deadline with a cost. In either case, if the vehicle arrives before $ep(m)$, it must wait until $ep(m)$ to make the delivery. For the remainder of this document, hard time windows are assumed unless stated otherwise.
- **Item Management Time**: Item $m$ has an associated pickup duration $\delta_{\mathtt{p}}(m)$ and delivery duration $\delta_{\mathtt{d}}(m)$. These are the times it takes to load and unload the item, respectively.
- **Maximum Item Transport Time**: The time between when the item $m$ is picked up and when it is delivered cannot exceed the maximum transportation time $\mathit{mtt}(m)$. This constraint could represent couriers delivering packages with deadlines, or human passengers with limited patience. If an item does not have a maximum transportation time, then $\mathit{mtt}(m) = \infty$.
- **Allowed Item Transport Vehicles**: The item $m$ can only be transported by vehicles in the set $\mathit{AV}(m)$. This constraint applies if any passengers or items have specific delivery requirements that can only be fulfilled by certain vehicles. If item $m$ can be transported by any vehicle, then $\mathit{AV}(m) = V$.
- **Priority**: The item $m$ has an associated delivery priority $\mathit{pri}(m)$, where the delivery of items with higher priority is prioritized. The priority will later be included in the objective function. In this thesis, unless otherwise specified, all items are assumed to have equal

priority.

In addition, a PDP must specify a **distance metric** $d_v(a, b)$ that returns an estimate of the time vehicle $v$ takes to travel from location $a$ to location $b$, where locations $a$ and $b$ are any locations on a map. The distance function may be Euclidean distance, Manhattan distance, or taken from a city street map. It can be estimated based purely on physical distance or learned from experience. The distance function is utilized both to determine the time vehicles take to travel between locations and to determine the cost of that motion in the objective function.

### 2.1.2 Actions in a Schedule

The goal of the PDP is to form a *schedule*, $S$, where $S = \cup_{v \in V} S^v$ is a set of schedules for all the vehicles. For each vehicle $v \in V S^v = (S_1^v, S_2^v, \ldots, S_n^v)$ is the sequence of actions $S_i^v \in A$ executed in order by vehicle $v$ in schedule $S$.

Every action $a \in A$ occurs at a location $\mathit{loc}(a)$. The vehicle arrives at $\mathit{loc}(a)$ at time $b(a)$, and executes $a$ at time $t(a)$. The action takes time $\delta(a)$ to execute. Note that the arrival time and execution time are not necessarily the same, and the vehicle may wait at the destination. Waiting is often necessary to satisfy time window constraints. For convenience, we say that $t(a)$ must fall within the time interval $W(a) = [e(a), l(a)]$, which is the appropriate time window for the given action type. If no time window constrains a given action, the window may be $(-\infty, \infty)$. The notation for actions in a schedule is presented in Table 2.2.

| Notation | Description |
|---:|---|
| $\mathit{loc}(a)$ | Location |
| $b(a)$ | Arrival Time |
| $t(a)$ | Execution Time |
| $\delta(a)$ | Execution Duration |
| $W(a) = [e(a), l(a)]$ | Allowed Time Window |

Table 2.2: Notation for Action $a$

The possible action types in $A$ are:

- START: The first action of every plan. If $a = S_i^v =$ START, then $\mathit{loc}(a) = \mathit{start}(v)$, $W(a) = W(v)$, and $\delta(a) = 0$.
- END: The final action of every plan. If $a = S_i^v =$ END, then $\mathit{loc}(a) = \mathit{end}(v)$, $W(a) = W(v)$, and $\delta(a) = 0$. If $\mathit{end}(v) = \emptyset$, then $\mathit{loc}(a)$ is the same as the previous action.
- PICKUP $(m)$: Pick up item $m$. If $a = S_i^v =$ PICKUP $(m)$, then $\mathit{loc}(a) = \mathit{start}(m)$, $W(a) = W_p(m)$, and $\delta(a) = \delta_{\mathrm{p}}(m)$. However, when using soft time windows, $W(a) = [ep(m), \infty)$.

- DELIVER$(m)$: Deliver item $m$. If $a = S_i^v = \text{DELIVER}(m)$, then $\mathit{loc}(a) = \mathit{end}(m)$, $W(a) = W_d(m)$, $\delta(a) = \delta_{\mathtt{d}}(m)$.

The START and END actions are not strictly necessary, since they are the same in every valid schedule. However, their inclusion makes the problem and various algorithms easier to formulate.

### 2.1.3 Determining Schedule Validity

To be considered valid, a schedule $S$ must contain only the action types mentioned above, and satisfy the following conditions:

1. **Start and End Actions**: Every vehicle's schedule must begin with a START action and end with an END action.

$$\forall v \in V: \; S_1^v = \text{START}, S_{|S^v|}^v = \text{END} \tag{2.1}$$

2. **Single Pickup and Delivery**: Every item is picked up and delivered at most once. However, the schedule is still valid if some items are not delivered at all.

$$\forall m \in M: \; |\{S_i^v \in S : S_i^v = \text{PICKUP}(m)\}| = |\{S_i^v \in S : S_i^v = \text{DELIVER}(m)\}| \leq 1 \tag{2.2}$$

3. **Delivery After Pickup**: Every item that is picked up must be delivered later by the same vehicle.

$$\forall S_i^v \in S \; s.t. \; S_i^v = \text{PICKUP}(m) : \exists j > i \; s.t. \; S_j^v = \text{DELIVER}(m) \tag{2.3}$$

4. **Feasible Travel Times**: It must be possible for a vehicle to arrive in time to execute each action according to the estimated distance function.

$$\forall v \in V, 1 \leq i < |S^v| : \; t(S_i^v) + \delta(S_i^v) + d_v\left(\mathit{loc}(S_i^v), \mathit{loc}(S_{i+1}^v)\right) \leq b\left(S_{i+1}^v\right) \leq t\left(S_{i+1}^v\right) \tag{2.4}$$

5. **Time Constraints**: Each action must be executed in the specified time window. These ensure that all pickup and delivery time window constraints as well as vehicle availability time windows are satisfied, since these time windows are both encoded in the actions' time windows $W(S_i^v)$. Each action must *begin* execution within the requested time window, but is not necessarily required to finish within that time window since each action may

have a non-zero duration.

$$\forall S_i^v \in S : \; \boldsymbol{t}\left(S_i^v\right) \in \; W\left(S_i^v\right) \tag{2.5}$$

6. **Capacity Constraints**: The demand of the items inside a vehicle may never exceed the vehicle's capacity.

$$\forall v \in V, 1 \leq i < |S^v| : \; \sum_{j=1}^{i} \boldsymbol{ad}\left(S_j^v\right) \leq \boldsymbol{cap}(v) \tag{2.6}$$

$$\text{where } \boldsymbol{ad}(a) = \begin{cases} \boldsymbol{dem}(m) & \text{if } a = \textsc{Pickup}\,(m) \\ -\boldsymbol{dem}(m) & \text{if } a = \textsc{Deliver}(m) \\ 0 & \text{otherwise} \end{cases}$$

7. **Maximum Vehicle Route Duration Constraints**: The length between each vehicle's START and END actions cannot exceed the maximum route duration.

$$\forall v \in V : \; \boldsymbol{t}\left(S_{|S^v|}^v\right) - \boldsymbol{t}\left(S_1^v\right) \leq \boldsymbol{mrd}\left(v\right) \tag{2.7}$$

8. **Maximum Item Transportation Time Constraints**: An item cannot be in transport for more than the maximum transportation time.

$$\forall S_i^v, S_j^v \in S \; s.t. \; S_i^v = \textsc{Pickup}\,(m)\,, S_j^v = \textsc{Deliver}(m) : \boldsymbol{t}\left(S_j^v\right) - \boldsymbol{t}\left(S_i^v\right) \leq \boldsymbol{mtt}(m) \tag{2.8}$$

9. **Allowed Vehicle Constraints**: Each item can only be transported by vehicles in its allowed vehicle set.

$$\forall S_i^v \in S \; s.t. \; S_i^v = \textsc{Pickup}\,(m) : v \in \boldsymbol{AV}(m) \tag{2.9}$$

### 2.1.4 Objective Function

The goal when solving a PDP is not simply to find any valid schedule, but to find the schedule that minimizes some objective function. In this thesis, we consider minimizing four main objectives:

- **Total Distance Traveled**: The total distance travelled by all the vehicles roughly corresponds to the energy required to complete the pickup and delivery tasks. We wish to minimize energy usage to conserve vehicle fuel or robot battery life:

$$TD(S) = \sum_{v \in V} \sum_{i=1}^{|S^v|-1} \boldsymbol{d}_v\left(S_i^v, S_{i+1}^v\right).$$

- **Total Vehicle Operation Time**: The total distance travelled metric does not give any cost to the time a vehicle spends idling. In reality, the time vehicles spend idling incurs costs as well, since the vehicle's driver must still be paid and robots will still consume battery life. To account for these costs, a second objective weighs the total time the vehicles are in operating, between the initial START action and the final END action:

$$OT(S) = \sum_{v \in V} \left( t\left(S^v_{|S^v|}\right) - t\left(S^v_1\right) \right).$$

- **Undelivered Items Penalty**: A schedule may be valid even if it doesn't deliver every item in $M$. For some problems, there may even be no valid solution which delivers all the items. So rather than requiring that all items be delivered, we set a penalty for undelivered items in the objective function. This penalty is the item's priority, $pri\,(m)$. We define the set of undelivered items in the schedule $S$ as

$$UND(S) = \{m \in M : \nexists \text{DELIVER}(m) \in S\}.$$

- **Soft Time Window Violation**: If the time windows are soft, they may be violated at a cost. In this thesis, when dealing with soft time windows, we apply a linear late delivery fee,

$$LD(S,m) = \begin{cases} 0 & \text{if } m \in UND(S) \\ \min\left(0, t\left(\text{PICKUP}\left(m\right)\right) - ld\left(m\right)\right) & \text{otherwise} \end{cases},$$

  however, a different late delivery fee could be applied instead. For example, a quadratic late fee would apply a hefty cost to individual items which are delivered especially late, as this could be considered worse than many items being delivered a little bit late. The total soft time window violation cost is

$$SW(S) = \sum_{m \in M} LD(S,m).$$

The final cost function to minimize is the weighted sum of all the objectives,

$$c\left(S\right) = \alpha TD(S) + \beta OT(S) + \gamma \sum_{m \in UND(S)} pri\left(m\right) + \delta SW(S).$$

Typically $\gamma$ will be set to a high value, such that delivering as many items as possible is prioritized over the other objectives. If the time windows are hard, then $\delta = 0$.

There are other objectives which could be used for the PDP. For example, we could try to complete schedule execution as quickly as possible by minimizing the makespan of the schedule,

$$\max_{v \in V} t\left(S^v_{|S_v|}\right) - \min_{v \in V} t\left(S^v_1\right).$$

Or, if the items should not be in the vehicle for long (e.g., delivering hot pizza, impatient passengers) a term could be added to minimize the time the items spend in the vehicles,

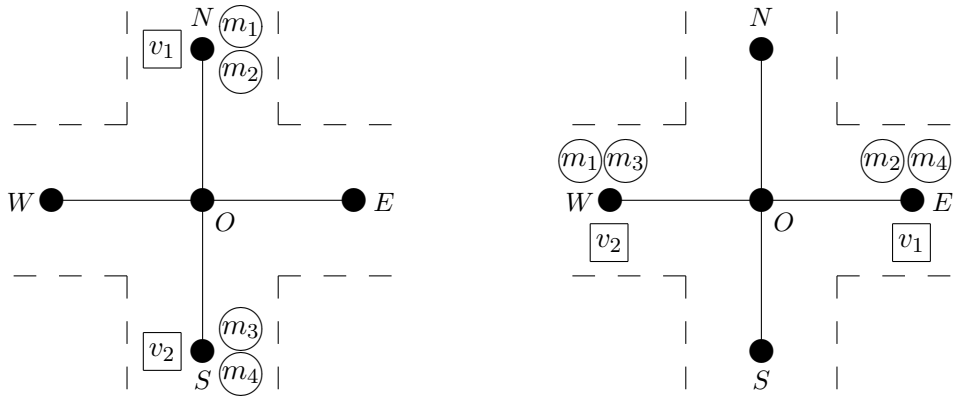
$$\sum_{m \in M \setminus \mathit{UND}(S)} \left(t\left(\textsc{Deliver}(m)\right) - t\left(\textsc{Pickup}\left(m\right)\right)\right)^2.$$

However, in this thesis our main focus is minimizing the objectives in $c\left(S\right)$.

### 2.1.5 Example PDP and Solution

In this section, we give an example of a PDP problem and show the optimal solution. We have two vehicles, $V = \{v_1, v_2\}$, and four items, $M = \{m_1, m_2, m_3, m_4\}$. The map contains five locations, $N$, $E$, $S$, $W$ and $O$, connected in a cross by corridors of length 1 which defines the distance functions $d_v(a, b)$.

The two vehicles start at the $N$ and $S$ locations ($\mathit{start}(v_1) = N, \mathit{start}(v_2) = S$), and end at the $E$ and $W$ locations ($\mathit{end}(v_1) = E, \mathit{end}(v_2) = W$). Two items start at each of the vehicle start locations ($\mathit{start}(m_1) = \mathit{start}(m_2) = N$, $\mathit{start}(m_3) = \mathit{start}(m_4) = S$. For two items with the same starting location, one has a destination at $W$ and one has a destination at $E$ ($\mathit{end}(m_1) = \mathit{end}(m_3) = W$, $\mathit{end}(m_2) = \mathit{end}(m_4) = E$). See Figure 2.1 for the problem setup.




(a) Starting State (b) Goal State

Figure 2.1: An example PDP problem with five locations connected by corridors of length 1. The squares represent vehicles, and the circles represent items.

We are given the hard time windows $\forall m \in M\ W_p(m) = [0, 5),\ W_d(m_1) = W_d(m_4) = [3, 12],\ W_d(m_2) = W_d(m_3) = [8, 12]$. Picking up and delivering objects each take one unit of time, and $\forall m \in M\ \delta_{\mathrm{p}}(m) = \delta_{\mathrm{d}}(m) = 1$. In this simple example, no other constraints are given: $\forall v \in V$ $\mathit{cap}(v) = \infty$, $W_a(v) = [0, \infty)$, $\mathit{mrd}(v) = \infty$, and $\forall m \in M$ $\mathit{mtt}(m) = \infty, \mathit{AV}(m) = V$. The goal is to minimize the total distance travelled by the vehicles, so $\alpha = 1, \beta = 0, \gamma = 10^6$ (and since the time windows are hard, $\delta = 0$).

Table 2.3 shows one example of an optimal schedule for this problem, illustrated in Figure 2.2. Each vehicle picks up the items nearest to its starting location, and delivers them both, while delivering the item nearest to its destination last. Note that the optimal solution to this problem is *not* unique. Another possible solution has $v_1$ pick up $m_2$, then $m_4$, and deliver both, while $v_2$ picks up $m_3$, then $m_1$ and deliver both. This schedule also has cost 8.

| $S$ | Action $a$ | $\mathit{loc}(a)$ | $b(a)$ | $t(a)$ | $\delta(a)$ |
|---|---|---|---|---|---|
| $S_1^{v_1}$ | START | N | 0 | 0 | 0 |
| $S_2^{v_1}$ | PICKUP $(m_1)$ | N | 0 | 0 | 1 |
| $S_3^{v_1}$ | PICKUP $(m_2)$ | N | 1 | 1 | 1 |
| $S_4^{v_1}$ | DELIVER$(m_1)$ | W | 4 | 4 | 1 |
| $S_5^{v_1}$ | DELIVER$(m_2)$ | E | 7 | 8 | 1 |
| $S_6^{v_1}$ | END | E | 9 | 9 | 0 |
| $S_1^{v_2}$ | START | S | 0 | 0 | 0 |
| $S_2^{v_2}$ | PICKUP $(m_3)$ | S | 0 | 0 | 1 |
| $S_3^{v_2}$ | PICKUP $(m_4)$ | S | 1 | 1 | 1 |
| $S_4^{v_2}$ | DELIVER$(m_4)$ | E | 4 | 4 | 1 |
| $S_5^{v_2}$ | DELIVER$(m_3)$ | W | 7 | 8 | 1 |
| $S_6^{v_2}$ | END | W | 9 | 9 | 0 |

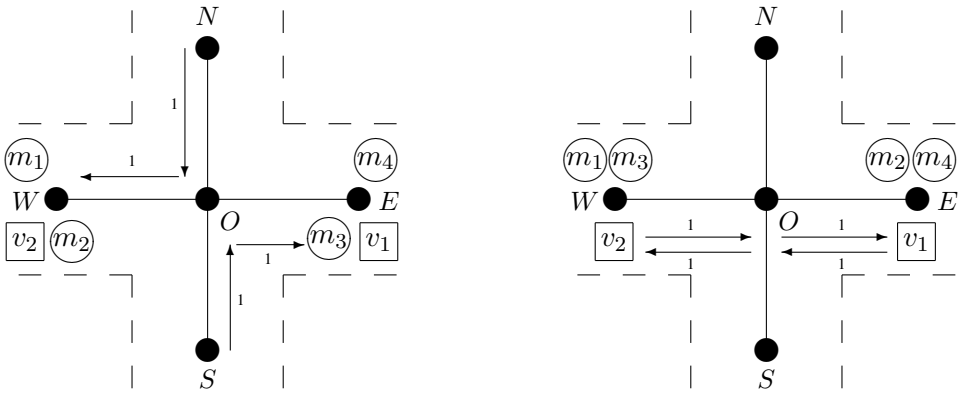
Table 2.3: An Optimal PDP Schedule, $c(S) = 8$

### 2.1.6 Difficulty of the PDP

**Theorem 1.** *The PDP is $\mathcal{NP}$-hard.*

*Proof.* We prove that the PDP is $\mathcal{NP}$-hard by reduction of the Traveling Salesman Problem (TSP) to the PDP. In the TSP, a single salesman begins at a location $l_0$, travels to a set of locations $l_1, l_2, \ldots, l_n$ to sell items, and finally returns to his starting location $l_0$. The goal is to find the ordering of locations to visit which minimizes the distance travelled by the salesman.

We construct a PDP with a single vehicle $v$ and a set of $n$ items $M = \{m_1, m_2, \ldots, m_n\}$. The vehicle must start and end at $l_0$, so $\mathit{start}(v) = \mathit{end}(v) = l_0$. All items are picked up from $s$ and delivered to the locations $l_1, l_2, \ldots, l_n$, so $\forall m_i \in M : \mathit{start}(m_i) = s, \mathit{end}(m_i) = l_i$. The objective




(a) Making First Deliveries (b) Making Final Deliveries

Figure 2.2: The paths taken by the vehicles in the optimal schedule given in Table 2.3.

is to minimize the distance ($\alpha = 1, \beta = 0, \gamma \to \infty$) and there are no capacities ($cap(v) = \infty$), time windows ($\forall m \in M \;\; W_p(m) = W_d(m) = [0, \infty)$) nor any other constraints. The optimal solution to this PDP gives the optimal solution to the TSP. Thus, the TSP reduces to the PDP in polynomial time, and since the TSP is $\mathcal{NP}$-hard, the PDP is also $\mathcal{NP}$-hard. □

Since the PDP is $\mathcal{NP}$-hard, solving large PDPs optimally is impractical. Therefore a large body of research, including much of this thesis, focuses on developing approximation algorithms, heuristics, and metaheuristics for the PDP.

## 2.2 The PDP with Transfers

The main contribution of this thesis is the introduction of **transfers** to the Pickup and Delivery Problem. We call this the Pickup and Delivery Problem with Transfers, or the **PDP-T**. In this section, we present exactly how the PDP-T is different from the PDP. To incorporate transfers, we introduce two **additional actions** to the schedule, and an extra term to the objective function. As before, we only discuss the problem itself, not any algorithms to find solutions.

### 2.2.1 Additional Transfer Actions

Previously, in the PDP, a single vehicle picked up an item and transported it all the way to its destination. Now, one vehicle may pick up an item, meet up with another vehicle, and transfer the item to that vehicle to complete the delivery.

We include transfers in the schedule by defining two additional actions:

**Definition 1.** *A **transfer** is a tuple $(T, R)$ of two paired actions in $S$, where $T = \text{TRANSFER}(v_2, m, l)$, $R = \text{RECEIVE}(v_1, m, l))$, and $T \in S^{v_1}, R \in S^{v_2}$. In this transfer, vehicle $v_1$ transfers the item $m$ to vehicle $v_2$ at location $l$. For every transfer pair, $\text{loc}(T) = \text{loc}(R) = l$, $W(T) = W(R) = [ep(m), ld(m)]$, and $\delta(T) = \delta_{\mathtt{t}}(m, v_1, v_2)$. For convenience, to denote paired actions in a single transfer, we define $\text{pair}(T) = R$ and $\text{pair}(R) = T$.*

An item may be transferred multiple times through a chain of transfers. When two vehicles meet up for a transfer, they may also transfer or exchange multiple items. These multiple transfers are modeled by multiple sequential RECEIVE and TRANSFER actions at the same location.

Unlike in hub and spoke networks, vehicles may not drop off the item and leave, but must be physically present when the vehicle receiving the item arrives in order to transfer the item. So for a TRANSFER or RECEIVE action $a$ (henceforth referred to simply as 'a transfer action'), $t(a) = t(\text{pair}(a))$, but it is not necessarily the case that $b(a) = b(\text{pair}(a))$. The PDP-T is a fundamentally different problem than hub and spoke networks, since each item must be planned for individually and the vehicles must be tightly coordinated.

Adding two possible actions to the vehicles' plans greatly increases the size of the search space. Furthermore, the transfer actions are parameterized with the location, and are not limited to pre-specified locations as with item pickups and deliveries, but can potentially take place at any location. The potential transfer locations depend on the problem domain. There may be a fixed set of locations where transfers can occur, or transfers can take place at any point in a region on a map.

For convenience, the notation for PDP-T problems is given in Table 2.4.

### 2.2.2 Schedule Validity with Transfers

When transfers are added, the constraints to determine a valid schedule also change. To be considered valid, a schedule $S$ must contain only the action types specified, and satisfy the following conditions. Conditions (2.3) and (2.6) from Section 2.1.3 are replaced, while the rest of the constraints from Section 2.1.3 still apply.

1. **Delivery After Pickup**: Every item that is picked up must be delivered. These constraints replace the constraints from (2.3) in the earlier section. It may be delivered by a different vehicle after a sequence of transfers. Each PICKUP or RECEIVE action must be followed by a DELIVER or TRANSFER action and vice-versa. In our formulation, an item may be transferred to the same vehicle at most once. Typically, transferring back and forth to the same vehicle multiple times would be inefficient, however, there could be unusual cases where such a behavior would be useful, for example, in order to obey capacity constraints

Vehicle $v$ Properties

| Notation | Description |
|---|---|
| $\mathit{start}(v)$ | Start Location |
| $\mathit{end}(v)$ | End Location |
| $\mathit{cap}(v)$ | Capacity |
| $W_a(v)$ | Availability Window |
| $\mathit{mrd}(v)$ | Max. Route Duration |
| $d_v(a, b)$ | Distance Metric |

Item $m$ Properties

| Notation | Description |
|---|---|
| $\mathit{start}(m)$ | Start Location |
| $\mathit{end}(m)$ | End Location |
| $\mathit{dem}(m)$ | Demand |
| $W_p(m)$ | Pickup Time Window |
| $W_d(m)$ | Delivery Time Window |
| $\delta_{\mathtt{p}}(m)$ | Pickup Duration |
| $\delta_{\mathtt{d}}(m)$ | Delivery Duration |
| $\delta_{\mathtt{t}}(m, v_1, v_2)$ | **Transfer Duration** |
| $\mathit{mnt}(m)$ | **Maximum Transfers** |
| $\mathit{mtt}(m)$ | Max. Transport Time |
| $\mathit{AV}(m)$ | Allowed Vehicles |
| $\mathit{pri}(m)$ | Priority |

Action $a$ Notation

| Notation | Description |
|---|---|
| $\mathit{loc}(a)$ | Location |
| $b(a)$ | Arrival Time |
| $t(a)$ | Execution Time |
| $\delta(a)$ | Execution Duration |
| $W(a)$ | Allowed Time Window |
| $e(a)$ | Earliest Time |
| $l(a)$ | Latest Time |
| $\mathit{pair}(a)$ | **Paired Transfer Action** |

Action Types

| Name | Description |
|---|---|
| START | Start Vehicle Operation |
| END | End Vehicle Execution |
| PICKUP$(m)$ | Pickup Item |
| DELIVER$(m)$ | DeliverItem |
| TRANSFER$(m, v, l)$ | **Transfer** $m$ **to** $v$ **at** $l$ |
| RECEIVE$(m, v, l)$ | **Receive** $m$ **from** $v$ **at** $l$ |

Table 2.4: PDP-T Notation (New Notation is Bolded)

by temporarily exchanging an item with another vehicle.

$$\begin{aligned} \forall S_i^v \in S \text{s.t. } S_i^v = \textsc{Pickup}(m) \text{ or } S_i^v = \textsc{Receive}(v_1, m, l_1) : \\ \exists \text{ a unique } j > i \text{ s.t. } S_j^v = \textsc{Deliver}(m) \text{ or } S_j^v = \textsc{Transfer}(v_2, m, l_2) \end{aligned} \tag{2.10}$$

$$\begin{aligned} \forall S_i^v \in S \text{s.t. } S_i^v = \textsc{Deliver}(m) \text{ or } S_i^v = \textsc{Transfer}(v_1, m, l_1) : \\ \exists \text{ a unique } j < i \text{ s.t. } S_j^v = \textsc{Pickup}(m) \text{ or } S_j^v = \textsc{Receive}(v_2, m, l_2) \end{aligned} \tag{2.11}$$

2. **Capacity Constraints**: The demand of the items inside a vehicle may never exceed the vehicle's capacity. These constraints replace constraint (2.6) from the previous section to incorporate the change in available capacity from transfer actions.

$$\forall v \in V, 1 \leq i < |S^v| : \sum_{j=1}^{i} \mathit{ad}\left(S_j^v\right) \leq \mathit{cap}(v) \tag{2.12}$$

$$\text{where } \mathit{ad}(a) = \begin{cases} \mathit{dem}(m) & \text{if } a = \textsc{Pickup}(m) \text{ or } a = \textsc{Receive}(v, m, l) \\ -\mathit{dem}(m) & \text{if } a = \textsc{Deliver}(m) \text{ or } a = \textsc{Transfer}(v, m, l) \\ 0 & \text{otherwise} \end{cases}$$

3. **Paired Transfer Actions**: Every RECEIVE action is paired with a TRANSFER action with the same time, vehicles, and location.

$$\begin{aligned} \forall S_i^{v_1} \in S \; s.t. \; S_i^{v_1} = \textsc{Transfer}(v_2, m, l) : \exists S_j^{v_2} = \textsc{Receive}(v_1, m, l) \; s.t. \\ \mathit{pair}(S_i^{v_1}) = S_j^{v_2}, \mathit{pair}\left(S_j^{v_2}\right) = S_i^{v_1}, \text{and } t(S_i^{v_1}) = t\left(S_j^{v_2}\right) \end{aligned} \tag{2.13}$$

$$\begin{aligned} \forall S_i^{v_1} \in S \; s.t. \; S_i^{v_1} = \textsc{Receive}(v_2, m, l) : \exists S_j^{v_2} = \textsc{Transfer}(v_1, m, l) \; s.t. \\ \mathit{pair}(S_i^{v_1}) = S_j^{v_2}, \mathit{pair}\left(S_j^{v_2}\right) = S_i^{v_1}, \text{and } t(S_i^{v_1}) = t\left(S_j^{v_2}\right) \end{aligned} \tag{2.14}$$

4. **No Deadlock**: Since each vehicle must execute the actions in its schedule in order, it is possible for a schedule to have deadlocks with cycles of transfer actions. For example, consider a schedule with the actions $S_i^v = \textsc{Transfer}(v', m, l)$, $S_{i+1}^v = \textsc{Receive}(v', m', l)$, $S_j^{v'} = \textsc{Receive}(v, m', l)$, $S_{j+1}^{v'} l$. When this schedule is executed, vehicle $v$ will attempt to send item $m$ to $v'$, while $v'$ attempts to receive item $m'$ from $v$. Both vehicles will wait on the other forever to do so, and the schedule deadlocks. Deadlocks are not only possible pairwise between vehicles, but also may occur if there is a cycle of dependencies between multiple vehicles.

   To detect deadlock, we define a directed graph, $G(S)$, with its node set $N(G(S))$ and edge

set $E(G(S))$, representing the schedule $S$. Every action in $S$ is a node in $G(S)$. The graph contains an edge from every action in the schedule to its successor, i.e. edges from $S_i^v$ to $S_{i+1}^v$. In addition, $G(S)$ contains an edge from every TRANSFER action to its pair, and from every RECEIVE action to its pair.

$$\begin{aligned} N(G(S)) &= \bigcup_{v \in V} \{a \in S^v\} \\ E(G(S)) &= \left( \bigcup_{v \in V} \left\{ (S_i^v, S_{i+1}^v) : 1 \leq i < |S^v| \right\} \right) \cup \\ &\quad \left\{ (S_i^v, S_j^{v'}) : S_i^v = \textsc{Transfer}(v', m, l), S_j^{v'} = \textit{pair}(S_i^v) \right\} \cup \\ &\quad \left\{ (S_i^v, S_j^{v'}) : S_i^v = \textsc{Receive}(v', m, l), S_j^{v'} = \textit{pair}(S_i^v) \right\} \end{aligned}$$

There is a deadlock in $S$ iff $G(S)$ has a cycle of length greater than two. Cycles of length two are discounted because every pair of transfer actions forms a cycle of length two, which does not cause deadlock.

$$G(S) \text{ has no cycles of length greater than two.} \tag{2.15}$$

5. **Maximum Number of Transfers**: For certain types of problems (i.e., with human passengers) transfers may be very inconvenient and so the maximum number of transfers for any given item $m$ is limited to be at most $\textit{mnt}(m)$. Unless otherwise stated, in this thesis we assume that $\textit{mnt}(m) = \infty$.

$$|\{S_i^v \in S : S_i^v = \textsc{Transfer}(v', m, l)\}| \leq \textit{mnt}(m)$$

With these few modified and additional constraints, we are equipped to determine whether a schedule with transfers is valid.

### 2.2.3 The Cost of Transfers

Adding transfers can reduce the distance travelled by the vehicles and the time taken to deliver the items. However, each transfer has a cost to perform, both in time and in energy.

To account for the cost in time, we set a time taken to make a transfer for an item $m$, as $\delta_{\mathtt{t}}(m, v_1, v_2)$. This duration is the time taken to perform the transfer, which is a function of the item and the vehicles performing the transfer.

Furthermore, there is a cost in energy and labor from transferring items. Fuel may be con-

sumed while the vehicles are idling, the drivers must physically labor to transfer the items, passengers may be inconvenienced, or transferring may drain a robot's battery. To account for these costs, we add a new term to the objective function. If $c_t(m, v_1, v_2)$ is the cost of transferring item $m$ from $v_1$ to $v_2$, then the new term in the objective function is

$$TC(S) = \sum_{v_1 \in V} \left( \sum_{\text{TRANSFER}(v_2, m, l) \in S^{v_1}} c_t(m, v_1, v_2) \right)$$

and the final objective is

$$\mathbf{c}(S) = \alpha TD(S) + \beta OT(S) + \gamma |\mathbf{UND}(S)| + \delta SW(S) + TC(S).$$

The final term for the cost of transfers does not require a weighting factor since this can be incorporated into the $c_t(m, v_1, v_2)$ function.

### 2.2.4 Example PDP-T Solution

As a running example, we return to the example problem from Section 2.1.5. However, this time we allow transfers. For this example, we set the cost of transfers to $0$ and the duration of transfers, $\delta_{\mathtt{t}}(m, v_1, v_2)$, to $1$ for all items and vehicles.

One optimal solution for this problem is given in Table 2.5 and illustrated in Figure 2.3. Here, both vehicles pick up both of the items at their starting location, exchange the items so that each carries the items intended for delivery at its own destination, and then delivers the items. The cost of this schedule is $4$, which is half the cost of $8$ for the optimal solution of the same problem without transfers. By transferring items, we are able to gain a factor of two reduction in the total distance travelled.

### 2.2.5 The Benefit of Transfers

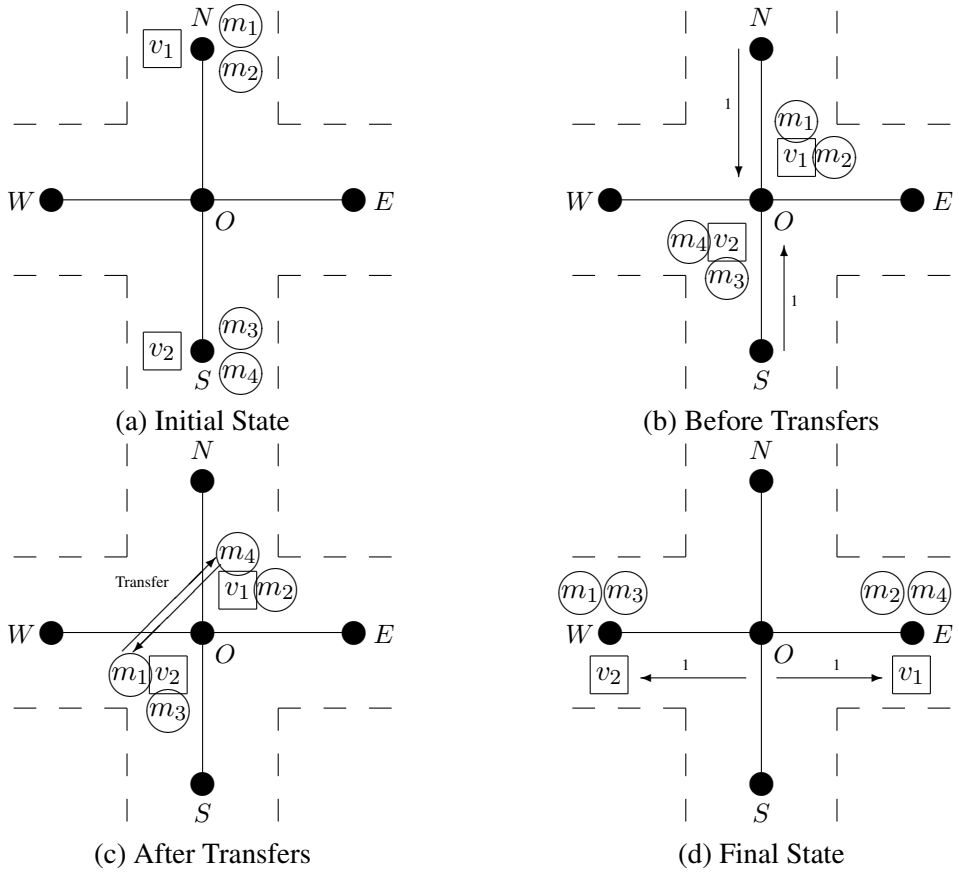
In the previous example, optimal solution cost for the PDP-T was half the cost of the optimal solution to the PDP. What is the maximum savings we can hope to achieve from transfers?

**Theorem 2.** *Let $R$ be the optimal schedule for any given PDP-T, and $S$ be the optimal solution for the same problem as a PDP, without transfers. Then $\mathbf{c}(R) \leq \mathbf{c}(S)$.*

*Proof.* $S$ is also a valid solution to the PDP-T, although it has no transfers. Therefore $\mathbf{c}(R) \leq \mathbf{c}(S)$. □

| $S$ | Action $a$ | $loc(a)$ | $b(a)$ | $t(a)$ | $\delta(a)$ |
|---|---|---|---|---|---|
| $S_1^{v_1}$ | START | N | 0 | 0 | 0 |
| $S_2^{v_1}$ | PICKUP $(m_1)$ | N | 0 | 0 | 1 |
| $S_3^{v_1}$ | PICKUP $(m_2)$ | N | 1 | 1 | 1 |
| $S_4^{v_1}$ | TRANSFER$(m_2, v_2, \text{O})$ | O | 3 | 3 | 1 |
| $S_5^{v_1}$ | RECEIVE$(m_3, v_2, \text{O})$ | O | 4 | 4 | 1 |
| $S_6^{v_1}$ | DELIVER$(m_1)$ | W | 6 | 6 | 1 |
| $S_7^{v_1}$ | DELIVER$(m_3)$ | W | 7 | 8 | 1 |
| $S_8^{v_1}$ | END | E | 9 | 9 | 0 |
| $S_1^{v_2}$ | START | S | 0 | 0 | 0 |
| $S_2^{v_2}$ | PICKUP $(m_3)$ | S | 0 | 0 | 1 |
| $S_3^{v_2}$ | PICKUP $(m_4)$ | S | 1 | 1 | 1 |
| $S_4^{v_1}$ | RECEIVE$(m_2, v_2, \text{O})$ | O | 3 | 3 | 1 |
| $S_5^{v_1}$ | TRANSFER$(m_3, v_2, \text{O})$ | O | 4 | 4 | 1 |
| $S_6^{v_2}$ | DELIVER$(m_4)$ | E | 6 | 6 | 1 |
| $S_7^{v_2}$ | DELIVER$(m_2)$ | E | 7 | 8 | 1 |
| $S_8^{v_2}$ | END | W | 9 | 9 | 0 |

Table 2.5: An Optimal PDP Schedule, $c(S) = 4$




(a) Initial State

(b) Before Transfers

(c) After Transfers

(d) Final State

Figure 2.3: The paths taken by the vehicles in the optimal schedule given in Table 2.5.

**Theorem 3.** *For a PDP-T without time windows, capacities, or any other optional constraints, with the objective of minimize distance travelled ($\alpha = 1$, $\beta = 0$), let $R$ be the optimal solution to the PDP-T. Then there exists a schedule $S$ without transfers, such that*

$$c(S) \leq 2c(R) + \sum_{v \in V} d_v(\textit{start}(v), \textit{end}(v)).$$

*Proof.* Let $R$ be the optimal solution to a PDP-T problem. With the following algorithm, we construct a new schedule $S$ and show that $c(S) \leq 2c(R)$.

1. Remove the START and END actions from $R$.
2. Construct $S$, such that $\forall v \in V\ S^v = []$, a schedule with no actions. Set the variables $v \leftarrow v_1$, $a \leftarrow R_1^v$, and $q \leftarrow []$ (an empty stack). We will iterate over the actions in $R$ to construct $S$, with $v$ as the vehicle we are currently constructing a schedule for, $a$ is the current action in the iteration, and $q$ as a stack used for backtracking in our search. Also, mark all actions in $R$ as unvisited ($\forall a \in R\ visited(a) \leftarrow False$).
3. Set $visited(a) \leftarrow True$. If $a$ is not a TRANSFER or RECEIVE action, append $a$ to $S^v$ and skip to Step 5.
4. Action $a$ is a TRANSFER or RECEIVE action. Let $v'$ be the action that executes $\textit{pair}(a)$, and let $a'$ be the first action in $R^{v'}$ preceding the action $\textit{pair}(a)$, such that $visited(a') = False$. If no such action $a'$ exists, do nothing, skipping the transfer action. Otherwise, if $a'$ does exist, push $a$ to $q$, mark $visited(\textit{pair}(a)) = True$, set $a \leftarrow a'$, and go to Step 3. We will insert the actions in the other vehicle's schedule in place of the transfer.
5. If $a$ is not the last action in $v$'s schedule, skip this step. Otherwise, we attempt to backtrack. If $q$ is not empty, pop the action $R_i^{v'}$ from $q$, set $a \leftarrow R_{i+1}^{v'}$ and go to Step 3. Otherwise, if $q$ is empty, select a new vehicle $v$ to construct a schedule for. Choose any vehicle $v' \in V$ with $visited(R_1^{v'}) = False$, set $v \leftarrow v'$, $a \leftarrow R_1^{v'}$, and go to Step 3. If no vehicle $v'$ with unvisited actions in $R^{v'}$ exists we are done. Set $\forall v \in V\ S^v \leftarrow [\text{START}] + S^v + [\text{END}]$ and terminate.
6. If $a = R_i^{v'}$, set $a \leftarrow R_{i+1}^{v'}$. If $visited(a)$ and $q$ contains an action from $v'$ schedule, pop an action $R_j^{v''}$ from $q$, set $a \leftarrow R_{j+1}^{v''}$, and go to Step 3.
7. If $visited(a)$, go to Step 6. Otherwise, go to Step 3.

This algorithm creates a valid schedule without transfers that delivers all items. From Section 2.1.3, on the conditions for a valid schedule, conditions (2.1), (2.4), (2.5), (2.6), (2.7), (2.8), and (2.9) are trivially satisfied since there are no constraints. Condition (2.2) is satisfied, since every non-transfer action in $R$ is marked as visited exactly once, when it is inserted into $S$ (Step

3), and therefore all items are picked up and delivered. It remains to be shown that condition (2.3) is satisfied, that is, that every item is picked up and later delivered by the same vehicle. We can see this is the case by considering the transfer graph $G(R)$ introduced in Section 2.2.2. For any pickup action $a$ and delivery action $b$, such that a path exists from $a$ to $b$ in $G(R)$, $a$ will be executed before $b$ in some robot's schedule in $S$. The actions $a$ and $b$ will not necessarily be executed by the same robot(s) as in $R$, because we include the actions in the schedules of all robots $v$ transfers to or from, directly or indirectly, into $S^v$, and guarantee that in $S$, any action that takes place before a transfer in $R$ precedes any action that took place after that same transfer in $R$. So $S$ is a valid schedule without transfers.

To show that $c(S) \leq 2c(R) + \sum_{v \in V} d_v(\mathit{start}(v), \mathit{end}(v))$, again consider the transfer graph $G(R)$ and how we traverse the graph with the action $a$. In Step 6, each edge in $G(R)$ is traversed exactly once. In Steps 4 and 5, we backtrack over edges in $G(R)$. In the first step, we backtrack over edges before a transfer, and in the second step, we backtrack over edges after a transfer. These backtracking steps do not overlap. Therefore we traverse every edge in $G(R)$ at most twice. This traversal does not include the edges to START and END edges, which we insert later, adding the term $\sum_{v \in V} d_v(\mathit{start}(v), \mathit{end}(v))$. The condition for popping from $q$ in Step 6 ensures that the original final ending action for a vehicle still precedes the END action, so the additional term really only applies to vehicles which do no pickups or deliveries in $S$. □

So if vehicles start and end at the same location or have unspecified ending locations, transfers will at most give a factor of two improvement for the total distance travelled metric. And, as we showed in the previous example, PDP-Ts do indeed exist which give this factor of two cost improvement.

However, take note of the assumptions behind this theorem: the proof breaks down when constraints are added to the problem. Given additional constraints, such as time windows and capacities, there are problems for which we can do better than a factor of two improvement for the objective function, and even deliver more items if transfers are allowed.

As an example, consider the problem in Figure 2.4 with $n$ items and $n + 1$ vehicles. One vehicle starts with all of the items at a location a distance $D$ from a central hub, where $n$ other vehicles begin. All vehicles have infinite capacity. All items must be delivered to distinct locations a distance $1$ from the central hub within the time window $[0, D + 1]$. All actions have duration $0$, and $\alpha = 1, \beta = 0$.

For the PDP, only a single item can be delivered within the time windows by $v_0$. For the PDP-T, $v_0$ picks up all the items, transfers them to $v_1$ through $v_n$ at the central hub, and these vehicles deliver the items simultaneously. All $n$ items are delivered, giving an enormous improvement in the objective function, from $D + 1 + \gamma(n - 1)$ for the PDP to $D + n$ for the PDP-T.

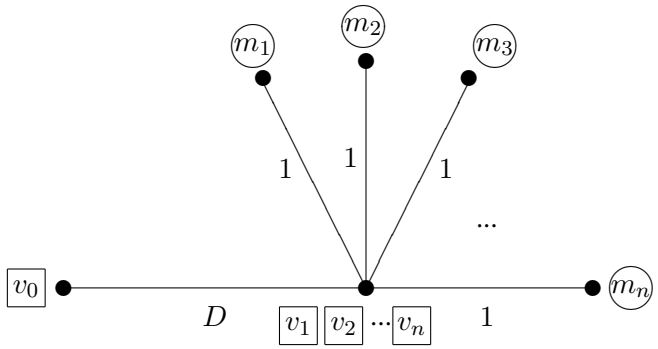



Figure 2.4: A hub and spoke PDP. All $n$ items start at the same location as the vehicle $v_0$, and end at the locations shown. The distance $D >> 1$.

Furthermore, consider the same problem without time windows, but with the minimum makespan objective (to finish delivering all the items as quickly as possible). For the PDP, one vehicle can deliver all the items, giving a makespan of $D + n$, or $n$ vehicles can each pick up and deliver one item, giving a makespan of $2D + 1$. For the PDP-T, with the same solution as before, the makespan is $D + 1$.

By allowing transfers, we can reduce the distance travelled by at most a factor of two in unconstrained problems. In more complex problem domains, transferring may enable us to deliver items we wouldn't be able to without transfers.

## 2.3 Variations on the PDP-T

The PDP without transfers has been studied extensively by many previous researchers (which we detail extensively in Section 8). However, the naming scheme in the literature is inconsistent, and it is often the case that when two researchers refer to a "PDP" they mean quite different things. Here, we seek to establish a consistent notation that will be used for the remainder of the thesis document.

In the scheduling literature, there are many variations on the specification for what is called a "PDP" problem. For example, sometimes when time windows are included, the problem is called a PDP with Time Windows (PDPTW) as opposed to a PDP which is assumed to not have time windows. Or a PDP is assumed to not have capacities, as opposed to a Capacitated PDP (CPDP) which does. Then there is the CPDPTW which includes capacities or time windows. In other work, each of these problems are referred to as simply the "PDP."

In this thesis, when we refer to the "PDP" we mean the most general version of the problem, which includes all of the constraints mentioned earlier in this section. We also consider two specialized, less general variants of the problem with fewer or slightly different constraints, which

we name:

- **Pickup and Single Delivery Problem (PSDP)** All the items are collected from different locations and then delivered to the **same** location. The vehicles do not have destinations specified, have unlimited capacities, and there are no time windows.
- **No Capacities Pickup and Delivery Problem (nCPDP)** A PDP with time windows but unlimited capacities and without vehicle destinations.
- **No Times Pickup and Delivery Problem (nTPDP)** A PDP in which there are capacities, but no time windows. The vehicles may deliver the items at any time.

Table 2.6 lists some properties and constraints of the problem, and shows how they differ in the PDP variants. In this thesis, we will begin with how to solve the simpler problems and progress to the more complex problems, building on previous work. Note that algorithms to solve the more general problems can also be used to solve the more specific problems. However, we can take advantage of the structure in simpler problems to create faster and better algorithms, particularly in the case of the PSDP, for which we are even able to provide theoretical guarantees of solution quality.

| Name | PSDP | nCPDP | nTPDP | PDP |
|---|---|---|---|---|
| Vehicle Starting Locations | X | X | X | X |
| Vehicle Ending Locations | | | X | X |
| Item Pickup Locations | X | X | X | X |
| Item Delivery Locations | Single | X | X | X |
| Capacities and Demands | | | X | X |
| Time Windows | | X | | X |
| Max. Route Durations | | | | X |
| Max. Transport Times | | | | X |

Table 2.6: Variations on the PDP

All of the variants of the PDP we examine are NP-hard, as they can be reduced to the TSP. Additional constraints reduce the size of the search space, and one would expect they would make the PDP easier to solve. However, in practice, for time window, maximum route duration, and maximum transportation time constraints, the additional computation time required to check the constraint's violation often outweighs the savings from the reduction in the size of the search space, making these more constrained problems more computationally expensive.

## 2.4 Online and Distributed PDP-T

Until this point, the PDP-T and PDP have been presented as static problems: all of the vehicles, items and constraints are known up front, and solved by a centralized algorithm. However, in real-world applications of the PDP-T, we often do not know all of this information beforehand, and must address the **Online PDP-T**.

In the Online PDP-T, the entire set of items is not known beforehand. New requests for item deliveries come in on the fly, and the new items must be added into the schedule, if possible. Each item $m \in M$ is associated with a request time $rt(m)$, at which point the request to pickup and deliver item $m$ is known. At a given time $t$, the known set of items is $M = \{m \in M : rt(m) \leq t\}$. Only these items are incorporated into the schedule at time $t$, not items that are announced later. Algorithms for the online PDP-T will need to **replan** as new information about the problem comes in, after part of a previous schedule has already been executed. When replanning, a valid schedule must consider whether any items have already been picked up, and the current locations of the vehicles. For the online PDP-T, it is impossible to always find the optimal solution in the same sense as for the offline PDP-T, since part of the schedule must be executed before the entire problem is known.

Additionally, in the online PDP-T, vehicle distance estimates may be incorrect, and vehicles may be delayed or fail entirely. Requests may also be cancelled or modified. The scheduler must replan to handle all of these changes inherent in a dynamic world.

A second variant on the PDP-T is the **Distributed PDP-T**, in which there is no central controller to form a schedule for all of the vehicles. In the distributed PDP-T, vehicles negotiate among themselves to form a schedule. Requests may also come in an online fashion for the distributed PDP-T.

We fully address the Online PDP-T, and discuss and consider the Distributed PDP-T in addition to the Static PDP-T in this thesis.

## 2.5 Chapter Summary

In this chapter, we formally introduced the PDP, along with the PDP-T and the idea of transfers. We established the notation for the remainder of this thesis, as well as what makes a valid schedule. Furthermore, we analyzed and discussed the potential benefits arising from allowing transfers.

# Chapter 3

# Task Scheduling and Execution on the CoBot Robots

The motivation for this thesis stems from the CoBot robots, which we have developed and deployed to fulfill user-requested tasks in a multi-story building. The robots autonomously navigate anywhere in the building, and pickup and deliver items. New tasks, which are requested by users in an online manner, must be incorporated into the schedule immediately so that the requester can be informed about whether their task will be executed. The CoBots have been deployed for nearly four years, and are very robust and reliable.

Originally, the robots were scheduled by solving a Mixed Integer Program (MIP) optimally [32]. However, this approach scales poorly with the number of robots and the number of tasks. Furthermore, we realized that the schedules could be improved if the robots were to *transfer* items, which became the main focus of this thesis. For example, if two robots were going to deliver an item to the same floor, one robot could transfer its item to the other, and only a single robot would need to take the elevator. Transfers could also reduce the duplicated distance taken by multiple robots down the same hallways, while allowing the robots to complete their tasks faster.

In this chapter, we present the CoBot robots and the tasks they can perform. We discuss how these tasks are scheduled optimally without transfers using an MIP, show how the robots execute tasks and how the task execution can be monitored and interrupted. Finally, we present results from deploying CoBot in the building to actual users for a two week study. Here we discuss scheduling without transfers, and in later chapters we discuss specific algorithms for scheduling pickup and delivery problems with transfers, including on the CoBots.

## 3.1 The CoBot Robots

Figure 3.1: CoBot-1, CoBot-2, and CoBot-4: Collaborative service robots deployed in our buildings. (An additional CoBot-3 is deployed off campus.)

We have deployed three robots in the Gates Hillman Center (see Figure 3.1). The robots navigate smoothly and quickly due to their omnidirectional bases.[1] All computation is done on an onboard tablet or laptop computer. For sensing, the robots use either a combination of planar LIDAR and an RGB-D camera, or multiple RGB-D cameras. The CoBots autonomously localize and navigate using depth-camera and LIDAR-based localization and navigation algorithms [16, 18, 17, 19]. The CoBots autonomously avoid obstacles by moving to the side of the hallway, but if they cannot avoid an obstacle, they stop and say "Please excuse me." until the obstacle is moved.

Part of the secret to getting the robots to run robustly and reliably every day is that we recognize that the robots currently cannot do everything we would like them to. For the things they cannot do, the robots ask humans in the building for help. For example, the CoBots do not have arms, so they ask humans to press the elevator buttons for them. We call this idea *symbiotic autonomy* [95].

The maps the CoBots use are extracted from floor plans of the building. A separate navigation map is a graph in which the edges are the corridors the CoBots can travel through. Dijkstra's

[1] Many thanks to Mike Licitra for designing and building the CoBot robots.

algorithm is applied to the navigation map to find the shortest path between any two locations in the building. These distances are divided by robot velocity estimates and added to a constant cost for elevator usage in order to estimate the time taken between any two locations for the scheduler's distance function $d(a, b)$. Positions and orientations are manually annotated on the maps for each room in the building, giving the location CoBot heads to to fulfill tasks requested at those locations.

The CoBots make use of the underlying abilities to move to any point in the building and to ask for human assistance to complete all of their tasks.

## 3.2 Tasks for the CoBots

The CoBots are deployed to complete user tasks. The tasks that users may request include:

1. **Visit a Room:** CoBot goes to a room to introduce itself.
2. **Deliver a Spoken Message:** CoBot travels to a room and speaks a message entered by the task solicitor to a room's occupant.
3. **Escort a Visitor:** CoBot meets a visitor at the elevator and leads them to a room.
4. **Pickup and Delivery:** CoBot goes to one room, asks a user to place a specified item in its basket, and delivers that item to a different room (see Figure 3.2).

(a) Pickup (b) Delivery (c) Find Object

Figure 3.2: In a pickup and delivery task, *(a)* one user places an item in CoBot's basket and *(b)* another user removes it when the item is delivered. In *(c)*, CoBot finds and retrieves a cup of coffee from the kitchen.

5. **Telepresence:** CoBot goes to a room and makes its telepresence interface available for trusted users to attend a meeting. Users can control CoBot over the web and communicate via videoconferencing [29].
6. **Multi-Object Delivery:** CoBot retrieves a set of items from a single location and delivers them to multiple locations. Examples include delivering candy and mail.
7. **Multi-Object Retrieval:** CoBot retrieves a set of items from multiple locations and delivers them all to a single location. Examples include picking up mail from assistants' offices and delivering it to a central office, delivering coffee and returning to a central location with payments, or delivering business cards and returning what is left over.
8. **Find and Retrieve an Object:** The user tells CoBot to find an item by name, such as "towel," "pen," or "coffee," and the robot queries OpenEval [99] to decide whether to go selected types of locations, e.g., an office, kitchen, common area, or restroom (see Figure 3.2c).

Although only task #4 is called a "Pickup and Delivery" task, *all* of the tasks can be modeled as tasks with a pickup and delivery and the problem can be formulated as an online nC-PDP, as described in Chapter 2. To do so, for tasks #1, #2, #5, and #8, we define an item $m$ with $\mathit{start}(m) = \mathit{end}(m)$, and with $\delta_{\mathrm{p}}(m)$ corresponding to the expected duration of the task. Task #3 is already a pickup and delivery task, except with robot dialogue that fits the transport of a person. Tasks #6 and #7 correspond to multiple pickup and delivery tasks which share a start or ending location.

## 3.3 Requesting Tasks

Figure 3.3 shows the high level architecture for task scheduling and execution on the CoBot robots. Users place task requests through one of three interfaces: a graphical UI on a CoBot, spoken dialogue on CoBot, or on a website. The requests are all sent to a centralized scheduler, which then immediately constructs and sends a schedule for each of the robots. The server must form a schedule within seconds so that the user can be informed of whether the new task was accepted. The robots inform the centralized server of their positions as they move, and send a message to the server when they complete or abort a scheduled task.

Each task request contains a number of fields, listed in Table 3.1. These include the location of the task, a time window in which the task should be executed, and any other task specific information. An example Pickup and Delivery task is shown in Table 3.2.

Each task request maps to one or multiple item requests $m$ in the PDP. The fields determine

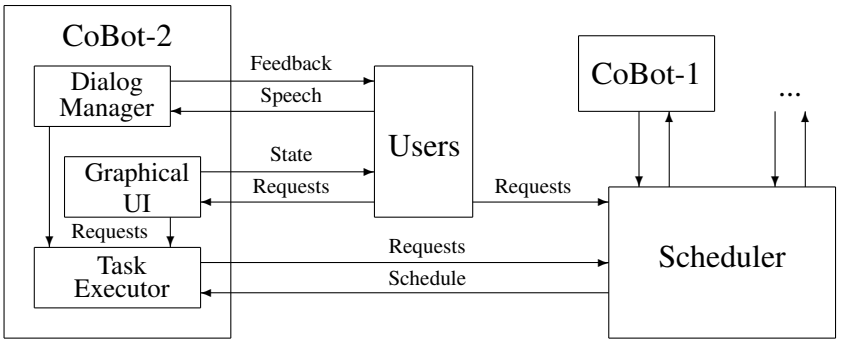



Figure 3.3: The system architecture of the CoBots. Users place requests through the dialog manager or a graphical UI on the robots, or directly to the web scheduler, which forms schedules for all the robots.

| **Parameter** | **Description** |
|---|---|
| ID | The unique id number of the task. |
| Type | Type of the task, e.g., Visit a Room, Pickup and Delivery, etc. |
| Time Window | The time interval in which the task should be completed. |
| Location(s) | A single room location or multiple room locations, depending on the task type. |
| Robot(s) | A list of robots the task may be executed on (default is any). |
| Object Name | Name of the object to be retrieved / delivered (if required). |
| Message | A spoken message to deliver (for message delivery tasks only). |
| Owner | Name and email address of the user who booked the task. |

Table 3.1: The fields in a task request.

| **Parameter** | **Value** |
|---|---|
| ID | 1107 |
| Type | Pickup and Delivery |
| Time Window | May 2, 3:00 PM - 4:00 PM |
| Location(s) | GHC 7008 to GHC 7412 |
| Robot(s) | Any |
| Object Name | Robot Parts |
| Owner | Christina |

Table 3.2: An example specification of a Pickup and Delivery Request.

the task time windows $W_p(m)$ and $W_d(m)$, as well as the locations *start*$(m)$ and *end*$(m)$.

Task solicitors may place a request with a variety of methods, including by visiting CoBot's website, or by interacting directly with the robots via speech or a touch screen interface. Task requests can also be placed autonomously by the robots themselves [67].

When users visit CoBot's website (see Figure 3.4), users select the task type and locations from drop down lists. They specify either "as soon as possible," choose a specific time, or enter a time window. The first two options are converted in the scheduler to a fifteen minute time window. Depending on the task type, users may enter additional parameters, such as pickup or dropoff locations, item or person names, or a message to deliver.

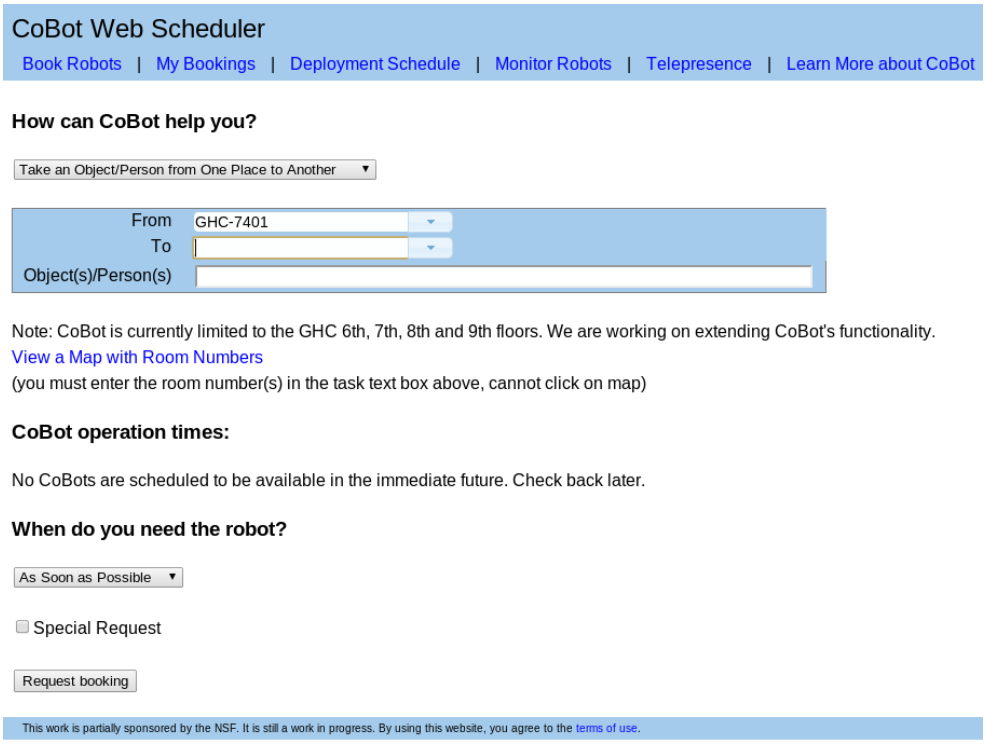



Figure 3.4: Users schedule tasks for CoBot on CoBot's web site. The user inputs correspond to the task request parameters in Table 3.1.

The touch screen interface, shown in Figure 3.5, is similar to the website, with drop down boxes to specify the tasks. We also have a speech interface, in which the user presses a button to speak to CoBot [66]. The robot listens to the user, responds, and the user presses the "speak" button once again to continue dialogue.[2] All three interfaces allow a user to fill out the parameters in Table 3.1 required for a task request. For convenience, in addition to specifying a window of

[2]Special thanks to Michael Murphy for designing CoBot's touch screen interface, and to Tom Kollar, Vittorio Perera, Robin Soetens, and Yichao Sun for developing CoBot's speech interface.

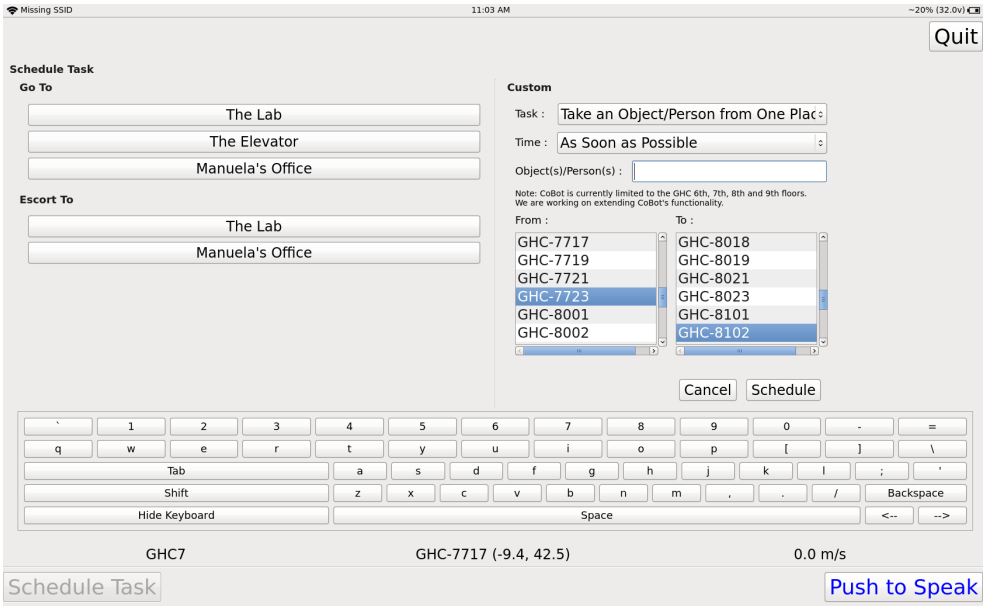



Figure 3.5: Users schedule tasks for CoBot on CoBot's touchscreen. The user inputs correspond to the task request parameters in Table 3.1.

time, users may request that a task be executed "as soon as possible," which attempts to schedule the task within the next fifteen minutes.

Task requests are submitted to a centralized scheduler running on the server. The scheduler may either accept the request or reject it. If the task is accepted, the user is not informed of the scheduled execution time, only that the request has been accepted, because the task may be delayed or rescheduled based on new incoming requests. The scheduler should quickly add tasks to the schedule, within a period of several seconds, so that the user may be immediately informed whether the task was accepted. Once a task is accepted, the user is able to cancel it from the website.

One difficulty in using a centralized scheduler arises from the fact that the robots move around in the building and may not always have internet access. This is especially true near the elevators and when the robots switch wireless access points. As a fallback method, if a robot cannot contact the scheduler, the robot inserts new requests into its own, separate, local task list. Tasks on the local task list are given precedence, and executed before tasks on the robot's schedule. The robot stores multiple actions on the server's schedule for it, so when an action is completed, the robot can proceed to the next action even without communication from the server. When network connectivity is regained, the server sends any updated schedules to the robot, and the robot alerts the server of any completed tasks. This way the robots remain fully functional even if the server or the network connection fails, although it will likely execute a lower quality schedule. There

may be situations where tasks are rescheduled to other robots when a new task comes in, and a robot that cannot communicate still executes a task that has been reassigned to another robot. However, this is rare, as communication failures are typically temporary.

CoBot is unfortunately not always available to execute tasks, because it needs to be switched on and unplugged from its charging by a human. To make the CoBots available as frequently as possible, administrators schedule time windows of several hours when a robot is available to the public. Upcoming publicly available timeslots for the next week are displayed on the scheduling requests page. Users can always schedule tasks on the robots, regardless of available timeslots. For testing purposes, administrators may also make privately available timeslots which are available only to trusted users. The public and private time windows correspond to the availability windows $W_a(v)$ in the PDP formulation (see Chapter 2). Requests $m \in M$ placed by the public have their allowed vehicles $AV(m)$ set to include only public timeslots.

Users often have tasks they wish the robots to do outside scheduled hours, particularly escorting visitors through the building and going to classrooms for demos. In these cases, users can check a "special request" box on the task request submission form and include a descriptive message. An email is sent to the administrator responsible for the CoBot robots on that day (a rolling schedule including the members of our group is available on the website to the administrators). The administrator clicks a link in the email to either accept the request, or deny the request with an optional explanation or suggestion for an alternative time.

## 3.4 Scheduling Tasks

The task requests are sent to a centralized server, which first converts all the user requests into a set of component actions $A$ from Chapter 2. In this step, a single request may be converted into multiple actions: for example, a **Pickup and Delivery** task becomes a PICKUP action and a DELIVERY action; a **Visit a Room** task becomes a single DELIVERY action; and a **Multi-object Delivery** task becomes a single PICKUP action and multiple DELIVERY actions. Although the problem can be formulated as an nC-PDP with a one-to-one matching of PICKUP to DELIVERY actions, as explained in Section 3.2, to simplify the solution method, here we allow one-to-multiple mappings (i.e., one DELIVERY action may correspond to multiple PICKUP actions, for the **Multi-Object Retrieval**, or to no PICKUP actions, for the **Visit a Room** task. However all pickups must still precede all associated deliveries, and all these associated actions must be executed by the same vehicle.

The central server attempts to fit all the requested actions into a schedule. The scheduler chooses the ordering of the actions and their assignment to robots, while a separate path planner

onboard the robot chooses which path the robot will take to go from place to place. In this chapter we present an initial approach based on Mixed Integer Programming (MIP) to schedule tasks. In later chapters we replace the MIP approach with more sophisticated algorithms which scale to larger problem instances and schedule transfers between robots. The infrastructure to request and execute tasks on the CoBots is independent of the specific scheduling algorithm used.

### 3.4.1 MIP Scheduling

We find a schedule by formulating and solving a MIP for the variables $t_a$ and $r_a^v$, where for each action $a$, $t_a = t(a)$ and $r_a^v$ is an indicator variable indicating whether or not vehicle $v \in V$ executes the action $a \in A$. From the times and vehicles of the actions, we are able to construct the schedule as formulated in the previous chapter. We add a DELIVER action in $A$ for each robot of duration 0, the location set to the robot's current location, and with the time window only containing the current time, so that the schedule models the robots' starting positions (recall that for this MIP formulation, DELIVER actions do not require corresponding PICKUP actions). In this problem, the vehicles in $V$ represent availability windows rather than robots, so each physical robot may have multiple entries in $V$ for different availability windows.

#### Single Vehicle MIP

To find the optimal schedule, we solve an MIP. If there is only a single vehicle / interval $v$, we define the following MIP for the variables $t_a$ and $p_{a,a'}$, where $p_{a,a'}$ is an indicator variable to indicate that action $a$ precedes action $a'$. Recall from Chapter 2 that $[e(a), l(a)]$ is the time window for task $a$, $\delta(a)$ is the task's duration, and $loc(a)$ is the task's location. The window $[e_v, l_v] = W_a(v)$ is the availability window for vehicle $v$.

$$\min \sum_{a \in A} t_a \tag{3.1}$$

$$\forall a \in A \;\; e(a) \leq t_a \leq l(a) \tag{3.2}$$

$$\forall a \in A \;\; e_v(a) \leq t_a \leq l_v(a) \tag{3.3}$$

$$\forall a, a' \in A \text{ s.t. } a = \text{PICKUP}(m), a' = \text{DELIVER}(m) \; t_a \leq t_{a'} \tag{3.4}$$

$$\forall a, a' \in A \;\; 0 \leq p_{a,a'} \leq 1 \text{ int} \tag{3.5}$$

$$\forall a, a' \in A \;\; t_a + \delta(a) + d(loc(a), loc(a')) - t_{a'} \leq |l(a') - e(a)|(1 - p_{a,a'}) \tag{3.6}$$

$$\forall a, a' \in A, v \in V \;\; t'_a + \delta(a') + d(loc(a'), loc(a)) - t_a \leq |l(a) - e(a')|p_{a,a'} \tag{3.7}$$

The objective is to execute all actions as early as possible (Equation 3.1). Equation 3.2

enforces all the task time constraints, while Equation 3.3 ensures that the action is scheduled during the single availability window. In Equation 3.4, the precedence constraints are enforced, that all pickup actions must occur before any corresponding delivery actions with the same item. In Equation 3.5 the variables $p_{a,a'}$ are defined as indicator variables, whose values must be either zero or one, indicating whether or not action $a$ is executed before action $a'$. Then Equations 3.6 and 3.7 ensure that the travel times between every pair of actions are feasible, that is, that $t_a + \delta(a) + d(\mathit{loc}(a), \mathit{loc}(a')) \leq t_{a'}$, and after every action there is sufficient time to execute the action and travel to the location of the next action, assuming that the estimated robot travel time is correct. On the right hand side of Equation 3.6, the value is zero if $p_{a,a'} = 1$, i.e., action $a$ precedes action $a'$, and the equation applies. If $p_{a,a'} = 0$, then the right side of the equation is greater than the left hand side can possibly be, and the equation is always satisfied. Equation 3.7 has the same idea but applies when action $a'$ precedes action $a$.

With this MIP formulation, we are able to solve the single vehicle scheduling problem optimally with commercial MIP solvers. It should be noted that unlike our formulation in the previous chapter, here the scheduling fails unless *all* tasks can be scheduled. This behavior is desirable for the CoBots, since we are committing to human users that we will complete their requested tasks. We do not want to commit to a task we cannot complete. If the scheduling fails, then we tell the user who requested the latest task that we cannot fulfill their request, and revert to the previous schedule.

### Multi-Vehicle MIP

Next, we extend the MIP to multiple vehicles by introducing indicator variables $r_a^v$ which indicate whether action $a$ is executed by vehicle $v$. We provide the full MIP for multiple vehicles:

$$\min \sum_{a \in A} t_a \tag{3.8}$$

$$\forall a \in A \;\; \boldsymbol{e}(a) \leq t_a \leq \boldsymbol{l}(a) \tag{3.9}$$

$$\forall a, a' \in A \text{ s.t. } a = \textsc{Pickup}(m), a' = \textsc{Deliver}(m) \;\; t_a \leq t_{a'} \tag{3.10}$$

$$\forall a, a' \in A \;\; 0 \leq p_{a,a'} \leq 1 \text{ int} \tag{3.11}$$

$$\forall a \in A, v \in V \;\; 0 \leq r_a^v \leq 1 \text{ int} \tag{3.12}$$

$$\forall a \in A \;\; \sum_{v \in V} r_a^v = 1 \tag{3.13}$$

$$\forall a \in A, v \notin \boldsymbol{AV}(a) \;\; r_a^v = 0 \tag{3.14}$$

$$\forall v \in V, a, a' \in A \text{ s.t. } a = \text{PICKUP}\,(m)\,, a' = \text{DELIVER}(m)\;\; r_a^v = r_{a'}^v \tag{3.15}$$

$$\forall a \in A, v \in V \;\; e_v(a) r_a^v \leq t_a \leq l_v(a) + l\,(a)\,(1 - r_a^v) \tag{3.16}$$

$$\begin{gathered}\forall a, a' \in A, a \neq a', v \in V \;\; t_a + \delta\,(a) + d(\mathit{loc}(a), \mathit{loc}(a')) - t_{a'} \leq \\ |l\,(a') - e\,(a)\,|(1 - p_{a,a'} + 2 - r_a^v - r_{a'}^v)\end{gathered} \tag{3.17}$$

$$\begin{gathered}\forall a, a' \in A, a \neq a', v \in V \;\; t_{a'} + \delta\,(a') + d(\mathit{loc}(a'), \mathit{loc}(a)) - t_a \leq \\ |l\,(a) - e\,(a')\,|(p_{a,a'} + 2 - r_a^v - r_{a'}^v)\end{gathered} \tag{3.18}$$

Equations 3.8, 3.9, 3.10, and 3.11 are identical to the single vehicle MIP. Equation 3.12 defines the variables $r_a^v$ as indicator variables, Equation 3.13 ensures that each action is assigned to exactly one vehicle, and Equation 3.14 ensures that actions are only assigned to the allowed vehicles, which the user may specify. Equation 3.15 requires that all actions involving the same item be executed on the same vehicle. Equation 3.16 replaces Equation 3.3 from the single vehicle MIP, and ensures that actions fall within the schedule vehicle time window. If action $a$ is not scheduled for vehicle $v$, then $r_a^v = 0$ and the equation reduces to $0 \leq t_a \leq l\,(a)$, which is always satisfied. Finally, Equations 3.17 and 3.18 replace Equations 3.6 and 3.7, ensuring that the travel times between actions are feasible, but now the change to the right side of the equation means that these equations only apply to actions executed on the same vehicle.

### Solving the MIP

We construct and solve a new MIP every time a new request comes in or a request is cancelled or modified. When rescheduling, if a PICKUP action has already been executed by a robot, that action is removed from the set of actions $A$ and not rescheduled, but additional constraints are added such that any corresponding DELIVERY actions are be executed on the same robot. Since the MIP is solved online when a new request is placed, it needs to be solved quickly, so that we can immediately inform the user that the task was accepted, or reject the task and ask the user to relax their constraints.

Although solving the MIP is an NP-hard problem, we have found that in practice it can be solved quickly for certain problems. We generated a thousand random problem instances, each of 15 tasks with two minutes to half an hour durations, with time windows over the course of four hours, and a single availability window. This is the type of input we expect the robot to receive in typical usage. The scheduler solved (either found a schedule or found that no schedule existed) 99% of the problems in under two seconds using the `lp_solve` MIP solver [14]. We also experimented with the CPLEX solver [57] and found that the performance was slightly better, but on the same order of magnitude.

As the MIP finds the optimal solution, it does not scale well to larger problem instances.

We have developed additional scheduling algorithms, which do not find the optimal solution but which find a solution more quickly and scale to much larger problem instances, which we present in the later chapters of this thesis.

### 3.4.2 Illustrative Scheduling Example

As an example, consider that four tasks are requested, which are divided into six component actions:

- A PICKUP action $a_1$ and a DELIVER action $a_2$ to pickup a paper with revisions from room 7205 and deliver it to room 7127. For both tasks, $W(a_1) = W(a_2) =$ [3:00 PM, 3:30 PM].
- An action $a_3$ to deliver a message to the occupant of office 7110, with $W(a_3) =$ [3:10 PM, 3:20 PM].
- An action $a_4$ to pick up a visitor from the elevator and an action $a_5$ to deliver the visitor to room room 7213, with $W(a_4) = W(a_5) =$ [3:00 PM, 3:10 PM].
- An action $a_6$ to deliver a message to the occupant of office 7801, with $W(a_6) =$ [3:10 PM, 3:15 PM].

The set of available vehicles is $V = \{$CoBot-1, CoBot-2$\}$, and all actions are allowed on any vehicle.

The task locations and the robots' initial positions are shown in Figure 3.6.

One potential schedule would be:

| CoBot-1 | | CoBot-2 | |
|---|---|---|---|
| Time ($t_a$) | Task | Time ($t_a$) | Task |
| 3:00 PM | $a_4$: Meet visitor at elevator. | 3:00 PM | $a_1$ : Pick up paper. |
| 3:07 PM | $a_5$: Deliver visitor. | 3:10 PM | $a_3$ : Deliver message |
| 3:12 PM | $a_6$: Deliver message. | 3:15 PM | $a_2$ : Deliver paper. |

Table 3.3: An example schedule.

The tasks are divided between the robots to complete more tasks more quickly. When this problem is input into the MIP scheduler, the output is the times $t_a$ as shown in the table, as well as the robot assignments $r_a^v$.

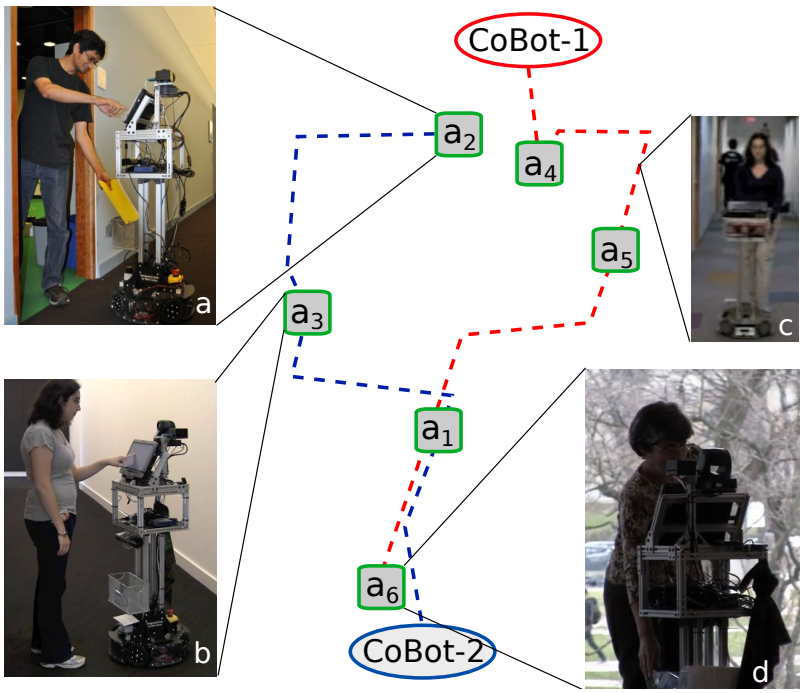



Figure 3.6: Robot starting positions, task locations, and the generated plan. While executing the plan, CoBot-2 *(d)* retrieves an envelope, *(b)* delivers a spoken message, and *(a)* delivers the envelope. CoBot-1 *c* escorts a visitor to an office before delivering a message.

## 3.5 Executing and Managing Schedules

Once the full schedule $S$ is formed, the server sends each robot $v$ a copy of its own schedule $S^v$ over the wireless network. Each robot then autonomously executes its own schedule. However, the robots may be interrupted by the users, who input new requests and modify or cancel existing requests. A new schedule is sent to each robot whenever the schedule changes.

### 3.5.1 Executing a Schedule

Given a non-empty schedule $S^v$, the robot immediately heads to the location $loc(S_0^v)$, and waits at this location until the time $e(S_0^v)$ when it begins executing the action $S_0^v$. We begin execution at $e(S_0^v)$ rather than the scheduled starting time $t(S_0^v)$ in case the previous action was finished early. The robot may still need to wait for a time at the location of the next action if it arrives too early. If the robot has no actions in its schedule, it remains in place until a new task is requested.

The *task executor* component handles execution of the task. This component instructs the robot where to go and what to say, using a finite state machine for each task type. The states of this finite state machine often consist of executing another finite state machine, such as sub-tasks

to travel to a location or to ask a human a question.

A waypoint path planner, given the robot's current location and destination, plans a path for the robot to arrive at the destination. This path is executed by a lower level navigation component, which takes the shortest path between points on the same floor. The way point path planner plans not only to arrive at the destination, but also to take the elevator and seek human help if necessary. Since CoBot does not have arms, it relies on human help to press the elevator buttons, and may even actively seek humans to help it.

The distance function $d(a, b)$ estimates the time to travel between two points based on the distance and the measured average velocity of the robot. More advanced distance functions could learn the time taken to travel different corridors, or even consider the time of day and traffic conditions (e.g., when students are released from class the hallways become very crowded). There may be a large variance in the time taken to complete a task since execution relies on finding human help. For this reason, our distance function $d(a, b)$ overestimates the time to receive an item, deliver an item, and take the elevator, so that the robots are unlikely to complete tasks late, but may finish ahead of schedule. In practice, humans typically show up at the elevator every several minutes, and are likely to help the robot. If a human does not offer to help within a few minutes, an email is sent to a list of the robots maintainers within the building, and one of them goes to help the robot. However, the distance function is still often incorrect, and to account for this we replan for delays in Chapter 6. For retrieval and delivery tasks, if a human does not appear after several minutes, the task is aborted and marked as failed.

Once a robot completes its task, it reports the completion status to the centralized server. The task may either succeed or fail. For example, a task to deliver a message will fail if no one presses the button to hear the message after a certain time. Once the task is complete, the robot removes the task $S_0^v$ from its schedule and begins to execute the next task in the schedule.

### 3.5.2 Managing Task Lists with Interruptible Autonomy

When deploying CoBot, we ran into the unexpected problem that the robots were *too* autonomous. Once they began a task we had no way to stop them. Tasks can be cancelled and new requests placed from CoBot's website, but a method was needed to modify the robots' schedules on the fly, in person. For example, if CoBot was delivering a package, there was no way for the recipient to stop CoBot in the middle of the hall and cancel the rest of the delivery trip, or to give CoBot another task.

To make the robots more responsive to users in the building, passerby can request a task from a CoBot robot as it moves, either through the touch-screen interface or through spoken interaction. Users stand in front of the robot to block its motion, then press a button to stop the

robot and either open the touch screen interface or initiate dialogue. In addition to assigning new tasks, users can also inquire about what the robot is doing and cancel tasks. We call this concept *interruptible autonomy*, where users can interrupt the robot's autonomous tasks and influence the robot's plan [107].

The interruption procedure through dialogue consists of three parts. First is the dialog parser, a dialog management module that interacts with the human user through speech. The dialog parser listens to a user's speech command and converts the speech into text candidates. Using a knowledge base to form a probabilistic model [66], we ground the user's speech command into actions the CoBots are capable of. The dialog parser first extracts linguistic constituents of the speech text, then parses the action and arguments for a task request. If the dialog parser does not understand the user's speech command, it further interacts with the user by prompting possible candidate options the user can choose from. It also updates its knowledge about the mapping relation between the speech text and the actual actions, locations and objects through the results of dialog so that they can be used for future interactions.

There are several possible actions for users to choose from when giving speech commands:

- **Inquire**: A user asks a CoBot about its current status. The CoBot responds by describing its current task type and destination.
- **Request Task**: A user requests a new task for a CoBot to execute. The request is passed on to the scheduler, and it may be scheduled either for immediate execution, scheduled for future execution, or rejected. If the task is rejected, the user is informed through dialogue and can adjust the time constraints so that the task can be executed. Figure 3.7 provides an example of an interruption to request a new task, which is scheduled for immediate execution.
- **Cancel Task**: A user cancels CoBot's current task. Only the owner of the task or the administrators are authorized to do this. Once a task is cancelled, the owner of the task receives an email notice of the cancellation.

So in addition to requesting tasks online, users can place requests, inquire about CoBot's status, and cancel existing requests in person, while the robot is moving.

## 3.6 Monitoring Schedule Execution

Although CoBot robustly and reliably executes its schedules autonomously, for ease of deployment, we found it essential to be able to monitor the execution of schedules remotely when necessary. We have developed a fully featured telepresence system for the CoBots, and users

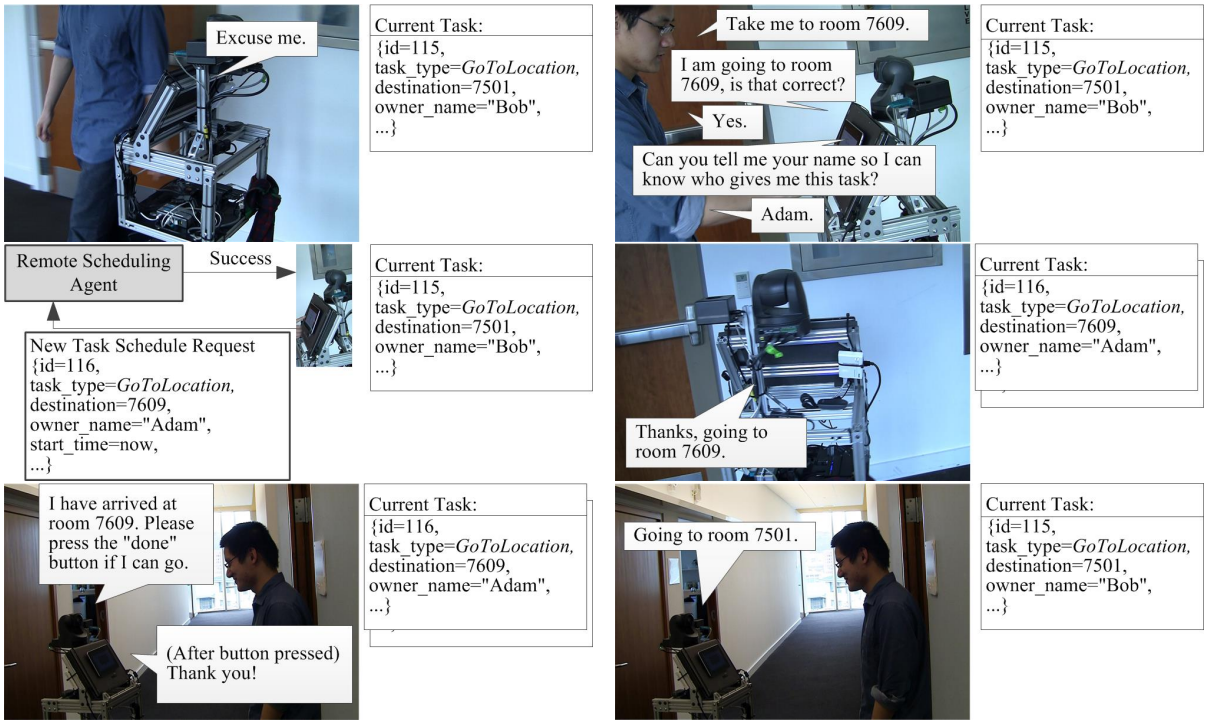



Figure 3.7: A CoBot robot was interrupted by a passerby while executing a visit a room task. A new task is assigned to the CoBot and successfully scheduled for immediate execution. CoBot begins executing the new task and resumes the previous task when the new one is finished.

may schedule tasks to attend meetings or poster sessions via CoBot. This telepresence system doubles as a way of monitoring the robot's execution for administrative users. For example, CoBot will sometimes be waiting for important visitors by the elevator to escort them to the lab, and we can easily check if the visitor has arrived yet. Or if CoBot is blocked for several minutes and sends the group an email, we can log in to check if someone has left a new obstacle in the hallway, or if CoBot's localization has become lost. In the rare case that the localization has failed we are then able to correct it via the same interface. Furthermore, the telepresence interface has been extremely useful for debugging new features for CoBot.

### 3.6.1 The Telepresence Interface

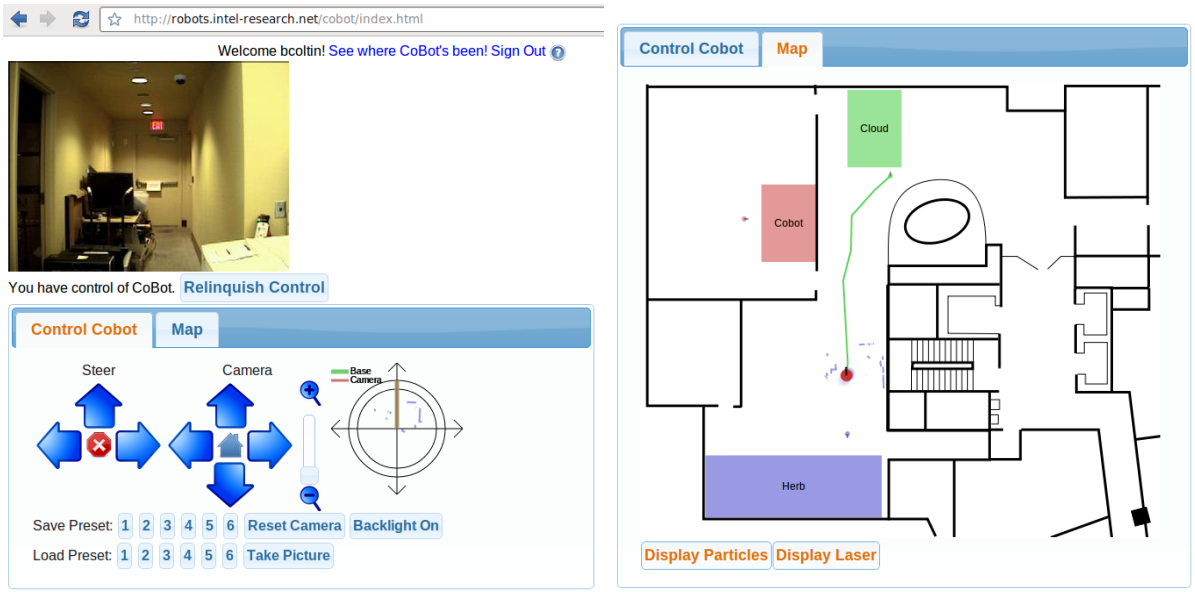
Users may interact with CoBot through a web-based browser interface. With the web-based interface, no special software needs to be installed to interact with CoBot. Furthermore, the software runs on multiple devices— desktop computers, and also mobile devices such as smartphones. The web client repeatedly requests robot state information ten times per second, and requests new images from the robot's camera five times per second. To enable voice communication and show the user's face on the screen, users can call the robot on Skype.

Users may switch between two tabs in the web interface: the Control tab, which contains buttons to control the camera and drive the robot, and the Map tab, which shows the map of the environment and the robot's position (see Figure 3.8). The image from the robot's camera is always displayed above the two tabs in the remote interface.

#### The Control Tab

From the control tab, users can control the robot's camera. There are arrow keys to move the camera, surrounding a "home" button which returns the camera to a forward facing position suitable for driving. Next to these arrows is a slider with which the user can set the zoom level of the camera. The rate of the camera's motion when using the arrow keys depends on the level of the camera's zoom— at higher zoom levels the camera moves more slowly to allow more precise control. All of these commands have visual icons representing their function (such as magnifying glasses with "+" and "-" for zooming, a picture of a house for the home button) and are associated with keyboard shortcuts. When the user clicks on an arrow or presses the associated keyboard shortcut, the arrow button on the screen becomes pressed, providing visual feedback.

Next to arrows for controlling the camera are arrows for steering CoBot. Although CoBot is capable of omnidirectional motion, these arrows only allow turning in place and moving directly




(a) The Control Tab

Figure 3.8: CoBot's web interface, with the Control and Map tabs visible. In the map tab, CoBot's current position, path and LIDAR readings are shown. Users may click on the map to set CoBot's localization position or travel to a point.

forwards. This is the type of interface we believed would be most familiar to users, and which would also cause the least confusion in conjunction with the movable camera. In the center of the arrows is an emergency stop button. The robot can also be moved with the arrow keys on the keyboard, and can be stopped by pressing space. CoBot autonomously performs obstacle avoidance while controlled with the arrow keys, adjusting its velocity to avoid obstacles.

On the right side of the Control tab is a compass-like object which displays the relative orientation of CoBot and its camera, each with a colored "compass needle". The needle representing CoBot's camera always points "north" on the compass, and the needle representing the orientation of the robot's base moves relative to this based on the current value of the camera's pan angle. The LIDAR readings are displayed on the compass so that the user can visualize objects in the immediate vicinity of CoBot. The compass is intended to provide situational awareness to a user who has moved the camera, so that they can tell which way the camera is facing, which is essential when driving with the arrow keys. It also allows an experience user to predict which obstacles CoBot will avoid automatically via the LIDAR readings. By clicking on the compass, users can turn CoBot precisely.

### The Camera Image: Visualization and Control

A 320x240 image from CoBot's camera is displayed at the top of the display, and refreshed every fifth of a second. Network latency is manageable enough that overseas users can control CoBot without difficulty.

In addition to displaying the environment, the image is used for control. When a user clicks on the image, the camera moves to center on the selected point, providing more precise control than the arrow keys. Additionally, by scrolling the mouse wheel with the cursor over the image, the camera zooms in or out (see Figure 3.9). By clicking on the ground plane in the image while holding the shift key, the user can also move CoBot to a specific spot on the floor. With this control scheme, it is irrelevant which direction the camera is facing, and lack of awareness of the robot's camera and orientation is not an issue. Furthermore, moving the robot by clicking on the image does not suffer from the latency problems of the arrow keys. Administrators may move the camera freely to inspect the robot's surroundings without interfering with CoBot's task execution.

Figure 3.9: At left, CoBot-2 views a scene at the default zoom level. At right, the user zooms in to the boxed area at the maximum level of 18X. The powerful zoom functionality of the robot allows users to inspect small or distant objects, and even read text remotely. Only CoBot-2 possesses such a powerful camera.

### The Map Tab

The map is shown as a separate tab so that the user need not scroll the browser window to view the entire interface. It displays a visualization of CoBot's environment, position and orientation. While CoBot moves to a point chosen by the user, the path to its' destination is displayed.

Users set a destination for CoBot to travel to by clicking on the map, then dragging and releasing the mouse button to set the orientation. Additionally, by holding the shift key while

choosing a location, the user can set CoBot's localization position from the Map tab. This feature is useful in the rare event that CoBot's localization becomes lost.

In the Map tab, users may still access much of the functionality from the Control tab, such as driving, emergency stop, and moving the camera, through keyboard shortcuts. The image from the robot's camera remains visible.

### 3.6.2 Monitoring Multiple CoBots

CoBot's interface is excellent for telepresence, but it also is very convenient to monitor the robots. We have another page, available only to administrators, which shows the positions and current tasks of all the robots currently running on a map of the building. To check that newly developed features are working as intended, administrators can view CoBot's progress remotely through the web site and easily detect, stop and fix the robot if something goes wrong.

One unexpected benefit of the telepresence interface is that often, when deploying CoBot, we will have no idea where or even on which floor of the building a robot is on. We can easily check the website, look at the map and locate CoBot while it performs tasks for users.

In addition to monitoring the robot's state from the website, administrators may also view the robot's complete schedule and can cancel any task remotely. They may also create and modify public availability windows to schedule tasks in. Non-administrative users may view the list of public availability slots, and view and cancel the tasks they have requested. Users may also view previous tasks they scheduled, whether the tasks succeeded, failed or were cancelled, and the time the task finished.

## 3.7 Selected Deployment Results

We deployed CoBot on the upper four floors of an office building for a two week period. CoBot was deployed for two hours every weekday and made available to the building occupants. Occupants were alerted of CoBot's availability through email and physical signs posted on bulletin boards and on the robot itself. The deployment times varied each day, and were announced beforehand on CoBot's website.

The response to CoBot's deployment was positive: over one hundred building occupants registered to use CoBot on the website. Users found creative ways to exploit the robot's capabilities, including, but not limited to:

- Sending messages to friends.
- Reminding occupants of meetings.

- Escorting visitors between offices.
- Delivering printouts, paper revisions, inter-office mail, USB sticks, snacks, owed money, and beverages to other building occupants.

Particularly in the first couple days of deployment, we found building occupants following the robot around to see where it was going and how it worked. Task solicitors have used this task to send various items including printouts, paper revisions, USB keys, money owed for lunch, mail received by assistants, and snacks.

We found that occupants scheduled the robot to transport objects between multiple floors of the building more often than they used the multi-floor functionality for other tasks (see Table 3.4). In particular, the transport task saved the task solicitors time because they did not have to travel between floors themselves. However, even the other scheduled tasks utilized the elevator 40% of the time.

Table 3.4: Total number of task requests per task type and the respective number that used the elevator.

| **Task Type** | **Total Requests** | **# Multi-floor** |
|---|---|---|
| Escort | 3 | 2 |
| GoToRoom | 52 | 22 |
| DeliverMessage | 56 | 20 |
| Transport | 29 | 22 |

In fulfillment of the user requested tasks, CoBot travelled a total of 8.7 km, which covered most of the building. CoBot spent

- 6 hours and 17 minutes navigating this distance,
- 36 minutes with a blocked path waiting for a person to move out of its way,
- 1 hour and 2 minutes waiting for help with the elevator,
- 1 hour and 18 minutes waiting for task solicitor help to complete its tasks.

Figure 3.10 shows how much time CoBot took to execute each task, and how that time was apportioned. A total of 140 tasks were completed during the two week deployment, which took 9 hours and 13 minutes. Based on these times, we find that task solicitors quickly responded to the robot's request for help at the start and end of tasks. Building occupants (even those that had never scheduled a task) were willing and able to help the robot in and out of the elevator. This finding supports our model of symbiotic autonomy— humans are willing to help a robot complete its tasks so that the robot is available and capable of performing tasks for them as well.

Although CoBot could be required to wait for human help indefinitely, the task execution times are limited. In general, little more than five minutes per task was spent in the elevator. This

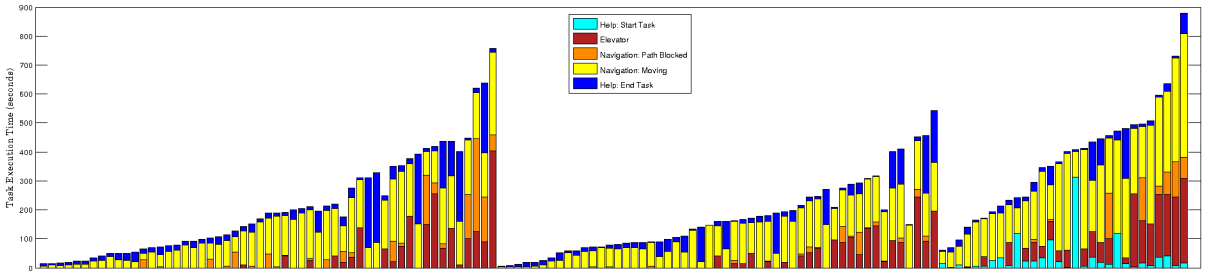



Figure 3.10: Execution times for, from left to right, Deliver Message tasks, Go to Room tasks, and Transport tasks. The breakdown includes 1) waiting for help to start the task, 2) riding the elevator, 3) navigating (not including time blocked by obstacles), 4) waiting blocked by an obstacle, and 5) waiting for help to end the task.

is because if CoBot spends more than five minutes waiting for human help, it sends an email to our research group asking for assistance. Typically, however, occupants helped CoBot and there was no need to do so. Furthermore, if at the end of a task no human pressed the button to indicate that the task was complete, CoBot marked the task as complete and moved on to the next task.

## 3.8 Chapter Summary

The CoBots robustly and reliably complete a variety of tasks in a multi-story building. Users can schedule tasks over the web or on CoBot directly through a graphical UI or through spoken dialogue. The tasks are scheduled on a central server by solving an MIP, and executed by the robot's task execution component. Users can interrupt the robots while they execute tasks to modify their schedules. We have deployed CoBot over multiple years to perform many tasks requested by users.

# Chapter 4

# The Pickup and Single Delivery Problem with Transfers

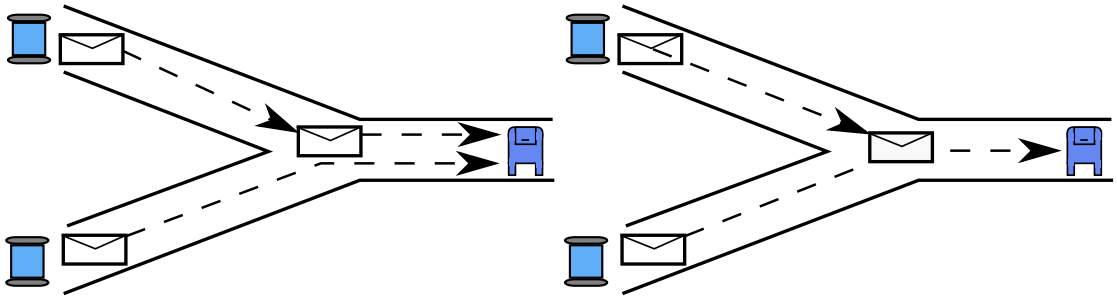
We begin our treatment of algorithms to solve the PDP-T by addressing a simplified version of the problem, the Pickup and Single Delivery Problem with Transfers (PSDP-T). [1] In the PSDP-T, the multiple vehicles retrieve a set of items from multiple locations and deliver them to a *single* central location. There are no capacity constraints, time constraints, or any other special constraints. We allow transfers to occur only at pickup locations. This makes sense in domains such as the CoBots, where human help is needed to transfer items.

The motivation for examining this particular problem stems from user requests for the CoBots, in particular, for multi-object retrieval tasks, as discussed in the previous chapter. In our building, popular requests include retrieving mail from the secretaries' offices and delivering it to the central office, as well as delivering business cards, fliers or candy to many offices and returning the leftovers. All these problems are instances of the PSDP-T.

The PSDP-T, like the general PDP-T, is NP-hard, so we introduce algorithms both to solve it optimally and approximately. In later chapters of the thesis, we introduce algorithms for more general versions of the PDP-T, which also can find solutions for the PSDP-T. However, due to the simplifications in the PSDP-T problem, we are able to construct faster heuristics with theoretical guarantees which only apply to this specific variant of the PDP-T.

In this chapter, we introduce the PSDP-T and formulate the problem as a delivery tree. We propose a two-approximate polynomial time algorithm for the PSDP-T, and a metaheuristic to improve on this solution. We further consider the PSDP with transfers anywhere (PSDP-TA), where transfers are not limited to pickup locations, and show that the optimal solution to the PSDP-T costs at most twice that of the PSDP-TA. We show the effectiveness of these algo-

[1] We previously referred to this problem as the Collection and Delivery Problem with Transfers [31].



(a) Without Transfers (b) With Transfers

Figure 4.1: Two vehicles collect letters and deliver them, minimizing the total distance travelled to conserve energy. Each corridor has length $c$. a Without transfers, the optimal solution has cost $4c$. b With transfers, the optimal cost is $3c$.

rithms in simulation and on physical robots, and demonstrate an approach for two robots to autonomously transfer objects. To our knowledge, this is the first time multiple robots have created and executed a schedule with transfers.

## 4.1 The PSDP-T Problem

As detailed in Chapter 2, we are given a set of vehicles $V$ and items $M$, and the goal is to form a schedule $S$ that minimizes an objective function. For the PSDP-T, we have the further restrictions that all items are delivered to the same location, that is, $\forall m \in M$ $\mathit{end}(m) = f$ for some final delivery location $f$. Furthermore, vehicles do not have ending locations, and $\forall v \in V$ $\mathit{end}(v) = \emptyset$. All vehicles have infinite capacities, and there are no time windows. All vehicles share a single distance metric $d(a, b)$.

For the PSDP-T, we introduce a graph-based problem representation as an alternative to thinking directly in the space of schedules and actions. We construct the complete weighted graph $G = (N, E)$ where

$$\begin{aligned} N &= V \cup L \cup \{f\} \\ L &= \{\mathit{start}(m) : m \in M\}, \end{aligned}$$

and the edge weights are defined by the metric $d$. Then, the goal is to find a path $P_v$ on $G$ for each vehicle $v$ such that all the objects are delivered. In a valid solution, the endpoint of every path $P_v$ is either $f$ or, in the case of a transfer, an intermediate node in another vehicle's path. Every vertex corresponding to $\mathit{start}(m)$ for some $m \in M$ is part of at least one path $P_v$ since every item is delivered (as there are no constraints, time or otherwise, delivering all items is always

feasible). Given the set of paths $P_v$, we can construct a schedule with the actions START, END, PICKUP, etc. as formulated in the earlier chapter.

Many valid sets of paths exist, and we consider two objectives to distinguish between them: a) minimize the total movement energy of the vehicles, corresponding to the sum of edge weights, and b) minimize the completion time, corresponding to the length of the longest path from any vehicle to the destination.

Next, consider the directed graph $T$ which is the union of all the paths of used vehicles $P_v$, where the paths are chosen to either minimize the total distance travelled or the maximum distance travelled to deliver any object. Note that some vehicles may not pickup any items, and these vehicles are not part of $T$.

**Theorem 4.** *There exists an optimal directed graph $T$ that is a tree.*

*Proof.* $T$ is a tree iff it is connected and has $|E| = |N| - 1$ edges. $T$ is connected, since every pickup location and used vehicle is part of a path to $f$. Every vehicle route $P_v$ either ends at $f$ or at a transfer point. If not, travelling to the final location on the path is either needless or retrieves an item that is not delivered to $f$. Furthermore, every route $P_v$ contains $f$ or a transfer point where items are given to a different vehicle nowhere else in the path. If these points do exist elsewhere, a solution of less than or equal cost can be constructed by removing the points due to the triangle inequality. Any items transferred earlier can still be delivered by transferring them at the route's final point instead. Hence, there is an optimal set of paths where each vehicle transfers or delivers items exactly once at the end of its path. Furthermore, there is an optimal solution where each node corresponding to an item pickup is not the endpoint of exactly one path $P_v$, as otherwise a point could be omitted from one of the paths to construct a solution with no worse cost. So there exists a set of paths $T$ which has $|E| = |M| + |V \cap T|$ edges, as each vertex aside from $f$ is not the endpoint of exactly one path $P_v$. $T$ has $|V| = |M| + |V \cap T| + 1$ vertices, and is hence a tree. □

An equivalent formulation for the PSDP-T is to construct a *delivery tree* $D$ with the following properties (see Fig. 4.2):

1. The interior nodes are the pickup locations and the final delivery point $f$.
2. The leaf nodes are a subset of $V$, the starting locations of the used vehicles.
3. Branch points represent transfers of one vehicle's load to another. All but one vehicle at a transfer point remain behind and are not used again.

Our goal is to either find the delivery tree of minimum weight $w(D)$ (minimum movement cost), of minimum weighted depth $\mathit{depth}(d)$ (minimum delivery time), or to minimize a linear combination $\alpha w(D) + (1 - \alpha)\mathit{depth}(D)$ where $0 \leq \alpha \leq 1$.

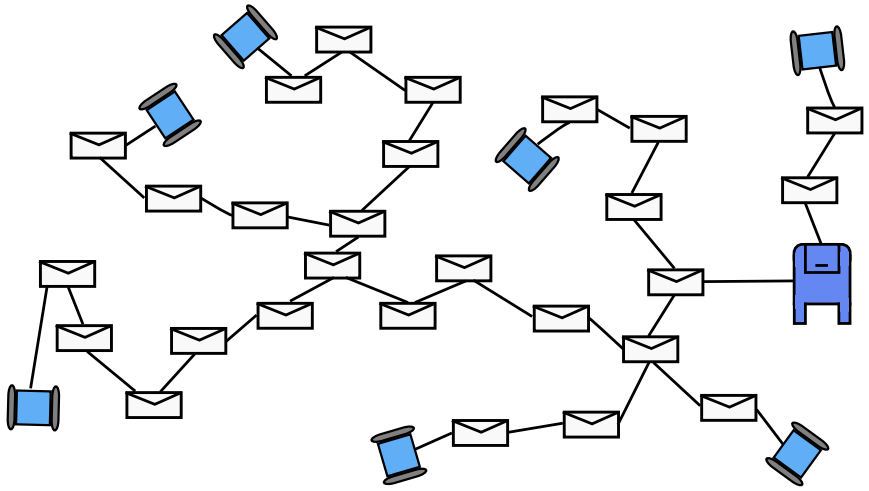


Figure 4.2: The optimal solution to the PSDP-T can be formulated as a delivery tree. Edges represent the motion of a single vehicle. Where the tree branches, all the vehicles at that branch point transfer their entire load to a single vehicle which continues alone.

We show that the PSDP-T problem is NP-hard by reducing the TSP to the PSDP-T. In the TSP, a salesman aims to find the minimum distance Hamiltonian tour that visits every city in a set $V$ exactly once, and return to the initial city. The TSP can be reduced to the PSDP-T problem by setting the cities $V$ as the item pickup locations. We have one vehicle $r$, which begins at the same location as the travelling salesman. We set the dropoff location $f$ to be the same location, $r$. So if the PSDP-T can be solved in polynomial time, so can the TSP. Hence the PSDP-T is NP-hard.

## 4.2 PSDP-T Algorithms

We present algorithms to solve the PSDP-T optimally using mixed integer programming, and approximately using a minimum spanning tree.

### 4.2.1 Optimal Approach

We formulate the PSDP-T as a mixed integer program.

$$\forall a, b \in N \quad a \neq b, 0 \leq x_{a,b} \quad \leq 1, \text{ int} \tag{4.1}$$

$$\sum_{n \in N} x_{f,n} \quad = 0 \tag{4.2}$$

$$\forall l \in L \quad \sum_{n \in N} x_{l,n} \quad = 1 \tag{4.3}$$

$$\forall v \in V \quad \sum_{n \in N} x_{v,n} \leq 1, \sum_{n \in N} x_{n,v} \quad = 0 \tag{4.4}$$

$$\forall n \in (L \cup \{f\}) \quad \sum_{n \in N} x_{n,n} \quad \geq 1 \tag{4.5}$$

$$\forall U \subset N \quad U \neq \emptyset \ \sum_{e \in \delta(U)} x_e \quad \geq 1 \tag{4.6}$$

$$\forall a, b \in N,\ v \in V,\ a \neq b \quad 0 \leq p_{e_{a,b},v} \quad \leq 1, \text{ int} \tag{4.7}$$

$$\forall v \in V,\ l \in L \quad p_{e_{v,l},v} = x_{v,l},\ p_{e_{l,v},v} = 0,\ p_{e_{f,l},v} \quad = 0 \tag{4.8}$$

$$\forall v_1, v_2 \in V, l \in L \cup \{f\} \quad p_{e_{v_2,l},v_1} \quad = 0 \tag{4.9}$$

$$\forall l_1, l_2 \in L \cup \{f\}, l_1 \neq l_2 \quad \sum_{v \in V} p_{e_{l_1,l_2},v} \quad \geq x_{l_1,l_2} \tag{4.10}$$

$$\forall l_1 \in L, l_2 \in L \cup \{f\} \setminus l_1, v \in V \quad x_{l_1,l_2} - 1 + \sum_{l_3 \in L \setminus l_1,l_2 \cup \{f\}} p_{e_{l_3,l_1},v} \quad \leq p_{e_{l_1,l_2},v} \tag{4.11}$$

$$\forall v \in V \quad \sum_{e \in E} p_{e,v} d(e) \quad \leq G \tag{4.12}$$

$$\text{Objective: } \min \alpha \left( \sum_{a,b \in N,\ a \neq b} x_{a,b} d(a,b) \right) + (1 - \alpha) G \tag{4.13}$$

We solve for binary variables $x_{a,b}$, which indicate whether the directed edge from node $a$ to node $b$ is in the solution (Eq. 4.1). The final destination $f$ has zero outgoing edges (Eq. 4.2), each location $l \in L$ has exactly one outgoing edge (Eq. 4.3), and each vehicle $v \in V$ has zero incoming edges and at most one outgoing edge, but may have zero (Eq. 4.4). The vertex $f$ and each point $l \in L$ have at least one incoming edge (Eq. 4.5)

The formulation as it stands still allows subtrees, in which the delivery tree is not connected. To address this, we introduce constraints similar to the subtour elimination constraints that are used to formulate the TSP as an MIP (Eq. 4.6), where $\delta(U)$ is the set of edges which link $U$ and $N \setminus U$.

This MIP formulation gives paths which solve the PSDP-T. The graph is connected (aside from unused vehicles), so every item is part of a vehicle's path to $f$. The graph has $|V| + |L| = |N| - 1$ edges since each used vehicle and item has exactly one outgoing edge, and is hence a tree. The vehicles are the only leaf nodes in the tree, since they have one edge while the items have at least two. We minimize the total distance $D = \sum_{e \in E} d(e)$ travelled by all the vehicles.

Alternatively, to minimize the time taken to deliver all items (the length of the longest path from a vehicle to the goal) we define binary variables $p_{e,r}$ which indicate whether the directed

edge $e$ is part of the path from $r$ to $f$ (Eq. 4.7). Edges from vehicles are part of that vehicle's path if the edge exists. No edges exist *to* vehicles, and $f$ has no outgoing edges (Eq. 4.8). Edges with vehicles as nodes are not part of another vehicle's extended path (Eq. 4.9). Each edge between retrieval and delivery points that is in the solution is part of the path for at least one vehicle (or more, with transfers) (Eq. 4.10). Finally, edges between two retrieval or delivery locations are only on a vehicle's path to $f$ if an incoming node was also on the extended path (Eq. 4.11).

We define a variable $G$ to be the length of the longest extended path. Then we add constraints so that $G$ is greater than the length of every vehicle's extended path (Eq. 4.12). Our objective function is then $\min G$, or a weighted sum of the two objectives, $\min \alpha D + (1-\alpha)G$ (Eq. 4.13).

### 4.2.2 Minimum Length Approximation

Computing the exact solution to the PSDP-T is difficult since the problem is NP-hard. Hence we are interested in approximation algorithms, which find a solution in polynomial time that is not optimal, but provably close to optimal.

The approximation algorithm we introduce generates two-approximate solutions in terms of total distance to the PSDP-T using only a single vehicle. In fact, a two-approximate solution to the PSDP-T is the best guarantee we can possibly make with only a *single* vehicle. If $O_n$ is the optimal solution with $n$ vehicles, and $O_1$ the optimal solution with any one of those vehicles, Fig. 4.3 shows a case where $w(O_1)$ approaches arbitrarily close to $2w(O_n)$. With multiple vehicles, the heuristic further reduces both the distance travelled and the delivery time.

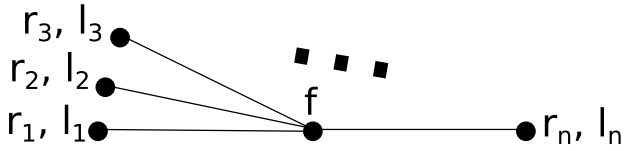



Figure 4.3: Robots and items are situated at the end of $n$ hallways of length one emanating from the delivery point $f$. $O_n = n$, where every vehicle travels to $f$. $O_1 = 2n - 1$, where one vehicle retrieves every item.

Our approximation is based on the minimum spanning tree two-approximate heuristic for the TSP, but is a generalization for multiple vehicles that transfer items. In this approximation, the minimum spanning tree $MST$ over all the vertices is generated. Since any solution to the TSP must visit all vertices and return to the original solution, $w(MST) \leq w(O_{TSP})$, where $O_{TSP}$ is the optimal solution to the TSP. Then, a tour $T$ is constructed that visits all the vertices and returns to the starting point while traversing each edge of $MST$ at most twice. So $w(T) \leq 2w(MST) \leq 2w(O_{TSP})$. The algorithm is shown in Algorithm 1. First, we construct the complete graph $G$

with pickup locations $L$ and drop-off location $f$ with edge lengths determined by the distance metric $d$, and its minimum spanning tree $T$. Next, choose the edge $e$ of lowest weight from a node in $T$ to a node in $v \in V$, and add $e$ and $v$ to $T$ to construct $T'$.

**Algorithm 1** *psdp_t*$(L, f, V)$: Construct a delivery tree given the set of pickup points $L$, the dropoff point $f$, and vehicles $V$. $N_T(n)$ gives the set of neighbors of $n$ in $T$.

$G \leftarrow$ *complete_graph*$(L \cup \{f\}, d)$
$T \leftarrow$ *mst*$(G)$
$s, n \leftarrow \text{argmin}_{s \in V, n \in N(G)}\, d(s, n)$
$T' \leftarrow$ *deliv_tree*$(V, L, f, T \cup$ *edge*$(s, n))$
**for** $v \in V, v \notin T'$ **do**
  $n \leftarrow \text{argmin}_{n \in N(G), N_{T'}(n) \cap V = \emptyset}\, d(v, n)$
  $T'' \leftarrow$ *deliv_tree*$(V, L, f, T' \cup$ *edge*$(v, n))$
  **if** *cost*$(T'') <$ *cost*$(T')$ **then**
    $T' \leftarrow T''$
  **end if**
**end for**
**return** $T'$

If $O$ is the delivery tree for the optimal solution, then $w(T) \leq w(O \setminus V)$, since $T$ is the minimum spanning tree over the same nodes. Since a valid delivery tree must have at least one edge connected to a vehicle , and we chose the minimum one, $w(T') \leq w(O)$. We call $T'$ an *intermediate delivery tree*, since it can be used to construct a delivery tree but not all leaf nodes are vehicles.

We then construct a tour $P$, starting at $v$ and ending at $f$, which visits each vertex in $T'$ at least once with the procedure *deliv_tree* (see Alg. 2). When $T'$ contains only a single vehicle, this algorithm is equivalent to the two-approximate TSP approximation, which traverses each edge at most twice. The *deliv_tree*$(T)$ algorithm extends this heuristic to multiple vehicles which can transfer items, and creates a valid delivery tree with weight at most $2w(T)$. Thus, $w(P) \leq 2w(T) + w(e) \leq 2w(O \setminus V) + w(e) \leq 2w(O \setminus R)$. So our single vehicle algorithm is two-approximate to the optimal multi-vehicle solution in terms of total distance.

We can lower the cost further with multiple vehicles, although no better guarantees on the approximation bound are obtained. We begin with the constructed intermediate delivery tree for a single vehicle, $T'$, and for each vehicle $v$, greedily add the shortest edge from $v$ to a node in $T$ to the tree $T'$. We construct a new delivery tree from $T'$ with *deliv_tree*, and keep the edge and vehicle in the intermediate delivery tree $T'$ if and only if the delivery tree's cost decreases according to our objective function. We then attempt to add the next vehicle to the updated $T'$, iterating through every vehicle. This procedure still gives a two-approximation to the PSDP, and additional vehicles will sometimes decrease both the total distance travelled and the time to

**Algorithm 2** *deliv_tree*$(V, L, f, T)$: Construct a delivery tree given the set of pickup points $L$, the dropoff point $f$, vehicles $V$, and an intermediate delivery tree $T$.

$A = \{l \in L : \exists v \in V$ s.t. $l \in$ *path(v, f)*$\}$
$T' =$ Null Graph
**for** $v \in V$ **do**
  $n = v$, $P = []$
  **while** $n \notin T'$ **do**
    Append $n$ to $P$
    If $n = f$, break
    Choose $n' = x \in N_T(n)$ s.t. $x \notin P$ and $x \notin A$
    If $\nexists n'$, $n' = x \in N_T(n)$ s.t. $x \notin P$ and $x \in A$
    $n = n'$
  **end while**
  $P' = P$ with duplicate vertices removed
  $T' = T' \cup P$
**end for**
**return** $T'$

completion, depending on the problem instance. See Fig. 4.4 for an example of the algorithm's results.

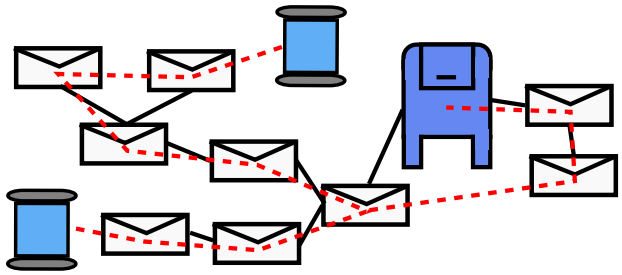


Figure 4.4: An example of the approximation algorithm. Solid black lines indicate edges on the minimum spanning tree, and dashed lines show the generated delivery tree.

We have constructed a two-approximate delivery tree in polynomial time that uses multiple vehicles, since the cost of the multi-vehicle heuristic is at most the cost of the single-vehicle heuristic which is two-approximate.

### 4.2.3 Improvement with Local Search

Next, we introduce a metaheuristic to improve upon the two-approximate solution with local search techniques. Specifically, we make use of simulated annealing [63]. Simulated annealing is a metaheuristic that begins at some state, and chooses a random "neighbor" of that state. With probability $accept(e, e', t)$ the new state is accepted as the current state, where $e$ is the "energy"

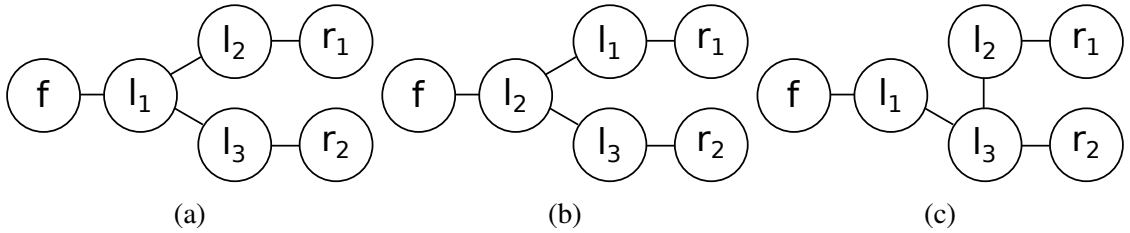



Figure 4.5: *(a)* The initial state. *(b)* A neighbor found by swapping $l_1$ and $l_2$ (the edges linking them to other nodes are different). *(c)* A neighbor with $l_2$ grafted from $l_1$ to $l_3$.

(in our case, the cost) of the current state, $e'$ is the energy of the new state, and $t$ is the temperature, or the fraction of iterations of the algorithm currently completed. If the new state is rejected we remain at the current state and repeat with a new neighbor. The algorithm continues either for a fixed number of iterations or until the energy crosses some threshold, when the best solution that has been encountered thus far is returned.

To apply simulated annealing to the PSDP, we must define a starting point, an energy function, an acceptance probability, and a function to return random neighbors of a state. We search over the underconstrained intermediate delivery trees rather than strict delivery trees, as this allows us to develop a broader concept of neighboring solutions that is more closely tied to the approximation heuristic. We use as a starting point the tree generated by our fast multi-vehicle heuristic.

The energy function is the cost function of the delivery tree constructed with *deliv_tree*, and it incorporates both the total weight and depth of the tree as a function of $\alpha$. The acceptance probability is 1 if $e' < e$, and $e^{\frac{e-e'}{t}}$ otherwise, a standard acceptance function frequently used in the literature.

Neighboring states are found either by *swapping* two non-vehicle, non-destination nodes on the intermediate delivery tree (that are not vehicles or $f$), swapping their neighbor sets, or by *grafting* one branch of the tree at a transfer point onto a neighboring node, replacing a single edge. See Fig. 4.5 for examples.

We run the simulated annealing algorithm for a thousand iterations. Every hundred iterations we restart from the best solution found thus far to explore the most promising regions more thoroughly. In practice, simulated annealing improves upon the solutions of the two-approximate heuristic, as we demonstrate later through experimental results.

## 4.3 Transfers at Any Location

Next, we consider the general case of PSDP-TA, where items can be transferred anywhere, rather than only at vertices in $L$ as in the PSDP-T. We show that the optimal solution for the metric PSDP-T, $O_v$, is within a factor of two of the cost of the optimal solution to the PSDP-TA, $O_a$, when minimizing total distance travelled.

The PSDP-TA is closely related to the Steiner tree problem, in which a set of points must be connected by the shortest possible set of edges. "Helper" points may be added (transfer points) to construct a solution. The PSDP-TA is different in that all the leaf nodes of a valid delivery tree must be vehicles or the final destination.

**Theorem 5.** $O_v \leq 2O_a$

The full proof is presented in our previous work [31] and relies on the properties of Steiner trees. In the case where our metric is Euclidean, we prove the stronger claim that $O_v \leq \frac{2}{\sqrt{3}}O_a$, given that an open conjecture regarding the Steiner ratio holds [58].

*Proof.* Given a delivery tree $T$ with optimal cost $O_a$ with transfers at any point (the transfer points are separate nodes in the tree), we construct a delivery tree $T'$ with transfers only at vertices and cost $O_v \leq 2O_a$. Let $X$ be the set of transfer points in $T$.

First, construct the forest $F$ with vertices $X \cup N(X)$, where $N(X)$ is the set of neighboring vertices to $X$ in $T$. Add all edges between the vertices of $F$ in $T$ to $F$ as well. For each connected component $F_i$ of $F$ (which may include multiple transfer points), define $S_i$ to be the complete graph with vertices $F_i \setminus X$. Then $w(F_i) = w(\textit{steiner}(S_i))$, where *steiner* gives the minimum Steiner tree. If this were not the case, then either the minimum Steiner tree could be used to construct a delivery tree of lower cost, or we do not have the minimum Steiner tree.

Let $S = \cup S_i$. Construct $T' = (T \setminus X) \cup \textit{msf}(S)$, where $\textit{msf(S)} = \cup_i \textit{mst}(S_i)$. $T'$ is a valid delivery tree where the items are transferred at retrieval or dropoff points.

Then $O_a = w(T) = w(T \setminus X) + w(F)$, since $T = (T \setminus X) \cup F$. Furthermore, $w(F) = \sum w(F_i) = \sum w(\textit{steiner}(F_i \setminus X))$. Similarly, $O_v \leq w(T') = w(T \setminus X) + w(\textit{msf}(S)) = w(T \setminus X) + \sum_i w(\textit{mst}(F_i \setminus X))$. The weight of a minimum spanning tree is bounded by the Steiner ratio $\kappa$ such that $w(\textit{mst}(G)) \leq \kappa w(\textit{steiner}(G))$. Hence, $O_v \leq w(T \setminus E) + \kappa \sum_i R_i \textit{steiner}(R_i \setminus E) \leq \kappa O_a$.

If our distance function is a metric, than $\kappa = 2$ [108], and the optimal solution to the PSDP-T is two-approximate to the PSDP-TA. If the distance function is Euclidean, $\kappa$ is conjectured to be $\frac{2}{\sqrt{3}} \approx 1.1547$. This conjecture is believed to be true, but is still open and unproven [58]. If it holds, then for the Euclidean case PSDP-T is a $\frac{2}{\sqrt{3}}$-approximation to PSDP-TA. Hence, our two-approximate heuristic for the PSDP-T provides a four-approximate heuristic for metric PSDP-TA

and a $\frac{4}{\sqrt{3}} \approx 2.31$-approximate for the Euclidean PSDP-TA, assuming the conjecture regarding the Euclidean Steiner ratio holds. □

## 4.4 Experimental Results

We tested our algorithms in simulation to demonstrate the effects of transfers and our algorithms on large problems, as well as on the CoBot robots. We also developed the CreBot robots which autonomously transfer items.

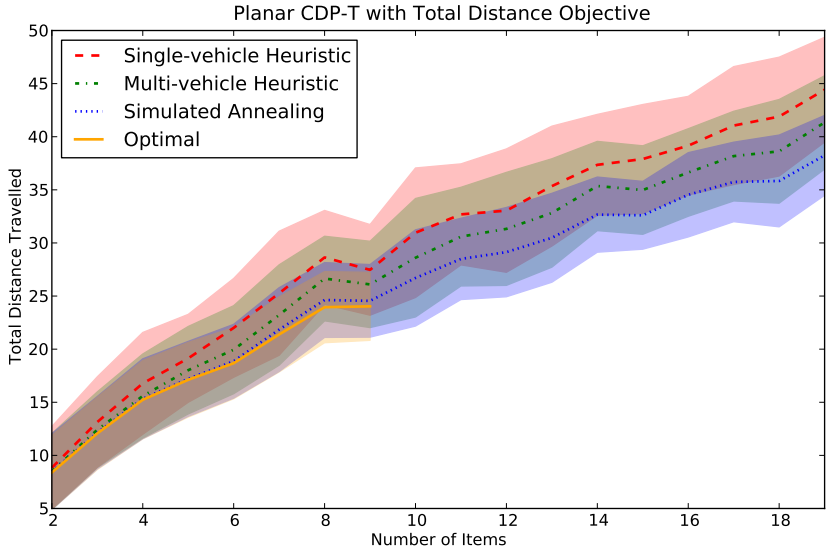
### 4.4.1 Simulation Results

To validate our approach, we tested the algorithms in two scenarios. We varied the number of items to retrieve, and created 50 random problem instances for each set of parameters. We solved each problem optimally (when feasible), with the single and multi-robot two-approximation algorithms, and with simulated annealing.

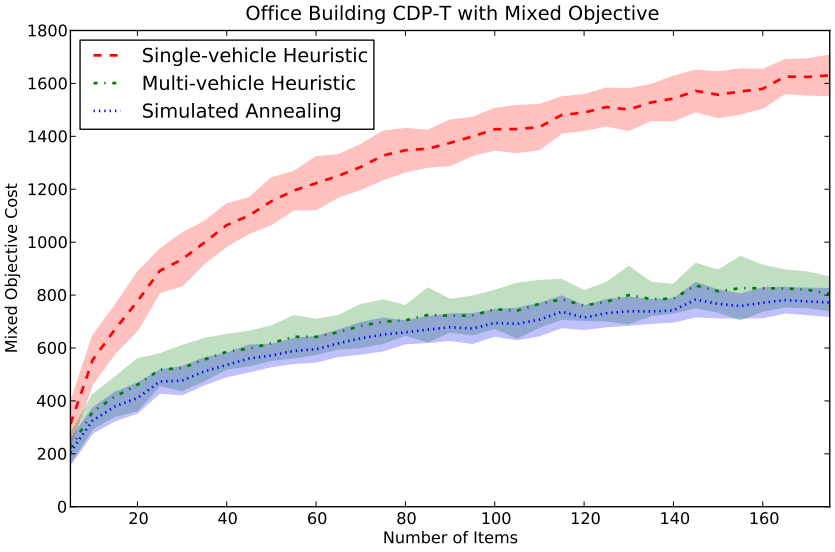
In the first scenario, we chose random pickup, delivery, and starting points from a plane, using a Euclidean distance function. Figure 4.6a shows results for three robots and up to twenty items. Each phase of the algorithm gives an improvement. The simulated annealing solution is near-optimal, when the optimal solution is found.

In the second scenario, points were chosen randomly from four floors of an office building in a simulation of the mail collection task. The distance function was an empirical estimate of robot travel times. The objective was to minimize a weighted sum of the moment energy cost and the delivery time ($\alpha = 0.5$). Figure 4.6b shows the results with 25 robots and up to 175 items. Each level of the algorithm offers a significant improvement, particularly the multi-vehicle heuristic, since it achieves much better depths than the single robot approximation. We are unable to find the optimal solutions for problems of this size. Note that a uniform distribution among rooms in the building may not be a realistic representation of some scenarios, and results for these other scenarios may differ.

On an Intel 2.83 GHz Core2 Quad CPU, the two heuristics ran in under a second for all instances of up to 500 vertices (these instances are not shown). Simulated annealing ran in under fifteen seconds for every instance. The optimal MIP formulations were solved with `lpsolve`, with each attempt aborted after thirty seconds. These results show that the approximation algorithms can solve large problems quickly, and provide near-optimal solutions when comparison is possible.




(a)




(b)

Figure 4.6: Results for the PSDP-T algorithms with *(a)* three robots and up to twenty items to retrieve under the total distance objective function in the planar domain, and *(b)* 25 robots and up to 175 items to retrieve under the mixed objective function in the office building domain. Lines indicate the mean objective function cost, and the filled area shows the standard deviation across trials. Darker areas indicate overlap.

### 4.4.2 Illustrative Deployments on CoBots

We have deployed our PSDP-T heuristic on the CoBot robots (as introduced in Chapter 3), and with them we show that transferring items can reduce energy consumption and delivery time. The CoBots cannot physically transfer items, and so they ask humans to help. The previously introduced centralized web server accepts user requests and sends schedules to the CoBots. The CoBots pickup and deliver items by arriving at an office and asking a human to place or remove the items.

We examine three scenarios where two CoBots were deployed to collect and deliver objects in an office building. The problem setups and the planned paths, both with transfers and without, are shown in Fig. 4.7. For each scenario, we recorded the time it took the robot(s) to complete the task and the total combined distance travelled by all of the robots. To reduce the variance in our experiments, we assumed that humans were always immediately available to place items in the CoBots' baskets and to help transfer items. However, the times recorded with transfers do include the time to ask and thank a human for their help, and are shown in Table 4.1.

| | With Transfers | | Single Robot | |
|---|---|---|---|---|
| Scenario | Time (min.) | Dist.(m) | Time (min.) | Dist. (m) |
| 1 | 4:22 | 229.35 | 8:18 | 287.12 |
| 2 | 5:52 | 220.68 | 7:55 | 238.66 |
| 3 | 7:00 | 230.56 | 9:36 | 272.37 |

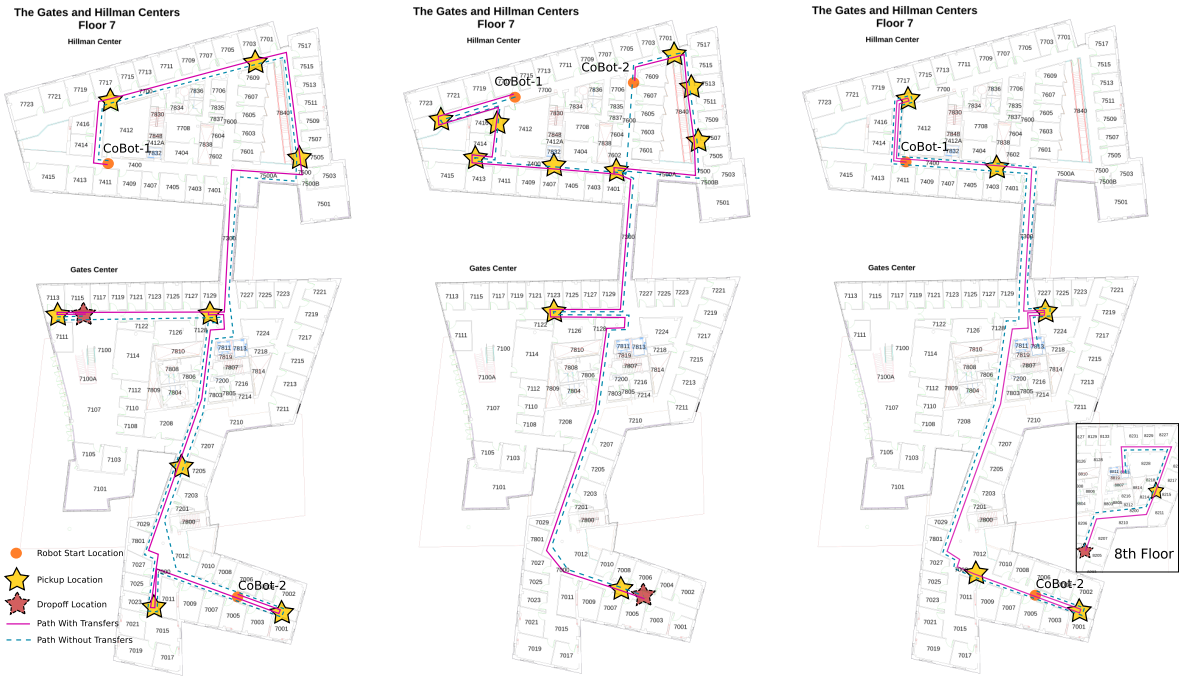
Table 4.1: Deployment Results for Selected Scenarios

In all three scenarios, using multiple robots with transfers reduced both the time to complete the task and the total distance travelled. This may not be the case in all scenarios, depending on the problem setup. However, we have demonstrated that in many scenarios, transferring items between robots can and does reduce the energy consumed and the time taken to complete the task.

### 4.4.3 Autonomous Transfers with CreBots

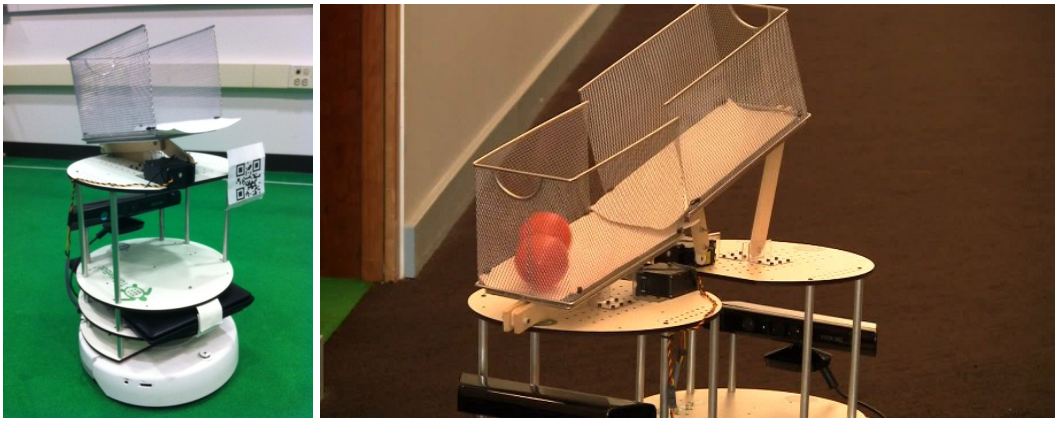
We have designed the CreBots to transfer items autonomously. They consist of an iRobot Create as a base, with Willow Garage Turtlebot shelves, a laptop, Kinect RGB-D camera, and a custom-built tilting tray on top to transfer items (see Fig. 4.8a).

Unlike the CoBots, the CreBots transfer items autonomously. To do so, two CreBots head towards the same location based on their localization information. The robots stop either when they reach the destination, or are blocked within two meters of the destination (presumably by




(a) (b) (c)

Figure 4.7: The paths planned by the approximation algorithm and taken by the robots, with and without transfers, for a Scenario 1, b Scenario 2, and c Scenario 3.



(a) (b)

Figure 4.8: a A CreBot robot, with a Create base, laptop computer, Kinect RGB-D camera, tilting tray, and QR code for alignment. b One CreBot transfers an item to another during a collection and delivery task.

the other robot). Next, the robots send each other their localization positions wirelessly and turn to face each other.

Localization is accurate and robust enough for a rough alignment, but not precise enough to transfer objects based solely on localization. For fine alignment, the robots are each equipped with a QR code which is detected by the Kinect and used for precise docking. The transferrer advances slowly until it comes within the range it can detect the QR code using an off the shelf library, which measures the QR code's bounding box. The transferrer computes the distance and angle to the QR code from its known size and camera parameters, then rotates in place to align with the QR code. The transferrer continues to move forwards until its bump sensor is triggered, and dumps its load. When dumping, the tilting tray shakes back and forth to ensure that all the objects are dislodged. The CreBots have successfully executed collection and delivery tasks with transfers.[2]

## 4.5 Chapter Summary

We introduced the Collection and Delivery Problem with Transfers, and showed that the solution comes in the form of a delivery tree. We solved the PSDP-T optimally with an MIP, and bound the cost with transfers anywhere in terms of the cost when transfers occur only at vertices. We introduced a two-approximate algorithm under the minimum length objective, and proposed a heuristic and metaheuristic to find solutions of lower depth while maintaining the bound on total length. We bounded the benefit from transferring items at any location rather than only at pickup locations. Furthermore, we demonstrated the effectiveness of our heuristics on large real-world problem instances, and showed the feasibility of transfers between physical robots.

[2]Video available at http://youtu.be/pzXv7p_aZhE.

# Chapter 5

# Heuristics for the nTPDP-T

In this chapter, we address the No Times Pickup and Delivery Problem with Transfers (nTPDP-T). The nTPDP-T differs from the previous chapter's PSDP-T in the following three ways:

1. In the PSDP-T all items have the same destination, in the nTPDP-T items may have distinct destinations.
2. In the nTPDP-T, vehicles may have their own destinations in addition to starting positions.
3. Vehicles in the nTPDP-T have limited capacities.
4. Vehicles may have a maximum total number of items they may transport during the entirety of their route, $mni(v)$. Once $mni(v)$ items are picked up or received, no more items may be transported by that vehicle, even if it delivers the other items. This particular constraint is considered only in this chapter of the thesis.

As in the PSDP-T, in the nTPDP-T we do not consider time constraints.

We present three heuristics to plan routes for items and vehicles for the nTPDP-T which incorporate transfers: 1) a greedy heuristic, 2) an auction algorithm, and 3) an approach based on graph search. Each algorithm involves a trade-off between solution quality and computational cost. Here we set aside time constraints to focus only on minimizing fuel usage. We assume that assignments are made centrally by an agent with full information.

We test these three algorithms extensively in simulation, and show that in certain cases, transferring items gives a significant improvement over not transferring items. We also demonstrate the heuristics' effectiveness in the real world, on a physical map using the trips of taxi passengers.

As this problem is more general than the PSDP-T, we cannot provide theoretical guarantees as we were able to in the previous chapter. In later chapters we build upon the heuristics for the nTPDP-T to create algorithms for still more general PDP-Ts.

## 5.1 Heuristics for the nTPDP without Transfers

We present two algorithms which take a set of routes without transfers as input, and output a modified schedule to deliver all items with transfers. Hence, we first introduce two algorithms to form schedules without transfers, a greedy approach and an auction approach. The auction approach is similar to [1] and [64]. An alternative algorithm, such as set cover [61], could be used instead.

### 5.1.1 The Greedy Approach

In the greedy approach, we iterate through every item $m$ and vehicle $v$, insert the PICKUP $(m)$ and DELIVER$(m)$ actions into $S^v$ at the points of lowest cost without rearranging the other actions. We then choose the vehicle / item pair $(v, m)$ which increases the cost $c(S)$ the least, and assign item $m$ to vehicle $v$ with the previously discovered best action insertion points. We repeat the process of greedily assigning items to vehicles until no unassigned items remain.

**Algorithm 3** `greedy_nt`$(V, P)$: greedily form schedules $S^v$ for vehicles $v \in V$ to delivery all items $m \in M$.

**for** $v \in V$ **do**
  $S^v \leftarrow \langle$START, END$\rangle$
**end for**
$A = \emptyset$
**while** $|A| < |M|$ **do**
  $v, m \leftarrow \text{argmin}_{m \in M \setminus A, v \in V}$ `c(route_insert`$(S, v, \langle$PICKUP $(m)$, DELIVER$(m)\rangle)$
  $S \leftarrow$ `route_insert`$(S, v, \langle$PICKUP $(m)$, DELIVER$(m)\rangle$
  $A \leftarrow A \cup \{m\}$
**end while**

The details are presented in Algorithm 3. The procedure `route_insert`$(S, v, a)$ uses brute force search to find the optimal placement to insert the actions in $a$ into the schedule $S^v$, while maintaining the new actions' ordering, and $v$'s capacity and maximum item number constraints. The function returns the resulting schedule, or returns failure if no route is available. We assume the distance function $d$ returns $\infty$ for an invalid schedule. We assume that a valid schedule always exists to transport all the items for the nTPDP-T. Since there are no time constraints, this is the case as long as the maximum number of item constraints can be satisfied: that is, $\sum_{v \in V} mni(v) \geq |M|$.

As specified in Algorithm 3, the greedy algorithm's runtime is $O(|M|^2|V|I^2)$ where $I = \max_{v \in V} mni(v)$. However, calls to `route_insert` will not change across iterations of the while loop, except for vehicles which were assigned an item in the previous iteration of the loop.

By caching these values, the algorithm's complexity becomes $O(|M||V|I^2)$.

### 5.1.2 The Auction Approach

Next, we present an auction approach to solve the rideshare problem without transfers. The auction consists of rounds, in which each vehicle places a single bid for the item that it can transport at lowest additional cost. For each item, the vehicle that can transport it at lowest additional cost is declared the winner, and the item's pickup and delivery are inserted into that vehicle's schedule. Rounds of the auction continue until all items are assigned to vehicles. See Algorithm 4 for the full algorithm. The auction algortihm is less greedy than the previous algorithm in the sense that it does not depend on the order the items or vehicles are examined in.

**Algorithm 4** `auction_nt`$(V, M)$: form a schedule $S$ for vehicles $V$ to deliver all items in $M$ via auction

**for** $v \in V$ **do**
  $S^v \leftarrow \langle \text{START}, \text{END} \rangle$
**end for**
$A = \emptyset$
**while** $M \setminus A \neq \emptyset$ **do**
  $\forall v \in V\ bid_v \leftarrow \infty, best_v \leftarrow \emptyset$
  **for** $m \in M \setminus A$ **do**
    $v \leftarrow \text{argmin}_{v \in V}$ `route_insert_cost`$(S, v, \langle \text{PICKUP}(m), \text{DELIVER}(m) \rangle)$
    $c \leftarrow$ `route_insert_cost`$(S, v, \langle \text{PICKUP}(m), \text{DELIVER}(m) \rangle)$
    **if** $c < bid_v$ **then**
      $bid_v \leftarrow c, best_v \leftarrow r$
    **end if**
  **end for**
  **for** $v \in V$ **do**
    **if** $best_v \neq \emptyset$ **then**
      $S \leftarrow$ `route_insert`$(S, best_v, \langle \text{PICKUP}(m), \text{DELIVER}(m) \rangle)$
      $A \leftarrow A \cup \{best_v\}$
    **end if**
  **end for**
**end while**

The worst-case runtime complexity of the auction is $O(|P|^2|V|M^2)$. However, since many items will be assigned to vehicles in the same round, in practice it often runs faster than the greedy algorithm.

One benefit of the auction algorithm is that it easily lends itself to a distributed implementation. Requests could be entered for items and sent to many vehicles. Then the vehicles could, in

a distributed manner, determine the costs to add the various items to their routes and place bids. The item senders could then select the lowest bidder to transport the item.

## 5.2 Heuristics with Transfers for the nTPDP-T

Next, we present three heuristics for solving the ridesharing problem with transfers: a greedy approach, an auction approach, and a graph-based approach. The greedy and auction algorithms take as input a solution to the rideshare problem without transfers, while the graph-based approach constructs a solution from scratch.

However, before presenting these algorithms in detail, we look at the problem of how to select a geometric transfer point at which two vehicles can transfer items. Each of the algorithms for ridesharing with transfers must solve this subproblem.

### 5.2.1 Finding a Transfer Point

Given an edge on vehicle $v_a$'s route between the locations $\langle a_1, a_2 \rangle$, and an edge on vehicle $v_b$'s route between two locations $\langle b_1, b_2 \rangle$, our goal is to select a transfer point $p$ such that $\sum_{v \in a_1, a_2, b_1, b_2} d(p, v)$ is minimized.

The point $p$ is known as the optimal meeting point. In the case where the map is Euclidean, this is called the Weber problem and the point $p$ is called the geometric mean. Although the idea of the optimal meeting point is quite simple, it is difficult to compute. In fact, it has been shown that no closed form solution exists. However, numerous algorithms have been developed to find the optimal meeting point with gradient descent and other techniques, including a near-optimal solution for general maps [117].

We introduce a function $\texttt{tpoint}(a_1, a_2, b_1, b_2)$ which returns a proposed transfer point. Our focus is not on finding the individual transfer points, but on finding the overall schedule of when the vehicles should transfer. To be able to evaluate many transfer points efficiently, we use a fast heuristic rather than the computationally expensive near-optimal methods. We simply consider each of the three possible pairings of the points $a_1, a_2, b_1$, and $b_2$, and find the intersection points of the shortest paths between each set of paired points. We choose the point which creates the lowest edge cost as a proposed transfer point. On a general map, none of the shortest paths may intersect, in which case we do not place a transfer point between these line segments.

Although we choose a non-optimal approach for selecting a transfer point, it should be noted that any other approach from the literature could be substituted to find a better solution at a cost of additional computation.

### 5.2.2 Splitting an Item's Route

In addition to the `tpoint` function, we introduce a helper routine, `split_route`$(S, m, a, b)$, which adds a transfer of the item $m$ between vehicle $a$ and vehicle $b$ in the schedule $S$. Initially, item $m$ is already transported all or in part by vehicle $a$, and $b$ does not transport item $m$. This means there is some action $R$ in $S^a$ where item $m$ is picked up or received from another vehicle, and an action $D$ where $m$ is delivered or transferred to another vehicle. Table 5.1 shows an example initial schedule.

| $S^a$ | | $S^b$ | |
|---|---|---|---|
| Name | Action | Name | Action |
| $S^a_1$ | START | $S^b_1$ | START |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $R$ | PICKUP $(m)$ **or** RECEIVE$(m, c, l_1)$ | $\vdots$ | $\vdots$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $D$ | DELIVER$(m)$ **or** TRANSFER$(m, d, l_2)$ | $\vdots$ | $\vdots$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $S^a_{\lvert S^a \rvert}$ | END | $S^b_{\lvert S^b \rvert}$ | END |

Table 5.1: Schedule Before Route Splitting

The function `split_route` modifies the schedule such that vehicle $b$ will transport the item for part of the way. Vehicle $b$ could execute the receiving action $R$ and then transfer the item to vehicle $a$, as in Table 5.2. Alternatively, $a$ could execute action $R$, then transfer the item to $b$, and $b$ could execute the delivery action $D$, as in Table 5.3.

The `split_route` algorithm effectively divides an item's transport between two vehicles, and can do so recursively. The full procedure is described in Algorithm 5. We first select a transfer point. Then, similar to the insertion heuristics presented in the previous section, we do a brute force search over insertion points for the new transfer actions into the schedule, without modifying the ordering of the other actions in the schedule. The function `split_route_cost` is the same as `split_route`, except it does not modify the schedule and returns only the change in cost. The function $\textit{ni}\,(S^v)$ returns the total number of items transported in the schedule $S^v$.

The function `best_transfer` searches through every feasible pair of edges on the routes of $a$ and $b$ and finds the transfer point of least cost which obeys the capacity constraints. We introduce a budgetary heuristic in the `best_transfer` function. If the start and ending points of item $m$, $R$ and $D$, in $S^a$, are both a greater distance than a budget $\textit{bud}\,(a)$ from any edge in $S^b$,

| $S^a$ | | $S^b$ | |
|---|---|---|---|
| Name | Action | Name | Action |
| $S_1^a$ | START | $S_1^b$ | START |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $\vdots$ | $\vdots$ | $R$ | PICKUP $(m)$ **or** RECEIVE$(m, c, l_1)$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $T_a$ | RECEIVE$(m, b, l_3)$ | $T_b$ | TRANSFER$(m, a, l_3)$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $D$ | DELIVER$(m)$ **or** TRANSFER$(m, d, l_2)$ | $\vdots$ | $\vdots$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $S_{\lvert S^a \rvert}^a$ | END | $S_{\lvert S^b \rvert}^b$ | END |

Table 5.2: Schedule After Route Splitting, $b$ Executes Action $R$

| $S^a$ | | $S^b$ | |
|---|---|---|---|
| Name | Action | Name | Action |
| $S_1^a$ | START | $S_1^b$ | START |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $R$ | PICKUP $(m)$ **or** RECEIVE$(m, c, l_1)$ | $\vdots$ | $\vdots$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $T_a$ | TRANSFER$(m, b, l_3)$ | $T_b$ | RECEIVE$(m, a, l_3)$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $\vdots$ | $\vdots$ | $D$ | DELIVER$(m)$ **or** TRANSFER$(m, d, l_2)$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $S_{\lvert S^a \rvert}^a$ | END | $S_{\lvert S^b \rvert}^b$ | END |

Table 5.3: Schedule After Route Splitting, $b$ Executes Action $D$

**Algorithm 5** `split_route_nt`$(S, m, a, b)$ splits an item $m$'s transport between vehicles $a$ and $b$ in schedule $S$.

$R \leftarrow$ PICKUP $(m)$ or RECEIVE$(m, c, l)$ in $S^a$
$D \leftarrow$ DELIVER$(m)$ or TRANSFER$(m, c, l)$ in $S^a$
**if** *ni* $\left(S^b\right) \geq$ *mni* $(b)$ **then**
  **return** $\infty$
**end if**
$loc \leftarrow$ `best_transfer`$(S, m, a, b)$
$w_{11} \leftarrow$ `route_insert_cost`$(S^a \setminus \{R, D\}, \langle R,$ TRANSFER$(m, b, loc)\rangle)$
$w_{12} \leftarrow$ `route_insert_cost`$(S^b, \langle$RECEIVE$(m, a, loc), D\rangle)$
$w_{21} \leftarrow$ `route_insert_cost`$(S^a \setminus \{R, D\}, \langle$RECEIVE$(m, b, loc), D\rangle)$
$w_{22} \leftarrow$ `route_insert_cost`$(S^b, \langle R,$ TRANSFER$(m, a, loc)\rangle)$
**if** $w_{11} + w_{12} = \infty$ and $w_{21} + w_{22} = \infty$ **then**
  **return** $\infty$
**end if**
**if** $w_{11} + w_{12} < w_{21} + w_{22}$ **then**
  $S^a \leftarrow$ `route_insert`$(S^a \setminus \{R, D\}, \langle R,$ TRANSFER$(m, b, loc)\rangle)$
  $S^b \leftarrow$ `route_insert`$(S^b, \langle$RECEIVE$(m, a, loc), D\rangle)$
  **return** $w_{11} + w_{12}$
**else**
  $S^a \leftarrow$ `route_insert`$(S^a \setminus \{R, D\}, \langle$RECEIVE$(m, b, loc), D\rangle)$
  $S^b \leftarrow$ `route_insert`$(S^b, \langle R,$ TRANSFER$(m, a, loc)\rangle)$
  **return** $w_{21} + w_{22}$
**end if**

then that potential transfer point is skipped over. With this heuristic, transfer points between vehicles that do not cross near each other are not considered. This helps alleviate the computational load of iterating through $O(|V|^2)$ potential transfer points.

Additionally, The `route_insert` function, when a transfer action is inserted, checks for cycles. Cycles are detected by following the edges of the transfer graph (introduced in Chapter 2) beginning at the new transfer point and checking if we return to the starting point.

### 5.2.3 Greedy Transfer Insertion

The first approach we propose greedily adds transfer points to an existing solution without transfer points. We form a queue $q$ containing items and the vehicles that are transporting them. For each item and vehicle in the queue, we iterate through the other vehicles and check if a different vehicle could take the item part of the way at a lower cost, using the `split_route` heuristic. If such a transfer exists, we find the other vehicle which would lower the cost the most, and apply the transfer to the schedule. We then add both halves of the split route back into the queue $q$,

where we may later add additional transfers to subdivide the route further. The full details of the greedy approach are presented in Algorithm 6.

**Algorithm 6** `greedy_transfer_nt`$(S)$: greedily add transfers to the schedule $S$.

```
q ← empty queue
for m ∈ M, v ∈ {v : m transported in S^v} do
  enqueue(q, (m, v))
end for
while q not empty do
  (m, a) ← dequeue(q)
  best_c ← ∞, best_v ← ∅
  for b ∈ V, b ≠ a do
    if in_budget(S^b, start(m)) or in_budget(S^b, end(m)) then
      c ← split_route_cost(S, m, a, b)
      if c < best_c then
        best_c ← c, best_v ← b
      end if
    end if
  end for
  if best_c ≠ ∞ then
    split_route(S, m, a, best_v)
    enqueue(q, (m, a)), enqueue(q, (m, b))
  end if
end while
```

The effectiveness of this algorithm, as with many greedy algorithms, is dependent on the order in which we iterate over the item and vehicle pairs. Furthermore, we only consider transfers between pairs of vehicles at a time, ignoring better routes which could be obtained by transferring between multiple vehicles. However, routes which transfer to multiple vehicles may be obtained when the greedy algorithm is applied recursively to partial routes of items on individual vehicles.

### 5.2.4 Transfer Insertion with Auctions

Our second approach is an auction, similar to the auction without transfers. In this auction, the items place bids for vehicles. In one round of the auction, each item finds the single transfer point which could be added to obtain the greatest cost decrease, by applying the `split_route` function. Each item places a bid for each vehicle. Then, each vehicle accepts the item with the lowest bid which will decrease the total cost the most. If no assignments were made the auction ends; otherwise we repeat the auction for another round. The full auction algorithm with transfers is shown in Algorithm 7.

**Algorithm 7** `auction_transfer_nt`$(S)$: add transfers to the schedule $S$ with an auction.

**for** $v \in V$ **do**
  $bid_v \leftarrow \infty$
**end for**
**for** $m \in M$ **do**
  $a, b \leftarrow \text{argmin}_{a \neq b \in V, m \text{ transported in } S^a}$ `split_route_cost`$(S, m, a, b)$
  $c \leftarrow$ `split_route_cost`$(S, m, a, b)$
  **if** $c \neq \infty$ and $bid_b > c$ **then**
    $bid_b \leftarrow c$, $bidp_b \leftarrow m$, $bidv_b \leftarrow a$)
  **end if**
**end for**
$done \leftarrow$ true
**for** $v \in V$ **do**
  **if** $bid_v \neq \infty$ **then**
    `split_route`$(S, bidp_v, bidv_v, v)$
    $done \leftarrow$ false
  **end if**
**end for**
**if** not $done$ **then**
  repeat auction
**end if**

Like the greedy approach, the auction algorithm has the shortcoming that it only considers a single transfer at a time. However, the auction approach is less greedy in the sense that the assignment does not depend on the ordering the items or vehicles are examined in.

### 5.2.5 Graph Search

Our final algorithm for solving the ridesharing with transfers problem differs from the first two in that it does not begin with a solution for ridesharing without transfers. Instead, it plans for transfers from the beginning. This allows the algorithm to find better solutions. However, because it searches for routes in an exponentially larger space which includes transfers, this algorithm is significantly slower to execute than the previous two approaches. It is still much faster than any optimal approach, but there is a tradeoff between solution quality and computation time.

The graph search algorithm is greedy in the sense that we iterate through each item, and plan the best path for that individual item. To do so, we construct a directed multi-graph representing all the vehicles' current schedules and potential transfer points. Each vehicle's initial schedule $S^v$ is set to contain only a STARTand ENDaction. The edges on the graph represent segments of robots' existing routes or transfers that an item could make on its route. The weights of the

edges represent the additional cost to the vehicles of using that means of transportation for the item. Then, the shortest path on the graph gives the route of least cost for the item. The actions found in that path are added to the schedule. The algorithm is outlined in Algorithm 8.

**Algorithm 8** `graph_schedule_nt`$(V, M)$: Form a schedule by searching on a graph.

$\forall v \in V\ S^v \leftarrow \langle$START, END$\rangle$
**for** $m \in M$ **do**
  $G \leftarrow$ graph constructed from $S$ for $m$
  $P \leftarrow$ shortest path in $G$ from PICKUP $(m)$ to DELIVER$(m)$
  $S \leftarrow S$ with PICKUP $(m)$, DELIVER$(m)$, TRANSFER and RECEIVE actions added from $P$
**end for**
**return** $S$

The graph $G$ constructed from $S$ to insert the item $m$ into the schedule contains the following nodes:

- $\forall v \in V, \forall a \in S^v$ $a$, every action in $S$;
- PICKUP $(m)$ and DELIVER$(m)$, the starting and ending actions on the item $m$'s route; and
- $\forall v_1 \neq v_2 \in V, 1 \leq i < |S^{v_1}|, 1 \leq j < |S^{v_2}|$ $T_{m,v_1,v_2,i,j}$. For each pair of edges on two robots' paths, we add a node $T_{m,v_1,v_2,i,j}$ representing the transfer point $\texttt{tpoint}(S_i^{v_1}, S_{i+1}^{v_1}, S_j^{v_2}, S_{j+1}^{v_2})$. Transfer points are not added if they would cause a cycle or if reaching them would exceed one of the two vehicles' budgets.

These nodes are connected by directed edges representing possible routes involving transfers taken by the item $m$, with the edge weights representing the change in the distance travelled if these routes were taken:

- $\forall v \in V, \forall 1 \leq i < |S^v|$ the edge from $S_i^v$ to $S_{i+1}^v$ with weight 0, representing the paths the vehicles already plan to travel (hence no additional cost);
- $\forall v \in V, \forall 1 \leq i < |S^v|$ the edge from PICKUP $(m)$ to $S_i^v$ with weight $d(S_i^v, \text{PICKUP}\,(m)) + d(\text{PICKUP}\,(m), S_{i+1}^v) - d(S_i^v, S_{i+1}^v)$ [1] representing the item being picked up on an edge of the original path;
- $\forall v \in V, \forall 1 \leq i < |S^v|$ the edge from DELIVER$(m)$ to $S_{i+1}^v$ with weight $d(S_i^v, \text{DELIVER}(m)) + d(\text{DELIVER}(m), S_{i+1}^v) - d(S_i^v, S_{i+1}^v)$, representing the item being dropped off on an edge of the original path;
- $\forall v \in V, \forall 1 \leq i < |S^v|$ an edge from PICKUP $(m)$ to DELIVER$(m)$ with weight $d(S_i^v, \text{PICKUP}\,(m)) + d(\text{PICKUP}\,(m), \text{DELIVER}(m)) + d(\text{DELIVER}(m), S_{i+1}^v) - d(S_i^v, S_{i+1}^v)$, representing the

[1] Note on notation: when actions $a$ and $b$ are passed to the distance function $d(a, b)$, which takes locations as arguments, this is equivalent to $d(\mathit{loc}(a), \mathit{loc}(b))$.

item being both picked up and dropped off on a single edge of the original path; and

- for each transfer point newly added as a node $X = \texttt{tpoint}(p^i_{v_1}, p^{i+1}_{v_1}, p^j_{v_2}, p^{j+1}_{v_2})$:
  - an edge from PICKUP $(m)$ to $X$ with weight $d(S^{v_1}_i, \text{PICKUP}\,(m))+d(\text{PICKUP}\,(m)\,, X)+ d(S^{v_2}_j, X) - d(S^{v_1}_i, S^{v_1}_{i+1}) - d(S^{v_2}_j, S^{v_2}_{j+1}) + c_t\,(m, v_1, v_2) + \epsilon$;
  - an edge from $S^{v_1}_i$ to $X$ with weight $d(S^{v_1}_i, X)+d(S^{v_2}_j, X)-d(S^{v_1}_i, S^{v_1}_{i+1})-d(S^{v_2}_j, S^{v_2}_{j+1})+ c_t\,(m, v_1, v_2) + \epsilon$;
  - an edge from $X$ to DELIVER$(m)$ with weight $d(X, \text{DELIVER}(m))+d(\text{DELIVER}(m), S^{v_2}_{j+1})+ d(X, S^{v_1}_{i+1})$;
  - an edge from $X$ to $S^{v_2}_{j+1}$ with weight $d(X, S^{v_2}_{j+1}) + d(X, S^{v_1}_{i+1})$; and, finally,
  - for every other additional node in $G$ that is a transfer point $Y = \texttt{tpoint}(S^{v_2}_j, S^{v_2}_{j+1}, S^{v_3}_k, S^{v_3}_{k+1})$, an additional edge from $X$ to $Y$ with weight $d(X, Y) + d(X, S^{v_1}_{i+1}) + d(S^{v_3}_k, Y) - d(S^{v_3}_k, S^{v_3}_{k+1})$. These edges allow a robot to receive an item and transfer it to another robot along the same edge in the original path, although it may not find the best transfer locations as if we were to construct additional transfer points recursively.

An edge is not added to the graph if that edge would cause a robot to exceed its capacity or maximum rider constraint.

Once a graph $G$ is constructed for an item $m$, we use the Bellman-Ford algorithm to find the shortest path on $G$ from PICKUP $(m)$ to DELIVER$(m)$. Dijkstra's algorithm cannot be used since the edge lengths may be negative. From the shortest path, we construct the route taken by the item, and modify the schedules $S^v$ of the appropriate vehicles to accommodate the items, insert the PICKUP $(m)$ and DELIVER$(m)$ actions as well as any necessary TRANSFER and RECEIVE actions into the appropriate places in the schedule. Then, we repeat the algorithm and find a route for the next item.

It should be noted that the resulting schedule found for an item may induce a cyclical dependency by passing through *multiple* transfer points. If this is the case, we remove the transfer points that caused cycles from the graph and try again.

This approach is significantly slower than the greedy and auction approaches, since the number of exchange points and the size of the graph increases quadratically with the number of edges in the vehicles' paths. However, there is substantial room for speed-ups if the graph is constructed incrementally. The cost in extra time provides a tradeoff with the higher solution quality the algorithm finds.

## 5.3 Selected Experimental Results

To verify the effectiveness of these three heuristics, we compared them in simulation and on a dataset of item routes in a city. We demonstrate that by considering transfers, we can solve many problems more efficiently compared to similar heuristics without transfers. We also show how the algorithms scale with more vehicles and items.

For each experiment, we set the number of vehicles and items $|V|$ and $|M|$. For every vehicle $v \in V$, the capacity $\mathit{cap}(v)$, the maximum number of riders $\mathit{mni}(v)$, and the budget $\mathit{bud}(v)$ is fixed. The starting and ending points for vehicles and items are selected randomly.

### 5.3.1 The Euclidean Plane

The first set of experiments were performed on a 10 by 10 unit Euclidean plane, where the vehicles travel in straight lines. The class of problems we examine is constrained to longer item routes of length at least 10 units, and medium-length vehicle routes of length 5 to 7 units. Problems with longer item routes than vehicle routes can benefit greatly from transfers. One example application of this class of problem is hitchhiking, where passengers travel long distances by catching rides with multiple drivers.

Figure 5.1 shows the costs of the solutions found by each of the algorithms in this scenario. The shaded regions denote the standard deviation across the fifty trials, and the "base cost" is what the cost in fuel would be if this were a ridesharing problem and all of the passengers and drivers drove themselves in their own vehicles directly to their destinations. In this particular domain, the greedy algorithm without transfers performs worse than the base cost, and the auction algorithm finds solutions of approximately equal cost to the base cost.

However, with transfers, we can outperform both the algorithms without transfers and the case when everyone drives themselves, reducing fuel usage. The greedy transfer algorithm, the auction transfer algorithm, and the graph-based algorithm each offer a successive improvement. With the graph-based algorithm, transfers reduce the distance travelled by nearly $30\%$ when there are 18 passengers.

Although the graph-based method finds the best solutions, its effectiveness compared to the auction and greedy approaches comes at a significant computational cost (see Figure 5.2. There is significant room for further optimization in each algorithm's implementation, particularly by constructing the graph used to find the route for each passenger incrementally based on the previous graphs.

The number of transfers proposed by each algorithm is shown in Figure 5.3. The number of transfers found can be lowered by increasing the cost of transfers, but this of course also reduces

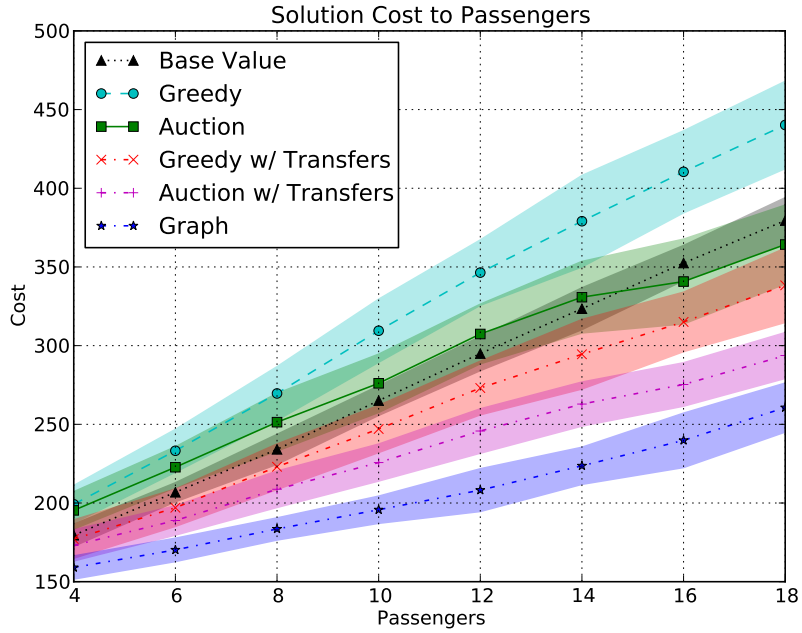



Figure 5.1: The solution cost for each method in the Euclidean domain with $|V| = 20$, $cap(v) = 5$, $mni(v) = 7$, $bud(v) = 6$, and $c_t(m, v_1, v_2) = 0$.

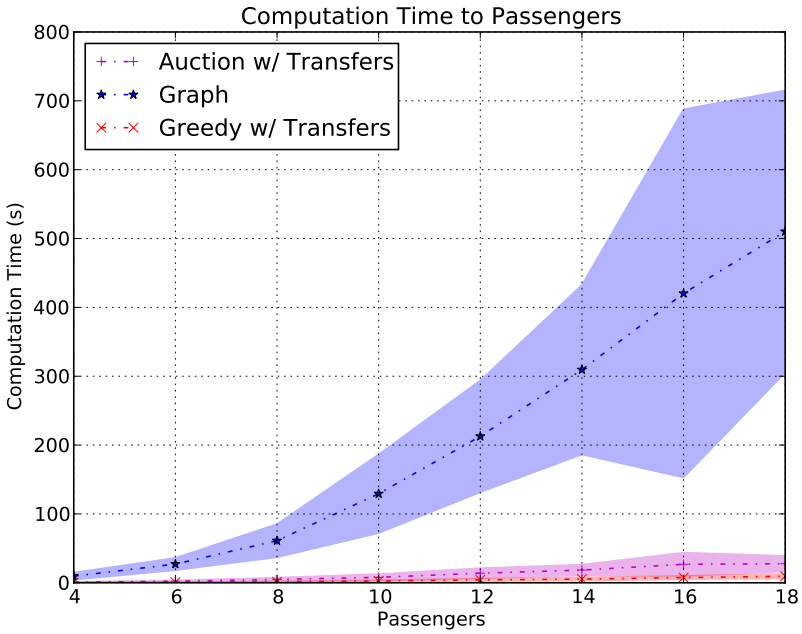



Figure 5.2: The computation time for each method in the Euclidean domain with $|V| = 20$, $cap(v) = 5$, $mni(v) = 7$, $bud(v) = 6$, and $c_t(m, v_1, v_2) = 0$.

the benefit gained from transfers.

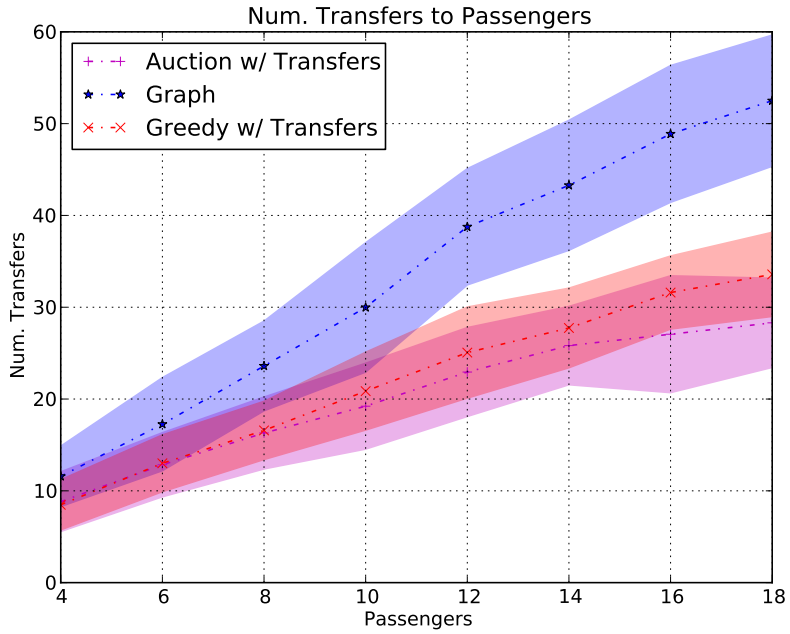



Figure 5.3: The number of transfers found for each method in the Euclidean domain with $|V| = 20$, $cap(v) = 5$, $mni(v) = 7$, $bud(v) = 6$, and $c_t(m, v_1, v_2) = 0$.

In Figure 5.4 we show the results of an experiment with the same parameters, except the number of vehicles changes rather than the number of passengers. When the number of vehicles varies, the approaches with transfers still significantly outperform the approaches without. The improvement, particularly with the graph algorithm, increases with the number of vehicles.

We also ran experiments where the length of the vehicle and passenger routes were unbiased, with starting points and destinations chosen purely at random. In these cases, transfers often reduced the total distance travelled as well. However, the amount of the improvement depended largely on the individual problem instances.

### 5.3.2 San Francisco

In addition to tests in the plane, we solved problems based on real world ridesharing problems. In these problems, non-professional drivers with their own destination pick up passengers along the way and transport them to their own destinations. By allowing transfers, we plan for the passengers to potentially take multiple vehicles to reach their destinations, allowing drivers to help transport passengers while going less out of their way to do so.

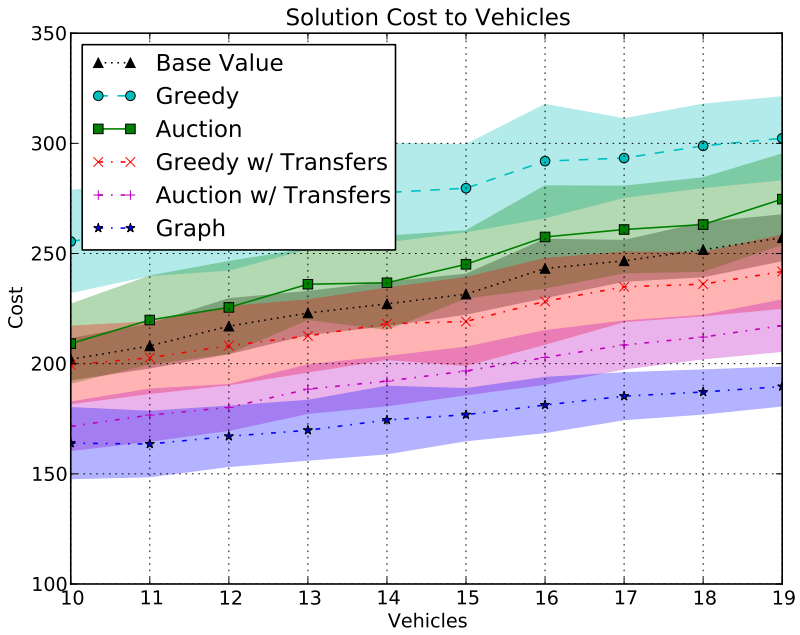



Figure 5.4: The solution cost found for each method in the Euclidean domain with $|M| = 10$, $cap(v) = 5$, $mni(v) = 7$, $bud(v) = 6$, and $c_t(m, v_1, v_2) = 0$.

To test these ridesharing problems, we obtained a map of San Francisco from OpenStreetMap [84] and found shortest paths on this map using pgRouting [111] (see Figure 5.5 for an example solution).

We sampled passenger and vehicle starting points from real-world taxi cab data [89], limited to the downtown area. Vehicles are constrained to trips between $0.22$ and $0.56$ km, and passengers are constrained to trips greater than 1.33 km. Selected results are shown in Figure 5.6.

Routing on the map of San Francisco with pgRouting is significantly more costly than on the Euclidean plane. For the case with seven passengers, the greedy algorithm with transfers took over 80 seconds and the auction algorithm with transfers took nearly 200 seconds. The graph algorithm was not run due to its' high cost. Faster path planning algorithms, or perhaps approximate distance estimates, will need to be used to make ridesharing with transfers a reality. However, there are still average savings of approximately $15\%$ found by using transfers.

## 5.4 Chapter Summary

We have presented three heuristic algorithms to solve the nTPDP-T: a greedy approach, an auction approach, and an approach based on graph search. Each of the algorithms trades off compu-

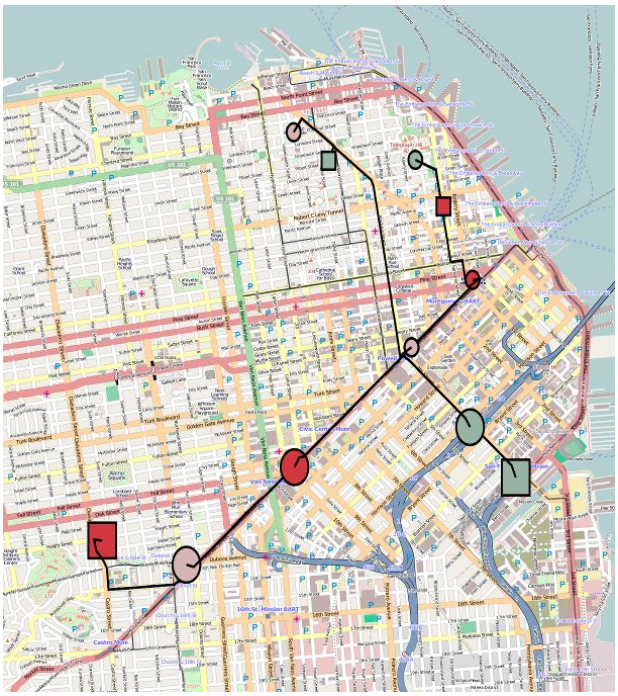


Figure 5.5: An example solution with transfers for a problem in San Francisco. Two passengers are transferred to other vehicles.

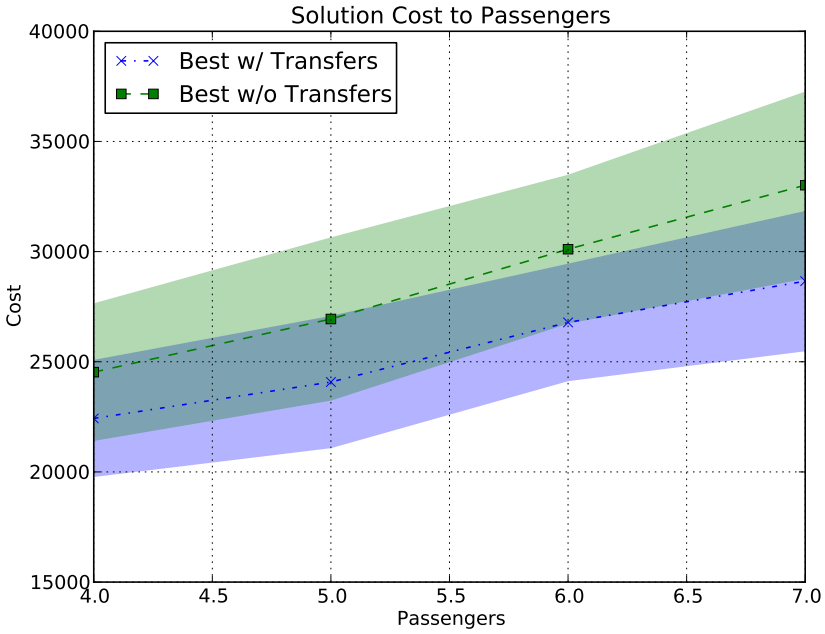



Figure 5.6: The solution cost found for each method in San Francisco with $|V| = 18$, $C_v = 4$, $M_v = 7$, $B_v = 1.5$ km, and $c_T = 0$.

tation time with solution effectiveness. We have demonstrated that in certain problem instances, transferring items can reduce the distance travelled by nearly 30%, including on ridesharing problems on a map of San Francisco. In the remaining chapters of this thesis, we build upon the ideas in these heuristics to find improved solutions for the general PDP-T, both offline and online.

# Chapter 6

# Online Rescheduling with Transfers

In Chapters 4 and 5, we presented algorithms to plan for variants of the PDP-T without time windows, when all the item requests are known beforehand. However, in many domains, the requests may not be known beforehand, but may come online. Item requests may also be modified or cancelled during schedule execution, and vehicles may fail or be delayed. In all of these cases the vehicles must **replan** and modify the schedule. That is the subject of this chapter.

In Chapter 3, we presented a preliminary approach to scheduling for PDP-T problems on the CoBots. This scheduler found an optimal schedule by solving a mixed integer program (MIP). Users made requests online, and a new MIP was solved every time a request arrived. This is effective for small numbers of tasks, on the order of two vehicles and fifteen tasks without transfers. However, solving an MIP is infeasible when there are large numbers of tasks and vehicles, and even more challenging with transfers. To solve the MIP from scratch quickly in response to online requests is even less reasonable.

Hence, we require an algorithm to modify an existing schedule, quickly and online, in response to new and changing requests. We present heuristics based on ideas similar to the heuristics presented in Chapter 5, which additionally consider time windows and are able to modify existing schedules. In this chapter we consider most of the constraints introduced in Chapter 2, with the exceptions of the maximum route duration and maximum transport time. In addition, we allow for *soft* time constraints in this chapter only.

In the second half of this chapter, we consider that the robots may not replan and reschedule based only on new requests and their own state, but also based on the *shared information* from other vehicles. For example, one of the CoBots could observe that a door is closed at an office another robot is making a delivery in, and the other robot could delay its scheduled delivery until the occupant returns. Or one CoBot may observe that a hallway is blocked, and warn the other CoBots to avoid it. We enable the CoBots to share these types of information and reschedule

based on the observations of other robots.

## 6.1 Auctions to Revise Schedules with Transfers Online

We introduce an auction mechanism to plan online, time-constrained PDP-T schedules. One advantage of an auction mechanism is that it could also be implemented in a distributed manner. We replan in response to new requests, and to vehicle delays and failures. We evaluate our approach experimentally on the CoBots and on large simulated problem instances.

### 6.1.1 The Online Auction Algorithm

At a high level, our rescheduling approach is to revise a schedule through an auction. Individual items are put up for auction. Initially, vehicles place bids based on the additional cost they would incur to pick up and deliver an item. This cost is determined by an insertion heuristic which inserts the item pickup and delivery actions into the vehicle's schedule. The vehicle with the lowest bid wins, and the item is added to that vehicle's schedule. Then, that vehicle holds a second auction, in which other vehicles can place bids to transfer the item to or from the original vehicle and complete part of the transport at lower cost. If a transfer is added to the schedule, recursive auctions may be held to further divide the item's transport.

In these auctions, time constraints are maintained with a Simple Temporal Network (STN). All scheduling is done by a centralized algorithm, which is aware of all requested tasks and the vehicles' current positions. The server already knows all of this information for the telepresence interface, so no additional communication is needed [29]. However, this auction algorithm could also be implemented in a distributed manner. The vehicle holding an auction would need to communicate its full schedule to the other vehicles, which could then use this information to find the best place to insert a transfer when placing bids.

We provide a top-down explanation, first discussing the high level auction, then the insertion heuristic, and finally how time constraints are maintained.

#### Auctioning Pickups, Deliveries and Transfers

Algorithm 9 shows the auction algorithm for scheduling transfers with time constraints.

When the auction algorithm is first called, we begin with an existing partial schedule. This partial schedule may include delivering other items which were scheduled earlier from online scheduling. Even if no items are delivered, the partial plans alway includes *Start* and *End* actions.

**Algorithm 9** `auction`$(S, V, M)$: Run an auction to form a schedule for the vehicles $V$ to deliver items $M$. The vehicles begin with the partial schedule $S$ (which may consist of solely *Start* and *End* actions).

1: $\forall v \in V$ $bids_v \leftarrow \infty$
2: **for** $m \in M$ **do**
3: **if** $m$ not in schedule $S$ **then**
4: $\forall v \in V$ `bid`$(m, v,$ `route_insert_cost`$(S, v, \langle \text{PICKUP}(m), \text{DELIVER}(m) \rangle))$
5: **else**
6: $\forall v_1, v_2 \in V$ s.t. $v_1$ transports $m$ in $S$, $v_2 \neq v_1$ `transfer_bid`(
7: $m, v_1, v_2,$ `insert_transfer`$(m, v_1, v_2)$)
8: **end if**
9: **end for**
10: $done \leftarrow$ True
11: **for** $v \in V$ **do**
12: **if** $v$ has a valid bid **then**
13: Vehicle $v$ wins bid of lowest cost, update the schedule
14: Cancel conflicting bids
15: $done \leftarrow$ False
16: **end if**
17: **end for**
18: **if** not $done$ **then**
19: Repeat auction
20: **end if**

First, we check if each item is delivered in the existing partial schedule (line 2). If not, each vehicle places a bid to pick up and deliver that item by inserting a *Retrieve* and a *Deliver* action into its plan without changing the rest of the plan's ordering (line 5). The value of the bid is the additional cost incurred by the vehicle, including both additional distance travelled and the penalty for soft time window violations. If the item already is part of some vehicle's plan, then for each such vehicle, we attempt to insert transfer actions to split its route with another vehicle (line 7). The cost is again the additional cost in distance and time window violations incurred by all of the vehicles. The `route_insert_cost` procedure was already introduced in the previous chapter. We explain the `insert_transfer` procedure in detail in the next section.

Each vehicle places a bid on each item and item transport split. Once all the bids are placed, the winning bids are evaluated (lines 11 - 17). Each vehicle's winning bid is applied, and the schedule is updated accordingly to insert either a new item or a new transfer. Winning bids may conflict with later bids, for example, if two vehicles bid on the same item, or if two vehicles bid to transfer an item to or from the same vehicle. Due to time constraints, more subtle conflicts may occur if a new introduced transfer causes an action on an entirely different vehicle to be

delayed. We detect these conflicts using temporal networks as discussed later.

To further optimize the auction algorithm, we use caching when possible so we do not need to reevaluate bids if the relevant section of the schedule has not changed.

### Insertion Heuristic

The `insert_transfer`$(m, v_1, v_2)$ subroutine inserts a transfer of an item $m$ transported by vehicle $v_1$ to or from an additional vehicle $v_2$. Intuitively, this routine splits vehicle $v_1$'s transport of $m$ in half with vehicle $v_2$: vehicle $v_2$ takes responsibility for either $m$'s retrieval or delivery (either a RETRIEVE or DELIVER action, or a RECEIVE or TRANSFER action) and the item is exchanged midway. This is a similar idea to the `split_route_nt` algorithm from the previous chapter. However, instead of choosing a transfer point up front and then inserting that transfer point into the schedule, we consider different transfer points as we consider different insertion points in the schedule. See Figure 6.1 for an example of the expected output of the `insert_transfer` algorithm.

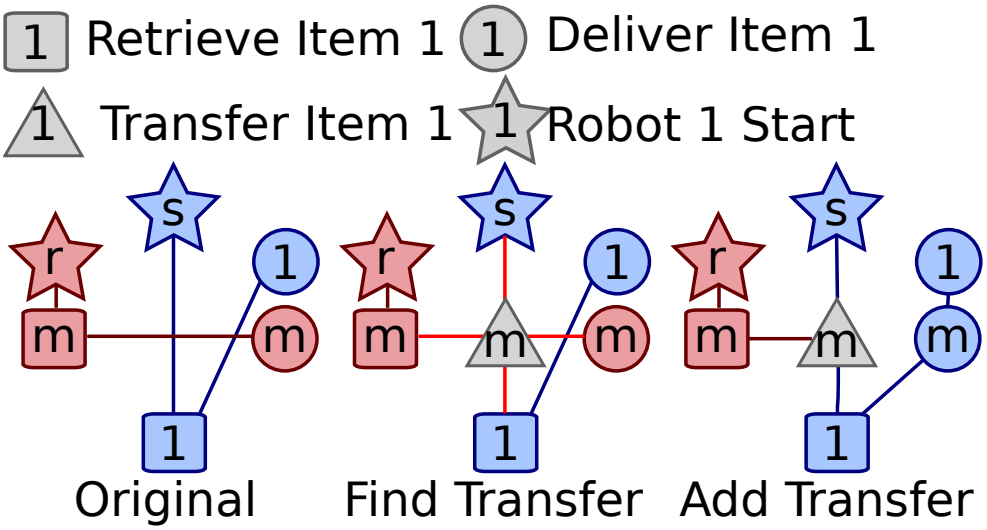



Figure 6.1: Vehicle $r$'s pickup and delivery of item $m$ is split with vehicle $s$ using `insert_transfer`. First, a transfer point is chosen between two subsequent tasks on each vehicle's plan. Then the delivery point is removed from $r$'s plan and inserted into $s$'s, lowering the delivery cost.

First, the `insert_transfer` algorithm searches over all subsequent pairs of actions $a$, $b$ in $S^{v_1}$, and subsequent actions $c$ and $d$ in $S^{v_2}$. A RECEIVE action will be inserted between actions $a$ and $b$, and a TRANSFER action will be inserted between actions $c$ and $d$ if vehicle $v_2$ will make the delivery, or vice versa if vehicle $v_2$ will pick up item $m$ originally in place of vehicle $v_1$. The inserted transfer point must not violate the capacity and must be reachable without violating any

hard time constraints. The algorithm computes a proposed exchange point from *loc*$(a)$, *loc*$(b)$, *loc*$(c)$, and *loc*$(d)$. For CoBot's map, we simply find the first intersection point between the shortest path from *loc*$(a)$ to *loc*$(b)$ and the shortest path from *loc*$(c)$ to *loc*$(d)$. If no such point exists then no transfer is made between these actions. More sophisticated methods of choosing transfer points can be used for other maps.

Once a transfer point is found, the algorithm attempts to have vehicle $v_2$ pick up the item in place of vehicle $v_1$, then transfer it to $v_1$ for $v_1$'s original delivery. Next, it attempts to have $v_1$ pick up the item in the original location, then transfer it to $v_2$, and have $v_2$ deliver the item to $v_1$'s original delivery location. We iterate through every possible insertion point for the new pickup or delivery point in $S^{v_2}$, and choose the plan of lowest cost.

During this search, we check that the newly introduced transfer does not induce a cycle by following the transfer graph beginning at the transfer actions and checking that neither of the initial transfer actions is reached a second time.

Although the `insert_transfer` routine runs in polynomial time, it is still expensive for large problem instances. To speed things up and reduce the number of considered transfer points, we add a budget *bud*$(v)$ for each vehicle as in the previous chapter. If the starting and ending points of item $m$'s portion of $v$'s route are both further than *bud*$(v)$ units of distance from $s$'s planned path, we disregard the potential transfer point. This limits the consideration of transfer points that are unlikely to be cost-effective.

### Maintaining Time Constraints

To maintain time constraints, we form a Simple Temporal Network [35]. Every action in the vehicles' plans is a node in the network, associated with a time window within which that action must occur. Each edge is associated with a time window which bounds the difference in time between two nodes. Every *Start* action node has the time window $[0, 0]$, and every *End* action has the time window $[0, \infty)$. Nodes for actions that transport item $m$ have the time window $[m_S, m_E]$.

Every pair of subsequent actions $a$ and $b$ in a vehicle $v$'s plan have nodes linked with an edge with duration $[\delta(a) + d_v(\textit{loc}(a), \textit{loc}(b)), \infty)$, the minimum time in which a vehicle can complete action $a$ and then travel to the location of action $b$. TRANSFER actions are connected to the corresponding RECEIVE actions with edges of duration $[0, 0]$, ensuring that both actions occur at the same time. Whenever the schedule is modified, we solve the constraints in the temporal network to find valid windows of time in which each action could be executed without violating any constraints. Figure 6.2 shows an example temporal network and solution.

With hard time constraints, when a new action or set of actions is inserted into the schedule,

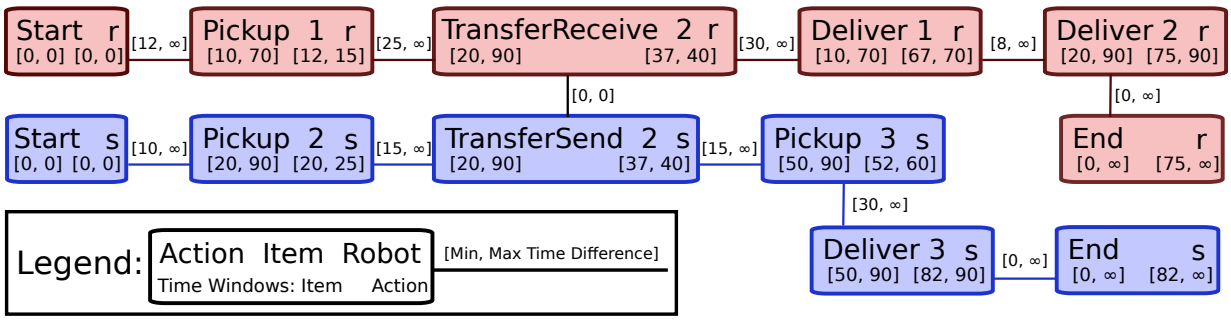



Figure 6.2: An example temporal network with two vehicles, three items and a single transfer. The feasible time windows for each action are computed based on the item time windows and the edge durations.

we attempt to insert the new actions into the temporal network to determine whether or not the schedule remains feasible with the new actions. This does not require reconstruction or recomputation of the entire temporal network from scratch; the changes can be propagated from the insertion points. The starts of windows are updated from left to right, and the ends of windows are updated from right to left, starting with the newly inserted action. With soft time constraints, the temporal network is used to compute the earliest feasible execution time of each action by setting all delivery deadlines to infinity. The action execution times are then used to determine the cost of violated soft time constraints.

### 6.1.2 Online Scheduling and Rescheduling

We presented an algorithm to revise a schedule to transport new items. To replan online with this scheduler, the existing schedule must first be updated. First, completed tasks are removed from the schedule. Then, vehicles currently carrying items have PICKUP actions added to their schedule at the current location, with a time window of $[a, a]$ where $a$ is the current time and a duration of $0$. These actions are not executed, but ensure the algorithm maintains its invariants.

We replan in four cases:

- **New Item Requested**: The auction algorithm inserts the new items into the existing schedule.
- **Request Cancelled**: Every action involving the removed item is removed from the schedule.
- **Delayed Vehicle**: If a vehicle is late to complete a task by a fixed amount of time, all tasks involving items the late vehicle is scheduled to carry and does not currently hold are removed from the schedule and re-inserted.

- **Disabled Vehicle**: If a vehicle does not communicate with the server for a fixed time, it is marked as disabled. Its tasks are re-added to the schedule, except for items it is holding. It will be impossible to deliver these items without manual intervention. This approach may lead to conflicts with multiple vehicles completing tasks if the vehicle is operational but communication has failed. However, it is unlikely for a vehicle to remain incommunicado for an extended time, and executing a task twice is preferrable to not executing the task at all.

### 6.1.3 Experiments in Online Rescheduling

We first present several illustrative problems run on the CoBot robots, demonstrating the scheduler's ability to revise schedules with transfers. We then share extensive results from larger problem instances solved in simulation, demonstrating the scheduler's scalability and effectiveness.

#### Illustrative Revised Schedules on CoBot

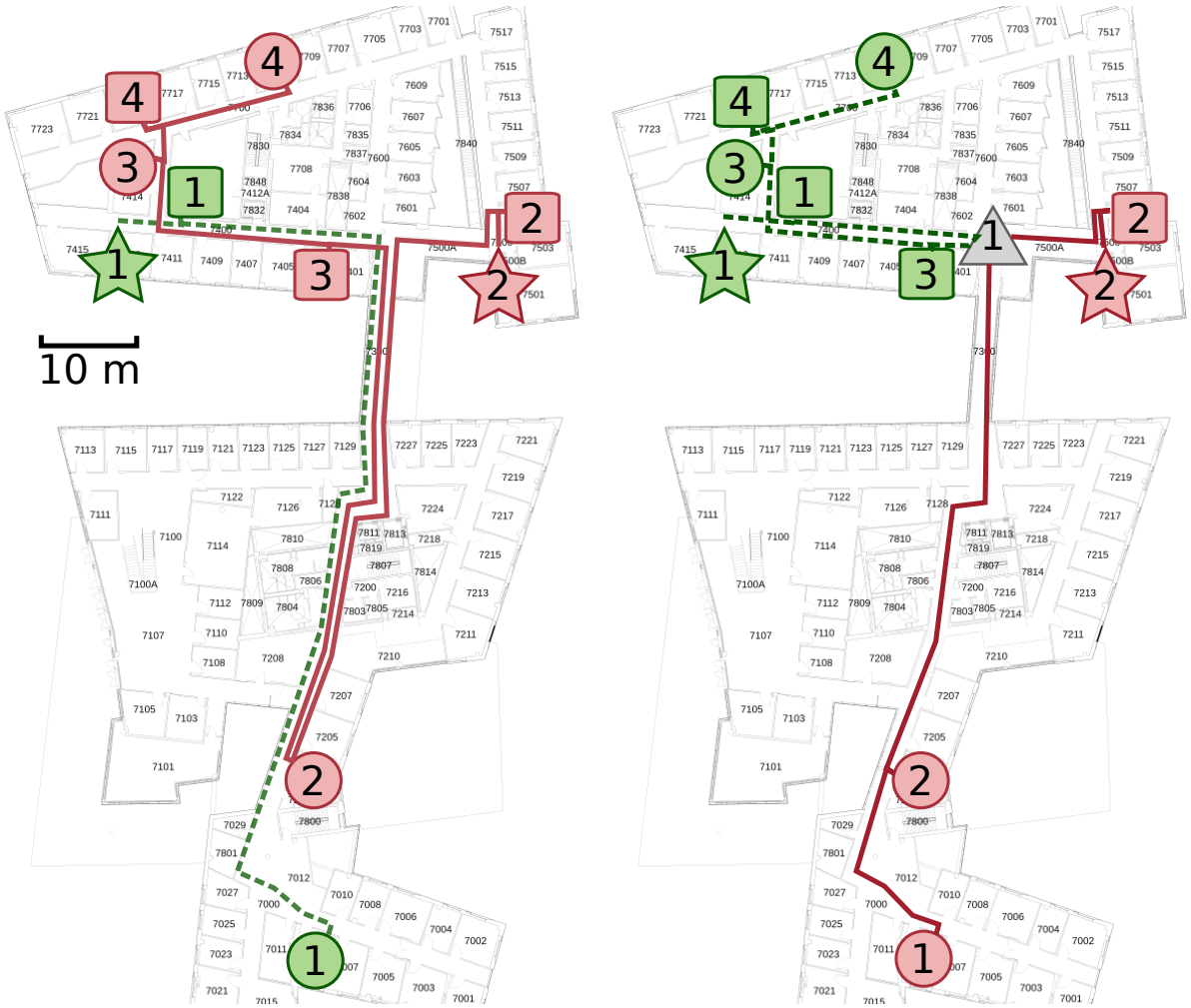
Since the CoBots do not have arms, pickups, deliveries and transfers are made with human help. We assume in these examples that humans are readily available to retrieve, deliver and transfer items. The time taken to ask for help is included in the results.

For the first experiment, two robots placed four pickup and delivery requests (see Figure 6.3). Items 1 and 2 were requested at time 0, Item 3 at 150 s, and Item 4 at 200 s. Solving the MIP optimally without transfers in an online manner (our original approach) results in each robot performing one of the initial tasks. When Tasks 3 and 4 arrive, one robot travels all the way back to the starting point. Using the auction algorithm with transfers, only one robot travels to the opposite end of the building initially. Then the other robot is free to deliver items 3 and 4. The MIP approach took approximately 5 minutes 45 seconds to execute and traversed 280.7 m. The auction algorithm with transfers took approximately 4 minutes to execute and traversed 162.1 m.

For the second example, we used three robots to perform five tasks placed at the same time. Two robots were scheduled to transfer an item to a third robot for delivery. However, we immediately turned off the robot scheduled to deliver the three items. The server detected this, and a new plan was formed which the robots then executed (see Figure 6.4).

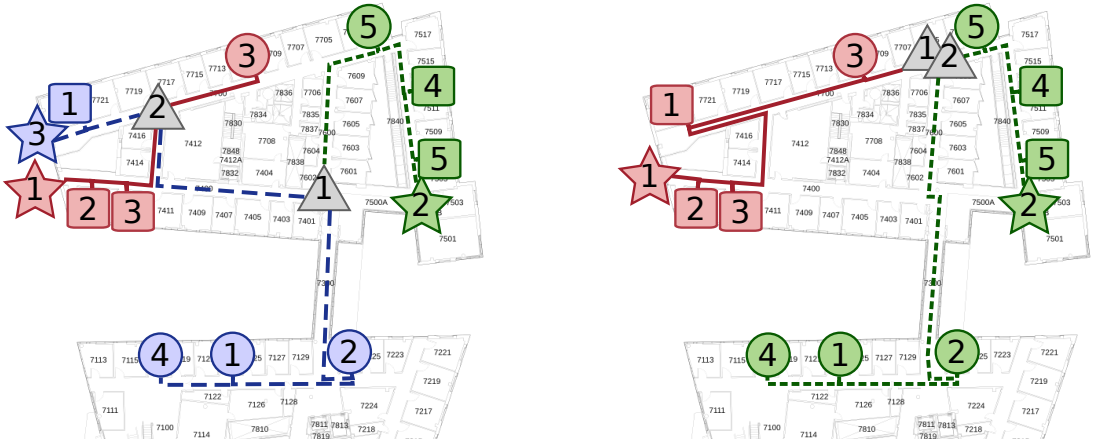
#### Large Simulated Problems

The final problem set was run on large simulated problem instances to test the scalability of the algorithms. The world is a 30x30 grid of city blocks, each one unit long. Item pickup and




(a) (b)

Figure 6.3: Planned schedules to deliver four items, scheduled with the *(a)* MIP without transfers, and *(b)* auction with transfers. Items 1 and 2 are requested at time 0, Item 3 is requested after 150 s, and Item 4 is requested after 200 s. See Fig. 6.1 for symbol meanings.



(a) (b)

Figure 6.4: *(a)* Deliveries are scheduled with three robots (including two transfers). *(b)* When one of the robots dies and fails to respond, the tasks are rescheduled. See Fig. 6.1 for symbol meanings.

dropoff locations are chosen from the block intersection at random. Unlike in the CoBot domain, vehicles have an assigned end location, a station where they return to charge. Corresponding start and end points for both vehicles and items are constrained to be at least five blocks apart.

We ran tests in this domain with $|V| = 80$ vehicles and with the number of items $|M|$ varying from 20 to 240. We formed schedules for fifty different trials for each value of $|M|$. For every vehicle $v$, $\mathit{cap}(v) = 3$ and $\mathit{bud}(v) = 5$. Each vehicle travels one block per minute. Soft time windows are used, and the cost function penalty for delays was $50$ per minute, meaning every minute a delivery is late adds 50 units to the cost function. The remainder of the cost function is based solely on total distance and the number of transfers, with $\alpha = 1$, $\beta = 0$, $\gamma = \infty$ (with soft time windows, every request can always be scheduled), and $c_t(m, v_1, v_2) = 4$. Each item is given a small time window of ten minutes for delivery. This is a very tight window: for some items it is not even physically possible to deliver the item in time. Our goal here is to make the delivery as quickly as possible. The start of this time window falls at a random point in the interval $[0, \frac{5}{4}|M|]$. The schedule is executed online, with new items scheduled one at a time. The scheduler is informed of each request half an hour before the time window begins (or immediately if the start of the window falls within the first half hour).

These problems are too large for us to solve optimally. Instead, we compare the auction algorithm with transfers to the auction algorithm without transfers. The costs of the solutions found with both algorithms are shown in Figure 6.5. With 240 items, the cost is reduced by almost 33% by using transfers. Most of this savings comes from an improved capability to

deliver items within their time windows due to the availability of additional scheduling options. With 240 items, an average of 26.6 transfers were made. The auction without transfers took an average of 6.03 s to execute in total, while the auction with transfers took an average of 13.98 s. This amounts to an average of 0.058 s per request.

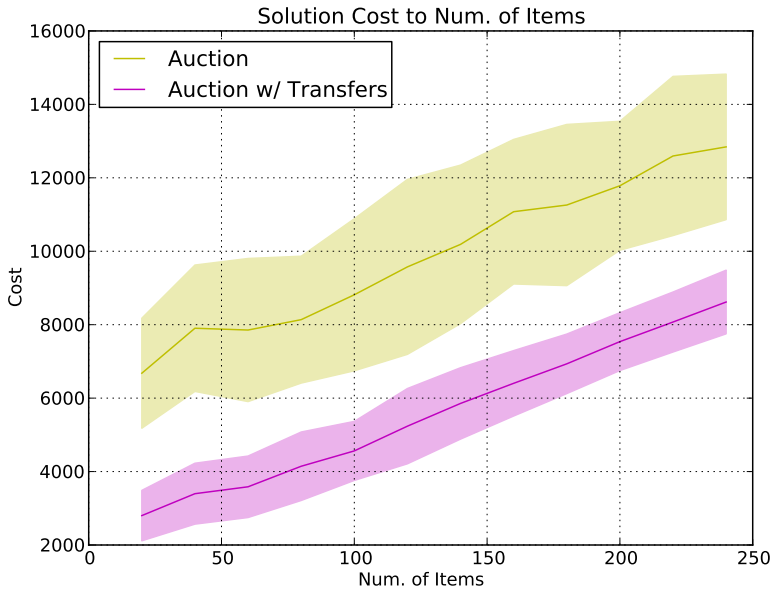



Figure 6.5: The mean cost of the generated schedules to the number of items. The shaded regions depict the standard deviation across the fifty trials.

## 6.2 Rescheduling with Shared Information

Aside from forming multi-vehicle schedules which may include transfers, we have thus far taken little advantage of the fact that there are multiple vehicles deployed to execute schedules. Once assigned a task, a vehicle executes that task independently without aid or communication from any other vehicle. In this section, we explore how multi-agent techniques can be applied to the CoBots to perform their existing tasks more effectively.

In the previous section, we enabled the vehicles to reschedule when task requests are made or modified, and when the vehicles are delayed or fail. In this section, we look at novel ways to improve task execution through distributed information sharing. We consider this problem specifically from the perspective of the CoBot robots, which can detect when doors are closed and when hallways are blocked, and then share this information to form better schedules more likely to succeed on time. We look at constructing a "rationale graph" detailing the dependen-

cies between these assumptions and the robots' schedules. We then use this rationale graph to communicate which assumptions have been violated and to replan for the affected tasks on the rationale graph. Replanning can be done in either a centralized or decentralized fashion.

### 6.2.1 Multi-Agent Rescheduling with Rationale Graphs

One of the main factors preventing service robots from being deployed in the real world is a lack of robustness to a dynamic world with unexpected events. This robustness can be increased through the use of multiple agents. Specifically, we propose to increase robustness among the CoBots by sharing information and using this information to replan. The information we envision the CoBots sharing includes:

- *Hallways that are blocked or have heavy traffic*. If one robot cannot navigate through a hallway, other robots should avoid that corridor until traffic decreases.
- *Closed and open doors*. Robots have some flexibility regarding when they deliver a message or item to an office. If one robot passes an office and sees it is currently closed, other robots should delay making trips to that office, if their schedule constraints allow it.
- *Robot failures*. If a robot is unable to move or the other robots are unable to communicate with it for a lengthy period of time, it will not be able to complete its tasks. The other robots should finish the tasks in its stead.
- *Human help availability*. CoBots rely on humans to help them take the elevator. If one robot sees humans waiting at the elevator, it could communicate this information to let the other robots know that it may be a good time to take the elevator. Also, communicating open door information can help other robots proactively seek help in occupied offices.

This shared information serves as preconditions in each robot's schedule. To complete a delivery task, the hallways the robot travels in must not be blocked, the door at the delivery point must be open, the robot must be in working order, and human help must be available when necessary. Each piece of information is time-sensitive, and the robots' certainty decays over time. Previously, we assumed that all these prerequisites would always be satisfied. If certain conditions were not satisfied, such as a hallway being blocked, a robot would be delayed and rescheduling would occur. Or, if no one was at an office to give the robot an item, the task would be reported as failed.

We assume that the multi-agent scheduler outputs these prerequisites for actions in the schedule, or *rationales* for the schedule along with the schedule itself. For example, a schedule for robot $r_1$ to deliver two items to Office A and Office B could be:

1. DELIVER$(A)$ because *open*(*end*$(A)) \wedge$ *unblocked*(Hall1)

2. DELIVER($B$) because *open*(*end*($B$)) $\wedge$ *unblocked*(Hall2)

If an assumption behind the schedule's reasoning changed, such as another robot observing Hall1 is blocked, then rescheduling would become necessary. The scheduler could adjust the schedule so that the robot went down a different hallway, or so that a different robot which could take a different path performed the task instead.

The largest difficulty is that the scheduler must provide reasoning along with a schedule. Sometimes the reasoning behind a schedule is not entirely clear. Negative reasoning— or rationales for why the robot *didn't* choose to execute a different schedule— is particularly difficult to come up with: if a robot chose to take one corridor first, there could be any number of other corridors it could have chosen had they been less crowded, or any number of other tasks that could have been completed earlier had one corridor had a different state or one door been open earlier. For example, one corridor could have been blocked, leading us to take a longer path. If that corridor becomes unblocked, we would then ideally want to reschedule to use that corridor.

However, positive rationales— reasons that the chosen schedule are feasible— are very easy to discover. In the earlier example, office A was chosen first because it was assumed to be open, and Hall1 was selected as part of the robot's path because it was unblocked. If these rationale are violated then the schedule is highly likely to change. If another path becomes open or a door opens, then a better schedule could become available, but the current schedule is still valid. So we focus on positive rationales. If negative rationales are returned by the scheduler, they can still be shared and exploited as well. It may be possible to ennumerate some negative rationales— for example, the list of blocked hallways is likely small— but considering all of the negative rationales, including every office door that was closed, or the scheduling of all the other tasks, would presumably be expensive.

Reasoning could be used to reschedule for even greater effect if a *rationale graph* can be formed. Here, each choice predicate (or a subset of the predicates) is linked to the specific decisions in the schedule which came about as a result of that predicate. Each of these decision points / scheduling assignments links to another choice. If this graph is known, then when a predicate changes, only the parts of the schedule that are its dependences in the rationale graph need to be updated. A rationale graph will likely be incomplete, with only positive rationales or with only selected negative rationales, since the set of negative rationales may be large. When a rationale is violated (i.e., a hallway is blocked), the tasks linked to it are rescheduled in light of the violated assumption.

Two major multi-agent technical challenges must be solved to accomplish this behavior. First, the robots must have a method of sharing prerequisites in the rationale graph and deciding which information to share with one another. Second, the robots must have an algorithm to adjust a

multi-agent schedule and decide when to reschedule.

### 6.2.2 Multi-Agent Rationale Sharing

The first challenge to rescheduling with shared information is choosing how to share rationales and selecting which rationales to share. We present two potential mechanisms for distributed information sharing.

#### Distributed World Model

In the first method, a distributed world model, all robots share complete information of the world. This is a common technique in multi-agent robot soccer [30, 62]. At regular intervals, each robot shares its full state with all other robots: it's position and orientation, its current schedule, and the observed state of the world, including open and closed doors and blocked hallways. The robots merge these observations to store the current state of each robot, the state of each door and the last time it was observed, and the state of each hallway and the last time it was observed. Hallway observations could consist of the last time a hallway was traversed and the time it took, or a "blocked" status if a robot is currently unable to progress in a hallway.

Forming a distributed world model makes it easy for each robot to have all necessary information to detect violated rationales in the rationale graph, and to schedule and to reschedule. It is also easy to detect when a robot has failed from a lack of communication. For a relatively small deployment a distributed world model may be the best option. However, it does not scale well to large numbers of robots, where each robot needs to share its entire state with $n$ other robots. Furthermore it requires significant bandwidth use, which is already limited by other applications such as telepresence which require a large part of the available wireless bandwidth.

#### Distributed Sharing of Invalidated Rationales

In the second method, the entire rationale graph is shared with all of the robots. When a robot detects an invalidated rationale, it shares the information only with the robots that are assigned tasks directly linked to that prerequisite in the rationale graph.

This approach requires little bandwidth compared to the frequent communication updates needed to form a distributed world model. Furthermore, it is less susceptible to network delays and failures, which are common with mobile robots required to switch between multiple wireless access points. The scheduler could be centralized, like our current MIP approach, or tasks could be assigned in a decentralized, online fashion.

However, a separate method is required to determine when a robot fails. This could be accomplished by scheduling regular communicated updates with specific robots. If an update does not occur, the robot can be assumed to have failed.

### 6.2.3 Multi-Agent Rescheduling

The remaining technical difficulty is how the robotic agents can reschedule given shared information, either from a distributed world model or from more limited reason-based information sharing. We discuss two potential methods of rescheduling: centralized rescheduling, and decentralized rescheduling, particularly by auctioning tasks to robots.

#### Centralized Multi-Agent Rescheduling

Currently a schedule is formed on a centralized server and then sent to the robots to execute. This setup makes sense since there is already a centralized server to accept user tasks on the website. The simplest way to reschedule is, when one of the original schedule's rationales is cancelled, to form a new schedule entirely from scratch. However, this is computationally expensive and is often unnecessary. Furthermore, it may sow confusion among users if the users were given a specific time a robot is expected to come: for this reason, we only tell users that a CoBot will arrive in their requested time window.

Instead of scheduling entirely from scratch, we can can make use of the rationale graph to partially reschedule only the tasks on the schedule whose rationales are broken. This can be done with a local search procedure, such as Tabu search or simulated annealing, searching for small changes to the affected area to find a new schedule. The rest of the schedule that is not affected by the rationale graph is treated as a base and remains unchanged.

If a full rationale graph is not available from the scheduler, then local search can be performed on the entire schedule with an additional weight towards changing the links causing direct conflicts.

When rescheduling, care must be taken so that the schedule includes the current state of the robot: the tasks that robots are already in the progress of completing, including pickup tasks the robots have already completed and items they are currently carrying.

#### Multi-Agent Rescheduling with Decentralized Auctions

Scheduling and rescheduling of multi-agent schedules can also be done in a distributed, online manner using auctions, such as with our algorithm earlier in this chapter. When one agent receives a request (it could be either the web server or a user communicating with a robot directly)

that robot broadcasts the request to the others to declare an auction. Each robot then checks if the new request could be fit into its schedule. If so, each robot places a bid with the cost in fuel and time of adding the request into its schedule. The bid with the lowest cost wins, and the auctioneer announces the winner to all robots.

Then, the winner communicates the positive rationales for its bid to the other robots. Each robot stores all the other robots assigned tasks along with their rationales. If a rationale is violated, the relevant robot is informed. That robot then attempts to adjust its own schedule using local search to still complete the task. For some violated rationales no rescheduling is necessary, for example, if a hallway is blocked and the path scheduler can select a different route. If the task cannot be completed, or the cost is significantly higher, the robot declares a new auction for the same task, to see if another robot can perform it more efficiently. Further auction rounds can be held to plan for transfers.

The rationale graph in this setup does not include negative rationales. However, negative rationales may be included at an additional communication cost. When placing bids, robots can share all the rationales for their bids, both positive and negative. If a negative rationale changes, and the robot that requested the task can now perform it at lower cost, it can communicate with the bid winner to initiate a new auction.

While sharing either positive rationales or all rationales, edges between tasks assignments on the rationale graph do not need to be shared. This is because they can be represented implicitly with the auction mechanism. When a rationale is violated and a new auction is held for a task, the two robots whose schedules changed may reexamine each of their tasks and bids. If one of their tasks comes at a much higher cost now, they may reinitiate the auction. Likewise, if a bid would have been higher (or been possible to make at all) given the new schedule, they may request that the task's current executor initiate a new auction to reschedule. In this manner, the auction mechanism recursively reschedules for intra-task links in the rationale graph.

### 6.2.4 Rescheduling on the CoBot Robots

We implemented centralized rescheduling with MIPs using rationale graphs from shared information on the CoBot robots. When a CoBot passes by a door, closed or open, it sends this information to the server. The CoBot has a door detector which matches its LIDAR scans to expected door locations marked on its map to determine whether the door is closed or open. The server stores this information in the database, including the location and the time the observation was made.

Next, the server checks if the new door information violates any of the rationales. consider violated positive rationales, meaning closed doors which we expected to be open. We assume that

the door information is only valid for five minutes. If the robot is scheduled to visit a closed door within the next five minutes, then rescheduling is performed. We could also consider negative rationales, if we also rescheduled when any door which is currently marked as closed is now detected as open, rescheduling in the same manner as for positive rationales.

When rescheduling, we add a penalty to scheduling a task when a door is expected to be closed. This rescheduling approach is for the MIP scheduler, although different approaches could be applied to other scheduling algorithms. We add an additional term to the objective function, to minimize $K \sum_i d_i$ where $K$ is a constant and $d_i$ is a binary variable indicating whether task $i$ was scheduled when a door was expected to be closed. For each task $i$ where we observed a closed door within the last five minutes, we also add the constraints

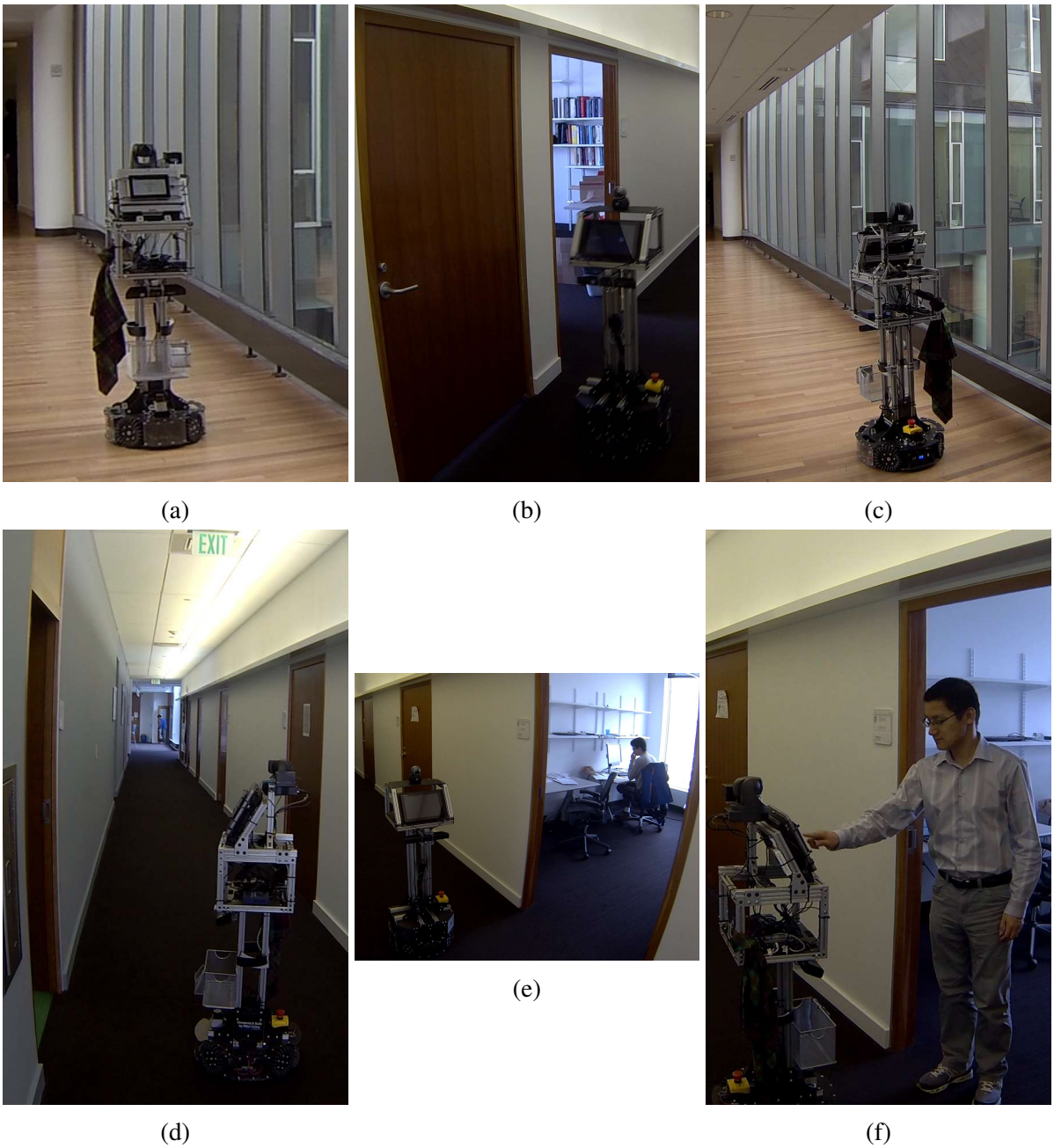
$$\begin{aligned} t_i - (D_i + 300) &\leq M(1 - d_i) \\ t_i - (D_i + 300) &\geq -M d_i \end{aligned}$$

to the MIP, where $t_i$ is the scheduled execution time of task $i$, $D_i$ is the time in seconds we observed the closed door, and $M$ is a large constant. This sets $d_i$ to 1 if $D_i \leq t_i \leq D_i + 300$ and 0 otherwise. By adding penalties instead of hard constraints, we are able to schedule attempting to deliver items to a closed door if no other better schedule exists.

Figure 6.6 shows an example of two robots executing this rescheduling algorithm. CoBot-2 is heading to Office A to deliver an object. CoBot-4, in an unrelated task, passes by Office A and observes that the door is closed. It communicates this information to the server, which reschedules the delivery to five minutes later, and CoBot-2 proceeds to head to Office B to complete what was originally the second task in its schedule, as it now has time. After five minutes, CoBot-2 can head to Office A and complete its delivery.

One feature of this rescheduling algorithm is that if Office A had remained closed, CoBot-2 would have observed this upon making its delivery, informed the server, and if there was time later, reschedule the delivery for later. The robot would keep delaying the task as many times as necessary, checking back until the door is opened. Once the time window expires or there is no time in the schedule to fit in the task later, CoBot-2 would attempt to make the delivery anyway, giving up after a few minutes and reporting the task as failed.

Blocked hallways are treated slightly different from closed doors. Rather than changing the schedule of tasks based on blocked hallways, we plan different paths to avoid the obstacles. As before, blocked hallways are recorded on the server and rationale violations, or robots which plan to use these blocked paths in the near future, are detected. Instead of rescheduling, the robots with violated rationales are informed of these blockages. A penalty is added to these corridors in




(a) (b) (c)

(d) (e) (f)

Figure 6.6: a CoBot-2 heads to Office A to make a delivery. b CoBot-4 passes Office A and observes the door is closed. It notifies the server. The server sees a violation in CoBot-2's rationale graph and reschedules its tasks. c CoBot-2 is scheduled to complete a task at Office B first, and turns around. d CoBot-2 completes its task at Office B. e CoBot-4 returns from its previous task and observes that the door to Office A is now open. f CoBot-2 completes its original task at Office A.

(a) CoBot-2 to Office (b) CoBot-4 Blocked (c) CoBot-2 Replans (d) Obstacle Avoided

Figure 6.7: a CoBot-2 heads towards an office to make a delivery. b CoBot-4 detects that a hallway is blocked, and tells the server. The server realizes that CoBot-2's rationale has been violated and communicates this information to CoBot-2. c CoBot-2 uses the information to change its plan. d CoBot-2 takes an alternate round to avoid the blocked hallway.

the path planner so that the robots avoid them if possible. When a robot finally does manage to traverse the blocked corriodr, as well as after a fixed time without any observations, the hallway is updated as unblocked and robots replan using that corridor.

Figure 6.7 shows one example of this algorithm executing on the CoBots. CoBot-2 is heading towards an office to make a delivery. However, CoBot-4, while heading down the same hallway in the opposite direction, encounters an obstacle and is blocked. It proceeds to communicate this information to the server, which detects a rationale violation in CoBot-2's schedule and informs CoBot-2. CoBot-2 replans its path and selects an alternate route to its destination which avoids the blocked hallway. The original and revised routes are shown in Figure 6.8.

## 6.3 Chapter Summary

We have introduced an auction-based algorithm to schedule pickup and delivery problems with transfers and time windows. The algorithm runs online and replans in response to new requests, dead vehicles, and shared information. We have demonstrated the schedules formed on robots and in large simulated problem instances. Furthermore, we have discussed centralized and distributed algorithms to reschedule from shared information, and demonstrated rescheduling from observations of closed doors and blocked hallways.

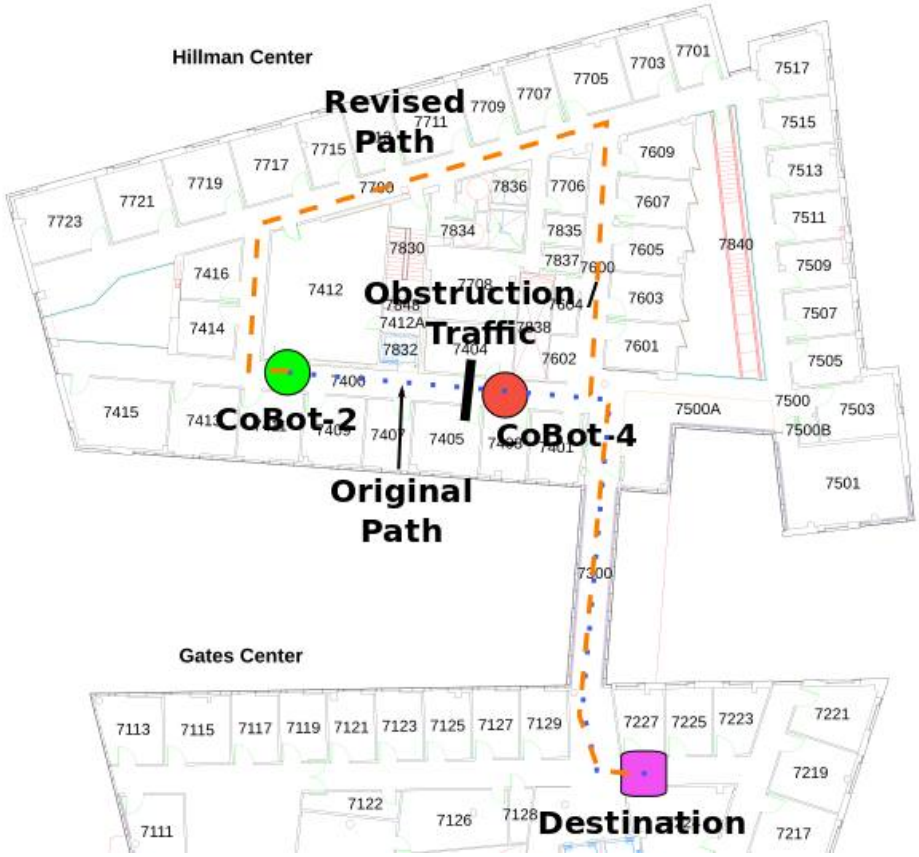



Figure 6.8: CoBot-2 replans its route after CoBot-4 reports that a hallway is blocked.

# Chapter 7

# Very Large Neighborhood Search with Transfers

We have previously presented algorithms to solve the PDP-T optimally, as well as heuristics to solve the PDP-T either offline or online. In this chapter, we introduce a metaheuristic, Very Large Neighborhood Search with Transfers (VLNS-T), to solve the PDP-T offline. The metaheuristic takes a much longer time to run that the heuristics presented previously, and cannot be run online. However, it searches over a larger space of solutions, and finds higher quality solutions.

Like the VLNS algorithm for the PDP presented in [93], VLNS-T forms a schedule with simulated annealing, where "neighboring" states are found by removing a randomized set of items from the proposed scheduled, then reinserting the items into the schedule using a set of heuristics. We introduce two new insertion heuristics which insert transfers into the schedule: one which "splits" existing segments of an item's route from a single vehicle with a second vehicle (similar to the heuristic from the previous chapters, except able to handle additional constraints), and another heuristic which inserts an item into an existing schedule where the item is picked up by one vehicle, transfered to another, and delivered by the second vehicle.

We also introduce an algorithm which, given heuristic proposes an ordering for the vehicles' actions, and a lower-level routine, given a set of vehicular plans, determines whether and how they can satisfy the vehicles' and items' time constraints. Here we consider the full PDP-T introduced in Chapter 2. For the first time in this thesis, we consider all of the allowed constraints at once, particularly the maximum vehicle route duration and maximum item transportation time. These constraints require a more complex approach to determine action times than the simple temporal networks of the previous section.

We evaluate the VLNS-T algorithm on benchmark problems and transportation problems derived from real world taxi rides in New York City, demonstrating the benefits of allowing

transports in both cases compared to state of the art PDP algorithms.

## 7.1 The VLNS-T Algorithm

Our algorithm to search for solutions to the PDP-T consists of three parts. From highest to lowest level, these are:

1. **Very Large Neighborhood Search**. The VLNS algorithm uses simulated annealing to randomly choose "neighboring" schedules and iteratively improve the schedule. Neighboring schedules are formed by removing random items and reinserting them with heuristics.
2. **Greedy Item Insertion Heuristics**. These heuristics insert items into the schedule (potentially with transfers) to find a new "neighboring" schedule.
3. **Valid Schedule Determination**. The insertion heuristics generate a list of actions for each vehicle, but not their execution times. Another routine computes the best times for the actions and determines if the schedule is valid.

We discuss these components of the algorithm in order.

### 7.1.1 Very Large Neighborhood Search

VLNS-T is based on the Adaptive VLNS algorithm introduced in [93] for the PDP without transfers. This is a variant of simulated annealing in which the neighborhood of states is "very large". In this case we remove random items from the schedule and then reinsert them with multiple heuristics to find "neighbors".

Alg. 10 shows the VLNS-T algorithm. On each iteration, a random number of items $q$ are removed from the schedule using one of several heuristics. Next, a second heuristic attempts to insert items that are not part of the schedule. The new schedule is accepted with a probability based on the temperature, which decreases over time. The initial temperature is based on the cost of the initial schedule, but with $\gamma = 0$ to ignore undelivered items. We use the same parameter values as [93].

For the `random_item` and `insert_item` routines, the VLNS algorithm is *adaptive* in the sense that it chooses from sets of heuristics and learns over time which removal and insertion heuristics are the most effective. For each insertion or removal heuristic $i$, we compute a weight $w_{j,i}$ over each adaptive "segment" $j$ of length $\tau$ iterations of VLNS. Removal and insertion heuristics are selected randomly in proportion to their weight. Initially all heuristics have equal weight. The weight is modified by a different amount depending on whether the heuristic gives

**Algorithm 10** `vlns_t`$(S)$: Form a schedule with VLNS-T from starting schedule $S$.

$S_{best} \leftarrow S$
$T \leftarrow \frac{\omega}{-\ln 0.5} \boldsymbol{c}(S, \gamma = 0)$
**for** $i \in 1, .., N$ **do**
  $q \leftarrow$rand. integer in $[\texttt{min}(4, |P|), \texttt{min}(100, \xi|P|)]$
  $R \leftarrow \texttt{random\_items}(S, q)$
  $S' \leftarrow \texttt{remove\_items}(S, R)$
  $S' \leftarrow \texttt{insert\_items}(S', \boldsymbol{UND}(S))$
  **if** $\boldsymbol{c}(S') < \boldsymbol{c}(S_{best})$ **then**
    $S_{best} \leftarrow S'$
  **end if**
  **if** $random() < \exp -(\boldsymbol{c}(S') - \boldsymbol{c}(S))/T$ **then**
    $S \leftarrow S'$
  **end if**
  $T \leftarrow cT$
**end for**
**return** $S_{best}$

a new best overall solution, a new solution better than the previous solution, or a newly accepted solution worse than the current solution. All the parameters for VLNS-T and the values we use are shown in Table 7.1.

### Removal Heuristics

For `random_item`, we use the three heuristics introduced in [93]: the *Shaw* removal heuristic, the *random* removal heuristic, and the *worst* removal heuristic.

The *Shaw* heuristic chooses items similar in distance, time, and demand to remove based on a relatedness metric which depends on location, time, and demand, choosing the first item at random and the remainder in proportion to their relatedness.

$$\begin{aligned} R(a,b) &= \phi(d(\boldsymbol{start}(a), \boldsymbol{start}(b)) + d(\boldsymbol{end}(a), \boldsymbol{end}(b))) + \\ &\quad \chi(|\boldsymbol{t}(a_{pickup}) - \boldsymbol{t}(b_{pickup})| + \\ &\quad |\boldsymbol{t}(a_{delivery}) - \boldsymbol{t}(b_{delivery})|) + \\ &\quad \psi\,|\boldsymbol{dem}(a) - \boldsymbol{dem}(b)| \end{aligned}$$

where $a_{pickup}$ and $a_{delivery}$ give the pickup and delivery actions for item $a$, respectively. The heuristic selects an initial item at random, then chooses the remaining $q-1$ items with probability proportional to their relatedness to the first. The *random* removal heuristic simply selects $q$ distinct items at random to remove, with equal probability. For the *worst* removal heuristic,

| Name | Value | Description |
|---|---|---|
| $\alpha$ | 1 | Cost weight to distance travelled. |
| $\beta$ | 0 | Cost weight to vehicle operation time. |
| $\gamma$ | $10^6$ | Cost weight to unassigned items. |
| $N$ | 25000 | Num. VLNS iterations. |
| $\xi$ | 0.4 | Maximum fraction removed items. |
| $\omega$ | 0.05 | Initial VLNS temperature weight. |
| $\tau$ | 100 | Adaptive segment size. |
| $\sigma_1$ | 33 | Adaptive score, best overall solution. |
| $\sigma_2$ | 9 | Adaptive score, new improved solution. |
| $\sigma_3$ | 13 | Adaptive score, new worse solution. |
| $r$ | 0.1 | Adaptive reaction factor. |
| $\phi$ | 9 | Relatedness distance weight. |
| $\chi$ | 3 | Relatedness time weight. |
| $\psi$ | 2 | Relatedness demand weight. |
| $\eta$ | 0.025 | Noise rate. |

Table 7.1: VLNS Parameters

items are selected for removal at random in proportion to the difference in cost of the current schedule and the current schedule with the item in question removed. With this heuristic, items which are more costly to deliver are more likely to be removed.

### Insertion Heuristics

Two greedy heuristics are used to insert items into the schedule with transfers. The first, the `split_routes` heuristic, takes existing pickup and delivery item route segments from a single vehicle and splits them with another vehicle. It is based on the `insert_transfer` heuristic from the previous chapter. The second heuristic, the `insert_item_with_transfer` heuristic, searches over possible pickups and deliveries with a single transfer between a pair of vehicles.

Both heuristics are greedy but in different ways. Each relies on another heuristic to insert item pickups and deliveries into the schedule without transfers, the same heuristic used in [93]. This heuristic is called `greedy_insert`$(items, regret, noise)$ and takes as parameters the items to insert, a value for the "regret", which is an integer, and an amount of noise to apply to the objective function. If we define $c_{i,j}$ as the $j^{th}$ best cost to insert item $i$ over all points to insert *Pickup* and *Delivery* actions, we iteratively choose item $i$ to insert such that $c_{i,regret} - c_{i,1}$ is maximized. VLNS-T chooses from a family of `split_routes` heuristics, with regrets of 1, 2, 3 and 4. For each value of regret, there are two choices of noise: one with 0 noise and one with noise $\eta \max d(l_1, l_2)$ where the noise is proportional to the maximum distance between any

two locations specified in the problem. When we compute the cost in the greedy algorithm, we sample from $c(S) \sim max(0, N(c(S), \sigma^2))$ to increase the diversity of generated solutions. For further details regarding the `greedy_insert` heuristic, see [93].

### 7.1.2 Greedy Insertion with Transfers

We first detail the algorithms used to insert item pickups and deliveries into an existing schedule with transfers. We are given an initial schedule $S$ and a set of items $M_{new}$, with the goal of returning a schedule of low cost with the items $M_{new}$ added into the schedule. It is prohibitively expensive to search over all possible schedules, so we use two greedy heuristics which iteratively insert individual items at a time. First we discuss an algorithm for inserting transfer points into vehicle schedules before delving into the higher-level greedy heuristics.

#### Transfer Insertion

Both heuristics rely on the algorithm `insert_split`$(S, v_1, v_2, m, T_{rec}, T_{send}, c_{best})$ , shown in Alg. 11, to insert transfers into the schedule. This routine finds the lowest cost schedule to transfer item $m$ from vehicle $v_1$ to vehicle $v_2$. Vehicle $v_1$ receives item $m$ with action $T_{rec}$ (either a *Pickup* or *Transfer* action) which must be inserted into the schedule, or with an action that is already part of the schedule, in which case $T_{rec} = \emptyset$. Likewise the action $T_{send}$ may be either a *Deliver* or *Receive* action to insert into $S^{v_2}$ after the new *Transfer* action, or $\emptyset$ if this action is already part of the plan. The `insert_split` algorithm searches over all possible places to insert the new *Receive* and *Transfer* actions into $S^{v_1}$ and $S^{v_2}$, respectively. Then it searches over all possible insertion points for the actions $T_{rec}$ and $T_{deliver}$, if applicable.

For each pair of insertion points for the new transfer actions, we compute the location of the transfer point with `find_transfer_point`$(a_1, a_2, b_1, b_2)$, which computes a transfer point given the endpoints to two line segments, $a$ and $b$, between which the new transfer point is inserted on the two vehicles' schedules. On the Euclidean plane, we return the lines' intersection or the endpoints' mean if they do not intersect. This is not necessarily the optimal transfer point, which may depend on the timing of the two vehicles' actions, but it is a reasonable and computationally inexpensive choice. A different method of finding a transfer point may be chosen without changing the higher level algorithm, such as choosing from a list of suitable points.

For every considered set of action insertions, the satisfiability of the proposed schedule's time constraints is checked with the `update_times` algorithm, which is explained in a later section. We also check that the new actions we insert do not create a cycle of transfers such that the vehicles deadlock. To check this, we start from the actions following each of the two provided

**Algorithm 11** `insert_split`$(s, a, b, m, T_{rec}, T_{send}, c_{best})$: Split item $m$'s delivery between vehicles $a$ and $b$ in schedule $s$. Optionally insert the task when $a$ receives $m$, $T_{rec}$, and when $b$ delivers or transfers away $m$, $T_{send}$, into the plans as well. These tasks may be $\emptyset$ if they are already part of the plans. An accepted insertion must have cost less than $c_{best}$.

$s_{best} \leftarrow \emptyset$
$astart \leftarrow 0$ **if** $T_{rec} \neq \emptyset$ **else** index of action with $m$ in $S^a$
$bend \leftarrow 0$ **if** $T_{send} \neq \emptyset$ **else** index of action with $m$ in $S^b$
**for** $i$ from $astart$ to $|S^a| - 1$ **do**
  **for** $j$ from 0 to $bend$ **do**
    $loc \leftarrow$ `find_transfer_point`$(S^a_i, S^a_{i+1}, S^b_j, S^b_{j+1})$
    $T_{ts} \leftarrow$ TRANSFER$(b, m, loc), T_{tr} \leftarrow$ RECEIVE$(a, m, loc)$
    $s' \leftarrow$ `insert`$(s, S^a, i+1, T_{ts})$
    $s' \leftarrow$ `insert`$(s', S^b, j+1, T_{tr})$
    **for** $k$ from $i+1$ to ($|S^a| - 1$ **if** $T_{rec} \neq \emptyset$ **else** $i+1$) **do**
      $s'' \leftarrow s'$
      **if** $T_{rec} \neq \emptyset$ **then**
        $s'' \leftarrow$ `insert`$(s', S^a, k+1, T_{rec})$
      **end if**
      **for** $l$ from 0 to ($j$ **if** $T_{send} \neq \emptyset$ **else** 0) **do**
        $s''' \leftarrow s''$
        **if** $T_{send} \neq \emptyset$ **then**
          $s''' \leftarrow$ `insert`$(s'', planb, l+1, T_{send})$
        **end if**
        $c \leftarrow \mathbf{c}(s''') - \mathbf{c}(s)$
        **if** $c \geq c_{best}$ or ($T_{rec}$ transfer and $cycle(T_{rec}, \mathbf{pair}(T_{rec}))$) or ($T_{send}$ transfer and $cycle(T_{send}, \mathbf{pair}(T_{send}))$) or $cycle(T_{tr}, T_{ts})$ or capacities violated **then**
          **continue**
        **end if**
        **if** `update_times`$(s''', [a, b])$ **then**
          $s_{best} \leftarrow s'''$, $c_{best} \leftarrow c$
        **end if**
      **end for**
    **end for**
  **end for**
**end for**
**return** $s_{best}$

paired actions and move forward through the schedule, checking that we do not later reach one of the two initial actions again.

The `insert_split` algorithm makes use of many checks to skip invalid insertion points and speed up the search. For example, the available capacity of the vehicles is tracked through the loops, and task insertion points which would violate capacity constraints are skipped. Similarly, insertion points which could not possibly satisfy the time constraints are not checked. Additionally, partial costs of adding transfer actions are calculated, and the insertion is skipped if adding the transfer actions alone exceeds cost $c_{best}$.

### Greedy Route Splitting

For the first heuristic, we take segments of each item's route on individual vehicles and "split" those segments with other vehicles in the `split_single_route`$(S, v_1, v_2, m)$ algorithm as shown in Alg. 12. For example, if vehicle $v_1$ has the actions $a_1 = \text{PICKUP}(m)$ and $a_2 = \text{DELIVER}(m)$ in its plan, `split_single_route`$(S, v_1, v_2, m)$ will attempt to make two insertions using the `insert_split` routine. First, it attempts to remove $a_1$ from $S^{v_1}$, inserting an action $\text{RECEIVE}(v_2, m, l)$ in its place and adding the actions $\text{PICKUP}(m)$ and $\text{TRANSFER}(v_1, m, l)$ into $S^{v_2}$. The route splitting routine also attempts to remove $a_2$ from $S^{v_1}$, insert an action $\text{TRANSFER}(v_2, m, l)$ instead, and add the actions $\text{RECEIVE}(v_1, m, l)$ and $\text{DELIVER}(m)$ to $S^{v_2}$. The routine may be applied to make multiple transfers if the actions $a_1$ and $a_2$ are transfer actions.

The first heuristic, shown in Alg. 13, first greedily inserts items into the plan without transfers. Then, it selects random items and greedily inserts the best transfers into their schedules using `split_single_route`. If a transfer can be inserted, we attempt to insert more transfers recursively. Due to the increased possibilities for where to insert transfers and the consequential increase in computational cost, unlike `greedy_insert`, we split the route of one item at a time instead of choosing the best split over all routes. To further speed up the algorithm, we pass the cost of the best split found so far to `insert_split` to more quickly reject costly transfer points.

### Split Item Insertion

For the second heuristic, instead of adding transfers to existing item delivery routes, we plan for a single transfer point from the beginning. We go through the items in a random order, and attempt to insert each in the schedule without transfers. We then apply the `insert_split` routine to see if a delivery can be made at lower cost by using a single transfer point, exploring the possible insertion points for the *Pickup*, *Deliver*, *Receive* and *Transfer* actions. The

**Algorithm 12** `split_single_route`$(s, v_1, v_2, p, c_{best})$: Split item $p$'s route segment on $v_1$ with $v_2$.

$c_{best} \leftarrow \infty$
$a_1 \leftarrow$ first action involving $p$ in $S^{v_1}$
$a_2 \leftarrow$ second action involving $p$ in $S^{v_2}$
$s' \leftarrow s, s'' \leftarrow s$
**if** $a_1 = \text{RECEIVE}(v_3, p, l)$ **then**
  $a_1' \leftarrow \text{RECEIVE}(v_3, p, l)$
  $s' \leftarrow s$ with *pair*$(a_1')$'s second vehicle as $v_2$
**else**
  $a_1' \leftarrow \text{PICKUP}(p)$
**end if**
**if** $a_2 = \text{TRANSFER}(v_3, p, l)$ **then**
  $a_2' \leftarrow \text{TRANSFER}(v_3, p, l)$
  $s'' \leftarrow s$ with *pair*$(a_2')$'s second vehicle as $v_2$
**else**
  $a_2' \leftarrow \text{DELIVER}(p)$
**end if**
$s_1 \leftarrow$ `insert_split`$(s', m, v_2, v_1, a_1', \emptyset, c_{best})$
$s_2 \leftarrow$ `insert_split`$(s'', m, v_1, v_2, \emptyset, a_2', c_{best})$
**return** $\text{argmin}_{s \in s_1, s_2} \boldsymbol{c}(s)$

**Algorithm 13** `split_routes`$(S, M_{new}, r, \sigma^2)$: Insert items $M_{new}$ into the schedule $S$, with the specified regret $r$ and noise $\sigma^2$.

$S \leftarrow$ `greedy_insert`$(S, M_{new}, r, \sigma^2)$
$M_{new} \leftarrow M_{new} \setminus \textit{UND}(S)$
**while** $|M_{new}| > 0$ **do**
  Pop random item $m$ from $M_{new}$
  $V_m \leftarrow$ set of vehicles that transport $m$ in $S$
  $S_b \leftarrow S$, $c \leftarrow \boldsymbol{c}(S)$, $c_b \leftarrow N(c, \sigma^2)$
  **for** $v_1 \in V_m$, $v_2 \in V \setminus V_m$ **do**
    $S' \leftarrow$ `split_single_route`$(S, v_1, v_2, m, c - \boldsymbol{c}(S_b))$
    $c' \leftarrow N(\boldsymbol{c}(S'), \sigma^2)$
    **if** $c' < c_b$ **then**
      $S_b \leftarrow S'$, $c_b \leftarrow c'$
    **end if**
  **end for**
  **if** $S_b \neq S$ **then**
    $S \leftarrow S_b$, $M \leftarrow M \cup \{m\}$
  **end if**
**end while**
**return** $S$

`insert_item_with_transfer` heuristic is shown in Alg. 14.

**Algorithm 14** `insert_item_with_transfer`$(s, items)$: Insert $items$ into the schedule $s$.

```
while |items| > 0 do
  m ← pop random item from items
  s_best ← greedy_insert(s, [m], 1, 0)
  for v_1 ∈ V, v_2 ∈ V, v_1 ≠ v_2 do
    s' ← insert_split(s, m, v_1, v_2, PICKUP(m), DELIVER(m), c(s_best))
    if c(s') < c(s_best) then
      s_best ← s'
    end if
  end for
  s ← s_best
end while
return s_best
```

This heuristic is more expensive than the `split_routes` heuristic, since it searches for insertion points of four actions over all pairs of vehicles. However, it can find solutions to some problems which the other heuristic cannot due to its greediness. For example, if an item cannot be delivered by a single vehicle due to the maximum vehicle duration constraint, the other heuristic would not find any solution, while the `insert_item_with_transfer` heuristic would be able to.

### 7.1.3 Determining Action Execution Times

Finally, we discuss the lowest level of the PDP-T algorithm, determining the execution times of the actions in a schedule and whether the schedule is valid. The `update_times`$(S, V)$ routine is an extension of the time determination algorithm presented in [33] which computes times when transfers are involved. This function makes use of two key subroutines:

- `earliest_times`$(S, [(a_1, t_1), \ldots])$: Given a list of actions $a_i$ and corresponding execution times $t_i$, update the actions following $a_i$ in the schedule to be executed as early as possible, obeying only time window constraints. This is Step 2 in the algorithm detailed in [33]. We extend this algorithm for transfers such that when a transfer action $a$'s times are updated, we update the times of the actions following $pair(a)$, as shown in Algorithm 15.
- `fslack`$(a, M_{new}, M_{ex})$: Determine the amount of time the execution of task $a$ can be delayed without violating additional constraints. These constraints include the transport time constraints for only items in $M_{ex}$. See Alg. 16 for details. The slacks are computed recursively for transfer actions. We also keep track of whether the lowest slack occurred

**Algorithm 15** `earliest_times`$(s, Q = [(a_1, t_1), \ldots)$: Update the times for the actions following those in $Q$ to be as early as possible obeying time window constraints.

**while** $|Q| > 0$ **do**
  $(a, t) \leftarrow$ pop item with lowest $t$ from $Q$
  **if** $t(a) \geq t$ **then**
    **continue**
  **end if**
  $t(a) \leftarrow \min(t, e(a))$
  **if** $t(a) > l(a)$ **then**
    **return** False
  **end if**
  **if** $a$ is *Receive* or *Transfer* **then**
    Add $(pair(a), t(a))$ to $Q$
  **end if**
  **if** $a' =$ action following $a$ in plan exists **then**
    $b(a') \leftarrow \max(b(a'), t(a) + \delta(a) + d(loc(a), loc(a'))/a_{vvel})$
    Add $(a', b(a'))$ to $Q$
  **end if**
**end while**
**return** True

due to an item's maximum transport time constraint that has not yet been satisfied. If so, we may need to redo the computation later.

Given these two routines, the `update_times`$(S, V)$ algorithm executes the following steps to update the action arrival and execution times and determine schedule validity:

1. *Determine affected vehicles.* Add all vehicles to $V$ which transfer to or from vehicles in $V$. The times for these vehicles' actions must also be updated.
2. *Compute the earliest possible execution times.* Call the `earliest_times`$(S, A, T)$ function, where $A$ is contains the initial action for each vehicle $v \in V$ and $T$ contains the earliest possible start time $e(v)$. If any time window constraints are not satisfied, the function returns false and the schedule is invalid.
3. *Enforce maximum vehicle route durations.* Call `earliest_times`$(S, A, T)$ again with the initial actions $A$ as before, but with $t \in T$ corresponding to $a \in A$ for vehicle $v$ set to $e(v) + \min(slack, sum)$ where $(slack, sum, redo) =$ `fslack`$(a, \emptyset, \emptyset)$. This delays the start of each vehicle's execution as much as possible while still obeying time window constraints. If after this step any vehicle $v$'s route duration exceeds $mrd(v)$, the schedule is invalid. If the problem has no maximum route duration constraints, this step can be skipped.

**Algorithm 16** `fslack`$(action, M_{new}, M_{ex})$: Determine the time $a$'s execution can be delayed without violating additional constraints. Returns a tuple containing the slack, the total waiting time, and whether the slack depends on pickup actions to be delayed which are not in $M_{ex}$, the set of items which have already had their pickup actions delayed.

$slack \leftarrow \infty$, $\Sigma_{wait} \leftarrow 0$, $redo \leftarrow$ False
**for** $a$ from $action$ to end of plan **do**
  **if** $a \neq action$ **then**
    $\Sigma_{wait} \leftarrow \Sigma_{wait} + (\boldsymbol{t}(a) - \boldsymbol{b}(a))$
    **if** $a$ is *Transfer* or *Receive* **then**
      $(s_1, \Sigma_1, r_1) \leftarrow$ `fslack`$(a, M_{new}, M_{ex})$
      $(s_2, \Sigma_2, r_2) \leftarrow$ `fslack`$(\boldsymbol{pair}(a), M_{new}, M_{ex})$
      $s_1 \leftarrow s_1 + \Sigma_{wait}$, $s_2 \leftarrow s_2 + \Sigma_{wait}$
      $\Sigma_{wait} \leftarrow \Sigma_{wait} + \min(\Sigma_1, \Sigma_2)$
      **if** $s_1 < slack$ **then**
        $slack \leftarrow s_1$, $redo \leftarrow r_1$
      **end if**
      **if** $s_2 < slack$ **then**
        $slack \leftarrow s_2$, $redo \leftarrow r_2$
      **end if**
      **return** $(slack, \Sigma_{wait}, redo)$
    **end if**
  **end if**
  $redo' \leftarrow False$, $slack' \leftarrow l(a) - \boldsymbol{t}(a)$
  **if** $a =$ DELIVER$(m)$, $\exists \boldsymbol{mtt}(m)$, $m \notin M_{new}$ **then**
    $x \leftarrow \boldsymbol{t}(m_{delivery}) - (\boldsymbol{t}(m_{pickup}) + \delta(m_{pickup}))$
    $slack' \leftarrow \max(0, \min(slack, \boldsymbol{mtt}(m) - x))$
    $redo' \leftarrow (m \notin M_{ex})$
  **end if**
  $slack' \leftarrow slack' + \Sigma_{wait}$
  **if** $slack' < slack$ **then**
    $slack \leftarrow slack'$, $redo \leftarrow redo'$
  **end if**
**end for**
**return** $(slack, \Sigma_{wait}, redo)$

4. *Enforce maximum transportation times.* For each item $m$, in increasing order of the time $t(m_{pickup})$, delay the action execution by $\min(slack, sum)$ where $(slack, sum, redo) = \texttt{fslack}(m_{pickup}, \emptyset, M_{ex})$. After the times are updated with the `earliest_times` routine, add $m$ to $M_{ex}$, and, if $redo$ is True, add $m$ to a list of actions that must be redone. After attempting to delay every item $m$, remove the items that were marked for redos from $M_{ex}$ and attempt to delay their execution again until no items are left. Retrying may be necessary because the computation of the forward slack assumes that all items are either in $M_{ex}$ with their time constraints already enforced, or the items are picked up after the starting action. Without transfers this is always the case, but with transfers the item may be picked up on a different robot such that the maximum transportation time constraint has not yet been enforced. If any maximum transportation time constraint is violated afterwards the schedule is invalid.

## 7.2 Experiments

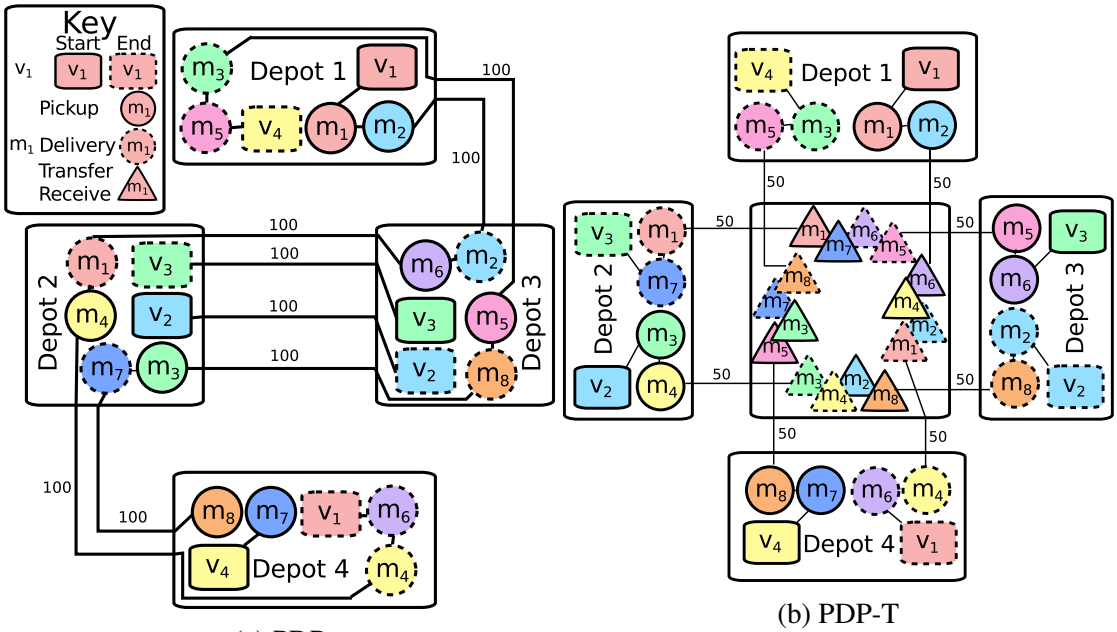
We introduce several simple problems that demonstrate the potential advantages of planning for transfers in comparison to state of the art PDP algorithms. Then we show results from solving the more complicated Cordeau benchmark problems as a PDP-T.

### 7.2.1 Example Problems

As a basic test to illustrate the effectiveness of VLNS-T, we ran VLNS-T on the problem in Figure 7.1. VLNS-T found the expected solution with a factor of two improvement.

In a second problem, ten hubs are evenly distributed around a circle of radius 50 on the Euclidean plane. One vehicle begins at each hub, and its destination is its starting hub. Items begin at random hubs, and are assigned a random destination from the three hubs on the opposite side of the circle. All time windows begin at time 0 and end at time 200. The vehicles have capacity 5 and maximum route duration $150$. Ten problem instances with $|M| = 15$ were all solved by the PDP-T algorithm, but no solution could be found for the PDP. This is because the maximum route duration of $150$ prevents vehicles from delivering items to the other side of the circle and returning to their starting position.

These two examples serve to demonstrate that VLNS-T can drastically reduce the solution cost and make previously infeasible problems feasible.




(a) PDP

(b) PDP-T

Figure 7.1: a The cost to deliver the items is 800 without transfers. b The cost is halved to 400 if transfers are allowed. Note that the order of the *Transfer* and *Receive* actions are arranged such that no deadlocking cycles are formed.

### 7.2.2 Benchmark Problems

We run VLNS-T on a set of benchmark problems [33], comparing to reported results without transfers from a Tabu search heuristic and to our own implementation of VLNS [93]. In these problems, all the vehicles begin and end their routes at the same central depot. The vehicles have maximum route durations and capacities, and the items have maximum transportation times. We set $\forall v \in V, m_1, m_2 \in M \; c_t(v, m_1, m_2) = 0.25$, run 25000 iterations of VLNS-T, and optimize for distance ($\alpha = 1$, $\beta = 0$). For these problems, we do not use the `insert_item_with_transfer` heuristic due to its increased computational cost in a domain that is already computationally expensive.

Table 7.2 shows the results. VLNS-T finds lower cost solutions for 19 of 20 problems, and still finds the best known solution for the final problem. The cost improvements may not seem large in proportion to the total scheduling cost, but note that in the PDP literature, the percentage improvement is often under 2% [90] or only a binary result of whether the best known solution was improved upon is given [93, 54], and improvements of this order of magnitude are significant. The vehicles do spend more time waiting in the solution with transfers, but waiting (the time between the arrival time and execution time at an action) is not included in the cost function for these problems.

| Problem | | | Tabu | VLNS | | | VLNS-T | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| # | $\|V\|$ | $\|M\|$ | Cost | Cost | Wait Time | Comp. Time | Cost | Tr. | Tr. Cost | Wait Time | Comp. Time |
| 01 | 3 | 24 | 190.02 | 190.02 | 224.26 | 0.06 | **186.46** | 8 | 2.00 | 481.02 | 0.23 |
| 02 | 5 | 48 | 302.08 | 303.39 | 817.68 | 0.18 | **290.89** | 8 | 2.00 | 884.39 | 1.62 |
| 03 | 7 | 72 | 532.08 | 544.00 | 805.92 | 0.35 | **531.02** | 13 | 3.25 | 810.18 | 4.78 |
| 04 | 9 | 96 | 572.78 | 581.39 | 1115.65 | 0.65 | **558.15** | 15 | 3.75 | 1309.50 | 10.89 |
| 05 | 11 | 120 | 636.97 | 640.91 | 1058.93 | 0.92 | **629.02** | 23 | 5.75 | 1084.88 | 29.73 |
| 06 | 13 | 144 | 801.40 | 805.33 | 1326.60 | 1.38 | **772.02** | 32 | 8.00 | 1668.58 | 60.55 |
| 07 | 4 | 36 | 291.71 | 291.71 | 500.39 | 0.11 | **288.82** | 6 | 1.50 | 501.41 | 0.50 |
| 08 | 6 | 72 | 494.89 | 505.74 | 415.84 | 0.34 | **480.17** | 12 | 3.00 | 459.49 | 4.68 |
| 09 | 8 | 108 | 672.44 | 675.64 | 270.70 | 0.82 | **639.64** | 21 | 5.25 | 528.98 | 18.90 |
| 10 | 10 | 144 | 878.76 | 873.58 | 742.35 | 1.44 | **861.73** | 18 | 4.50 | 846.41 | 52.85 |
| 11 | 3 | 24 | **164.46** | **164.46** | 407.33 | 0.07 | **164.46** | 0 | 0.00 | 407.33 | 0.17 |
| 12 | 5 | 48 | 296.06 | 297.67 | 369.89 | 0.24 | **292.22** | 9 | 2.25 | 487.39 | 1.43 |
| 13 | 7 | 72 | 493.30 | 491.92 | 442.73 | 0.43 | **479.75** | 14 | 3.50 | 592.78 | 4.20 |
| 14 | 9 | 96 | 535.90 | 550.37 | 538.67 | 0.85 | **527.31** | 16 | 4.00 | 776.95 | 13.38 |
| 15 | 11 | 120 | 589.74 | 589.67 | 965.71 | 1.31 | **584.60** | 24 | 6.00 | 1078.81 | 33.86 |
| 16 | 13 | 144 | 743.60 | 754.95 | 988.85 | 1.80 | **736.82** | 24 | 6.00 | 1143.21 | 47.91 |
| 17 | 4 | 36 | 248.21 | 248.21 | 159.37 | 0.13 | **245.69** | 4 | 1.00 | 337.38 | 0.40 |
| 18 | 6 | 72 | 462.69 | 466.82 | 464.36 | 0.45 | **457.82** | 10 | 2.50 | 842.90 | 5.17 |
| 19 | 8 | 108 | 601.96 | 600.24 | 525.53 | 1.14 | **585.36** | 11 | 2.75 | 686.23 | 17.31 |
| 20 | 10 | 144 | 798.63 | 811.36 | 415.74 | 2.00 | **787.36** | 18 | 4.50 | 542.79 | 51.41 |

Table 7.2: Cordeau benchmark results. All times are in hours. The "Tabu" column gives the best results for Tabu search [33], the "VLNS" section for one run of VLNS [93], and the "VLNS-T" section for one run of our algorithm. The "Tr." column is the number of transfers. Computation times are in hours.

VLNS-T finds better solutions, but this comes at a cost of computation time. We have done little optimization of our implementations of VLNS and VLNS-T as our focus is on determining the effectiveness of transfers, but heuristics to exclude transfer points from the search are a promising area for future work. The number of transfers per item is low, i.e., Problem 6 has 32 transfers for 144 items, which is $0.22$ transfers per item. An improvement in 19 out of 20 problems suggests that many PDP problems may benefit from allowing transfers. The benefit of transfers may be greater if we optimize for criteria other than distance, such as delivery time, which these benchmarks do not address.

### 7.2.3 New York Taxi Problems

We constructed a set of problems sampled from taxi routes in New York, using over 400,000 data points for Tuesday, 9/25/2012 [44]. The data includes taxi ride start and end times, and GPS coordinates of the start and end. First, we cluster the points to find the most popular pickup and dropoff locations, and select the 20 most popular points in Manhattan and the 80 most popular points elsewhere. We sample from taxi routes which start and end near these points which occur within the same hour window. Distances and travel times are estimated using OpenStreetMap with OSRM [75]. Vehicles have capacity 4.

In the first set of problems, all items are given the same hour long time window. We varied

| $\|V\|$ | $\|M\|$ | Average Improvement From Transfers | Maximum Improvement From Transfers |
|---|---|---|---|
| 30 | 25 | 5.63% | 11.60% |
| 30 | 50 | 7.59% | 15.40% |
| 30 | 75 | 6.36% | 10.80% |
| 30 | 100 | 5.21% | 9.50% |
| 40 | 25 | 4.72% | 7.04$ |
| 40 | 50 | 6.20% | 8.11% |
| 40 | 75 | 6.68% | 10.60% |
| 40 | 100 | 7.37% | 12.00% |

Table 7.3: New York City Taxi Experiment Results

$|V|$ from 20 to 40 and $|M|$ from 25 to 100, solving 20 instances for each parameterization. A summary of the results is shown in Table 7.3. For the second set of problems, we began the item time windows at the pickup times in the original data, and set the delivery deadline as the actual delivery time plus the estimated travel time. For $|V| = 30$ and $|M| = 60$, the average improvement was $6.78\%$, and the maximum was $10.4\%$. For all selected $|V|$ and $|M|$, the average improvement of VLNS-T over VLNS varied between $4.27\%$ and $6.82\%$. These results indicate that transfers can improve efficiency in real-world transportation domains, for example alternative shared taxi services or ridesharing.

## 7.3 Chapter Summary

We have introduced the VLNS-T algorithm, which searches over schedules schedules by removing and reinserting a randomized set of items. We introduced two heuristics to insert items into a schedule with transfers, and an algorithm to compute the execution time for actions in such a schedule. We experimentally demonstrated the large cost savings possible from VLNS-T in comparison to a state of the art PDP algorithm without transfers, both on a set of benchmark problems and on transportation problems on real-world maps.

# Chapter 8

# Background and Related Work

Next, we provide an overview of related work pertaining to this thesis and how the thesis fits in. Broadly speaking, this work falls at the intersection of two fields: scheduling and robotics. The scheduling community has studied the PDP problem extensively, along with other, more general variants of the Vehicle Routing Problem (VRP), of which PDPs are a subset. This thesis extends the scheduling community's work on the PDP by considering transfers. The robotics community has extensively studied the related problems of task allocation, distributed algorithms, and robot rendezvous. Furthermore, prior researchers have deployed robots that deliver items. We have built on this robotics work as well to execute PDP-T solutions on robots.

## 8.1 Scheduling: The Vehicle Routing Problem

The scheduling community has conducted extensive research on the Vehicle Routing Problem (VRP). In the VRP, a set of vehicles must visit a set of locations to service customer requests. A solution to the VRP is a *schedule*, which assigns requests to vehicles and provides an ordering in which each vehicle should visit its assigned requests.

Table 8.1 shows an overview of major variants of the VRP. Types of VRPs are widely varied, and their naming in the literature is inconsistent. Here, we attempt to provide a broad overview. We refer the reader to more thorough taxonomies [41] and surveys [87, 88, 113] for further details.

### 8.1.1 The Traveling Salesman Problem

The most famous variant of the VRP is the Traveling Salesman Problem (TSP), in which a single traveling salesman finds the shortest path to visit a set of cities, sells his wares, and returns

| Problem | Description | Vehicles | Items | Ref. |
|---|---|---|---|---|
| Traveling Salesman Problem (TSP) | Service a set of locations and return to the starting location. | Single | None | [28] |
| Multiple TSP | Multiple vehicles service a set of locations and return to the starting location. | Multiple | None | [10] |
| Single Vehicle Routing Problem with Backhauls | Service a set of pickup and delivery requests to or from central depots with a single vehicle. | Single | To / From Depots | [87] |
| Multi-Vehicle Routing Problem with Backhauls | Service a set of pickup and delivery requests to or from central depots. | Multiple | To / From Depots | [87] |
| Single Vehicle Pickup and Delivery Problem (PDP) | Service a set of pickup and delivery requests with a single robot, then return to the depot. | Single | Multiple | [50] |
| Multi-Vehicle Pickup and Delivery Problem (PDP) | Service a set up pickup and delivery requests to and from arbitrary locations. | Multiple | Multiple | [88] |

Table 8.1: Selected variants of the Vehicle Routing Problem.

home. The TSP is NP-hard. However, constant factor approximation algorithms exist to find solutions within a set factor of the optimal solution in polynomial time. For the general TSP, a heuristic that creates a tour on the edges of the minimum spanning tree gives a two-approximation algorithm [94]. For metric TSP, the best known approximation is 1.5-approximate [28], and for the Euclidean TSP, an algorithm which gives a $(1 + \epsilon)$-approximation in polynomial time is known [4, 79].

The multi-TSP extends the traveling salesman problem to multiple salesmen. The multi-TSP has been addressed with integer programs, other exact methods, and with heuristics [10].

### 8.1.2 The Vehicle Routing Problem with Backhauls

The remaining classes of problems deal with transporting items, rather than only visiting locations. First, we discuss the Vehicle Routing Problem with Backhauls (VRPB). In problems of this class, vehicles deliver items to customers from a central depot and deliver items from customers to a central depot or depots, but do not deliver items between customer locations. The vehicles may deliver all the items before retrieving any, or may intermix deliveries and pickups. In the simultaneous delivery and pickup problem, vehicles must both deliver and retrieve a single customer's items in one stop. The VRPB may have additional constraints, such as capacities [112], time windows [40], and multiple depots [81]. The VRPB can be solved optimally

[87], with many proposed heuristics [47, 37, 36, 81], or with metaheuristics such as tabu search [40, 2], simulated annealing [85], genetic algorithms [5], ant colony optimization [25], and particle swarm optimization [60].

### 8.1.3 The Pickup and Delivery Problem

The last variant of the VRP we discuss is the Pickup and Delivery Problem (PDP), which is the focus of this thesis. Here, customer requests involve retrieving items from one location and delivering them to another location. The pickup and delivery locations are not limited to central depots. The PDP is sometimes called the Dial-a-Ride-Problem (DARP), named after how users dial a taxi for a ride [12]. The only difference between the two problems is that DARP plans for passengers, so user convenience (i.e., the time the passengers spend in transit) is often valued more highly. We use Table 8.2 to describe the many variants of the PDP, such as including capacities and time windows.

| Type | Name | Description |
|---|---|---|
| Vehicles | Capacities | Vehicles may carry a limited number of items. |
| | Split Deliveries | Loads can be divided between multiple vehicles. |
| | Vehicle Destinations | Vehicles have their own ultimate destinations (ridesharing). |
| | Limited Fuel | Vehicles can only travel a limited distance before refueling. |
| | Heterogeneous Vehicles | Vehicles have different capacities, speeds and other capabilities (i.e., cars, trucks, boats and planes). |
| Time | Release Times | Items can only be picked up at a specified release time. |
| | Deadlines | Items must be delivered by a deadline. |
| | Time Windows | Items have both release times and deadlines. |
| | Soft Time Windows | Time windows can be violated, but with a penalty. |
| Requests | Request Priorities | Certain tasks are more important to complete. |
| | Customer Preferences | Penalties may be incurred based on item to vehicle assignment. |
| Dynamic | Stochastic Service Times | Vehicles spend a variable time servicing customers. |
| | Stochastic Travel Times | Vehicles spend a variable time traveling routes. |
| | Dynamic Requests | Customer requests are not known beforehand. |
| | Dynamic Destinations | Customer destinations are not known in advance. |
| | Dynamic Vehicles | Available vehicles are not known beforehand (ridesharing). |
| | Vehicle Failures | Vehicles may break down and become inoperable. |
| Transfers | Transfer Cost | Vehicles pay a penalty to transfer items. |
| | Fixed Transfer Points | Vehicles can transfer items at fixed points. |
| | Dynamic Transfer Points | Vehicles can transfer items anywhere. |
| | Distributed | Vehicles must plan and communicate locally. |

Table 8.2: Selected variants of the Pickup and Delivery Problem (PDP).

The PDP has been solved exactly with branch and bound methods [92, 73] for small problem instances. It has also been addressed for larger problems with numerous heuristics such as con-

struction heuristics [74] and column generation heuristics [116]. The other approaches to this class of problems have largely consisted of metaheuristics, such as Tabu search [83, 71], genetic algorithms [59], and simulated annealing [11, 93].

A few approximation algorithms with guarantees have been developed for the VRP as well. Approximation algorithms have been proposed for requests with release times at which jobs can first be performed but without deadlines [15], with time windows for a single vehicle [7], and for the capacitated single vehicle pickup and delivery problem [27, 51].

### 8.1.4 Dynamic Pickup and Delivery Problems

In dynamic pickup and delivery problems, requests come in over time and are not known beforehand. The two general approaches used to solve these problems are to apply static solutions as new information comes in, or to apply heuristics to adjust existing schedules [13, 69]. Many such heuristics have been proposed [101, 91, 38, 98]. As with static pickup and delivery problems, metaheuristics, such as tabu search and simulated annealing [52], have also been applied to the dynamic PDP. Researchers have also developed heuristics to respond to other types of dynamic events, such as cancellations, traffic delays, and accidents [53, 115, 97].

### 8.1.5 Pickup and Delivery Problems with Transfers

Less work has focused on vehicles that transfer items. The idea of transfers is already central to hub and spoke distribution networks, such as mail services and air travel. In these networks, items or passengers are transferred to central hubs, or "transhipment" points, of varying granularities before being brought to their final destination. These hub and spoke networks are effective for networks with large amounts of traffic, but less effective in networks that make fewer deliveries, in which the cost of transporting only a few items to and from the hubs may not be justified. We speculate that furthermore, the efficiency of hub and spoke networks could be improved for large problem sizes by planning for each item individually; however, this has not occurred due to the underlying computational challenges. See [24] for a more detailed overview of hub and spoke networks in the airline industry.

Algorithms have been developed to find the optimal solution [34] and heuristics [80] for the PDP with fixed transshipment points, and for the capacitated PDP with time windows and a single transshipment point [82]. One of the biggest differences between the prior work and the PDP-T heuristics we present in this thesis in Chapters 5 and 6 is that we consider *multiple* transfers that can occur at *any* location. We are not limited to a small set of pre-selected transfer points, but determine transfer points on the fly geometrically depending on the pickup and delivery

locations. Furthermore, with multiple transfers, we can find better solutions to certain problems than we could with only a single transfer, and potentially deliver items that we would be unable to deliver with a single transfer.

In [110], transfers at any pickup or delivery location are considered in the construction of heuristics for the PDP with time windows. Another heuristic was proposed for the case where transfers can occur anywhere [103]. Transfers at arbitrary locations have also been allowed in [22] for the dynamic PDP-T. This heuristic, unlike much of our own work on the online PDP-T, does not consider time windows, capacity constraints, or any other constraints.

In [48], a randomized $O(\log^3 n)$ approximation, where $n$ is the number of pickup and delivery vertices, is given for the preemptive capacitated PDP, where objects may be dropped off at pickup points and retrieved by other vehicles later. We require both vehicles to be present at the transfer locations concurrently for the PDP-T.

Masson et. al. have recently introduced a VLNS-based algorithm to solve PDPs with transfers, along with a helper algorithm to determine the feasibility of a request insertion with transfers [77, 76, 78]. This is similar to our VLNS-T algorithm presented in Chapter 7, but our work differs in that we allow a single item to be transferred multiple times, consider additional constraints such as maximum item transportation times and maximum route durations, allow transfers at any location, consider a cost for transfers, and require both vehicles to be present for a transfer.

## 8.2 Related Robotics Research

Next, we discuss related work from the robotics community. This includes research in robots to do pickup and delivery tasks, task allocation, ridesharing, and robot rendezevous.

### 8.2.1 Pickup and Delivery Robots

Previous researchers have deployed individual robots to pick up and deliver items. In 1994, the HelpMate courier robots transported items within hospitals [42].

Later robots, such as Xavier, allowed users to request tasks over the internet [106, 105]. Other robots soon followed in allowing users to request tasks over the web [104, 49, 100, 102]. Likewise, users can request tasks for our CoBot robots over the web, including pickup and delivery tasks. The CoBots differ from these previous efforts in the scale and length of their deployment: we have deployed the robots for years in a multi-story building, and our system includes multiple robots. Xavier was only a single robot, and did not consider time in its plans. Additionally, none of these previous deployments have transfered items. We believe we are the first to form

schedules with transfers and execute these schedules on robots.

### 8.2.2 Task Allocation

A well-studied field of research closely related to this thesis is robot task allocation, in which tasks must be assigned to robots to maximize a utility function. However, approaches to robot task allocation typically assume that tasks are independent [45]. In the PDP, the tasks are not independent, as the cost of a schedule is highly dependent on the ordering of the tasks and the distances that must be travelled between tasks. Introducing transfers makes the tasks even more interdependent, as the insertion of transfer points depends on the ordering of the rest of the schedule.

Since task allocation is strongly $\mathcal{NP}$-hard, a common approach is to greedily assign tasks to robots as the robots become available [46]. Another popular approach is a free-market based auction system, where the mobile robots place bids on tasks based on the cost to accomplish them [118]. Another approach, common in RoboCup soccer, is complete sharing of information and assignment of roles based on a predefined plan [23].

Researchers have also studied dynamic task allocation problems, where the tasks are not known beforehand [70] and where the robots must respond to failures and environmental changes [86, 21].

### 8.2.3 Ridesharing

With the advent of autonomous cars, researchers have begun to investigate systems to plan for ridesharing. The idea of ridesharing is that drivers, going about their ordinary driving, will offer rides to passengers traveling nearby to help offset their fuel costs. Systems have been built such that drivers and passengers input their destinations on their cell phones, and software helps match passengers with drivers [26, 8, 43]. The ridesharing problem is an instance of the PDP, with the additional property that drivers have their own destinations, just like passengers do.

Limited prior work has been done on the ridesharing problem without transfers. The most common approach is the use of auctions to assign passengers to vehicles. In [1] a distributed, multi-agent rideshare system is presented where an auction assigns passengers to vehicles. Later work has focused on making auctions that are incentive compatible and encourage users to negotiate truthfully. This work attempts to not only minimize the vehicular miles travelled (VMT), but also to maximize the probability of successful rideshares by considering user preferences. However, currently this work is restricted to assigning a single rider per driver [64]. In [61], passengers are assigned to vehicles with a set cover approximation algorithm, and the mechanism

design to construct a fair payment system is considered. A third approach to dynamic ridesharing is presented in [56]: a combination genetic algorithm and insertion heuristic, which considers the problem when passengers have time windows.

The work in this thesis can be applied to the problem of ridesharing when the vehicles can transfer passengers, since ridesharing is a PDP, and we allow vehicles to have their own destinations. To the best of our knowledge, the only researchers which has considered the problem of dynamic ridesharing with transfers is [55]. However, this work lies in finding routes for a single passenger while optimizing multiple objectives: cost, time, and the number of drivers in the route. An evolutionary algorithm is presented to plan multi-hop routes with multiple objectives. In this thesis, we form full schedules for multiple passengers with transfers, rather than a route for an individual passenger. The cost of the passengers' routes are highly co-dependent when the passengers share vehicles.

### 8.2.4 Robot Rendezvous

The robot rendezvous problem is to find a rendezvous point for a set of robots that minimizes the total distance travelled to reach it. Although the idea of the optimal meeting point is quite simple, it is difficult to compute. In fact, it has been shown that no closed form solution exists [117]. However, numerous algorithms have been developed to find the optimal meeting point with gradient descent and other techniques, including a near-optimal solution for general maps [117]. Other researchers have devised algorithms to find rendezvous locations given a schedule of meetings [3] or for a set of worker robots to rendezvous with a single tanker robot [72]. The robot rendezvous problem has also been studied for two robots in an unknown environment [96].

This thesis focuses on forming a schedule for robots to transfer items, while the robot rendezvous research finds only a meeting point for multiple robots. However, this prior work in robot rendezvous could be used as a subcomponent in forming the schedule. We do not do so since these robot rendezvous techniques are computationally expensive, and we opt to use inexpensive heuristics instead.

### 8.2.5 Distributed Algorithms

A large community of researchers has been studying how to build distributed algorithms. Applications of distributed algorithms include multi-robot coverage [9], exploration [118], formation control [6], and task allocation [68, 86], and among the most common approaches are biologically inspired local control rules [68] and multi-agent auctions [39, 65].

For scheduling problems specifically, distributed auctions have been applied [114]. Another

approach is to replan for individual agents but with added constraints to maintain commitments between multiple agents [109]. Multi-agent Simple Temporal Networks have also been developed for distributed settings to maintain time constraints between multiple agents [20].

In this thesis, we presented an auction algorithm which could be implemented in a distributed manner to form schedules with transfers. We believe this is the first distributed algorithm which specifically plans schedules with transfers. Additionally, we discussed distributed algorithms to reschedule from shared information in Ch. 6.

# Chapter 9

# Conclusion

We review the major scientific contributions of this thesis before discussing promising directions for future research.

## 9.1 Contributions

The concrete scientific contributions of this thesis include:

- We formally define the Pickup and Delivery Problem with Transfers (PDP-T) and the properties of a valid PDP-T schedule. We introduce and categorize simplified PDP-T variants, including the Pickup and Single Delivery Problem with Transfers (PSDP-T), and the No Times Pickup and Delivery Problem with Transfers (nTPDP-T).
- We analyze the potential benefits of transfers, and prove that when minimizing total distance travelled with minimal constraints, transfers can reduce the objective cost by at most a factor of two. We show that with additional constraints, transfers may enable additional items to be delivered.
- We implement online task requests, scheduling and execution on the CoBot robots. Users can interrupt the CoBot robots as they move to request new tasks either in person on a touch screen or via voice, or by visiting a website. The tasks are scheduled on the server with an MIP (or a novel scheduling algorithm with transfers) and executed by the robots. The execution of schedules can be monitored remotely via the web. We present results from a study where users schedule tasks on the robots for two weeks in a multi-story building.
- We introduce the CreBot robots, which autonomously transfer items between each other. The CoBots can transfer items as well but rely on help from humans.
- We contribute a provably two-approximate heuristic and a metaheuristic that builds on it

for the PSDP-T. We prove a bound for the improvement from allowing transfers for the PSDP-T at any location versus allowing them only at pickup locations. We illustrate the results of our algorithms in simulation and on the CoBots and CreBots.

- We propose three heuristics for the nTPDP-T: a greedy insertion heuristic, an auction insertion heuristic, and an insertion heuristic based on finding the shortest path in a graph. We compare the three heuristics on simulated problems in the Euclidean plane, and on maps of San Francisco with problems generated from real world taxi data.
- We create an auction-based heuristic to schedule PDP-Ts online. The heuristic reschedules in response to new requests, delays, and failures, and could be implemented in a distributed manner. The algorithms plans with transfers to satisfy either hard or soft time windows. We implement the online algorithm on the CoBot robots, and demonstrate the robots rescheduling in response to delays and robot failures.
- We introduce the novel idea of rescheduling based on shared information from other robots. We examine what information to share, and how rescheduling could be implemented in a distributed setting. We demonstrate rescheduling based on shared observations of closed doors and blocked hallways on the CoBot robots.
- We contribute the VLNS-T algorithm to solve the general PDP-T, with multiple constraints, including time windows, capacities, maximum vehicle route durations, and maximum item transport times. We show that VLNS-T outperforms state of the art PDP algorithms on benchmark problems and on problems generated from real-world taxi data on maps of New York City.

## 9.2 Future Research Directions

There are many promising directions for future research stemming from this thesis. We discuss a few briefly:

- **Further Optimization for VLNS-T.** The VLNS-T algorithm improves upon VLNS solutions, but is computationally expensive. One area for future research is in creating heuristics and optimizations to eliminate transfer points from the search space and increase the speed of VLNS-T so that larger problems can be solved more quickly.
- **Comparison to Hub and Spoke Approaches.** If PDP-T algorithms could be run on scales where hub and spoke approaches are practical, it would be useful to learn how much of an improvement planning for individual packages with transfers can be over using pre-defined fixed routes to hubs for different problem sizes.

- **Optimizing for Makespan and Delivery Time Objectives.** This thesis mainly focused on minimizing the total distance travelled by the vehicles and the total time the vehicles were operational. How would we form schedules to minimize the makespan, the maximum time any vehicle is operational and the time before all tasks are completed?
- **Divisible resources.** How could we plan to deliver divisible resources with transfers? For example, delivering water to put out fires or food to deliver to refugees that could be split between multiple vehicles?
- **Online destinations.** Taxis that pick up passengers from the curb do not know their destinations until the passengers enter the vehicle. How would this change the way plans are made?
- **Predicting and Preparing for Online Requests.** Given sufficient data for learning, it could be possible to learn a distribution of where new requests are expected to occur for the online PDP-T. Given this distribution, could the vehicles plan to prepare for expected new requests in order to fulfill them faster?
- **Geometric constraints.** Maps often have distinctive geometric features, such as bottleneck bridges and elevators in the Gates-Hillman Center. Could these geometric constraints be exploited to form better plans or to form plans more quickly?
- **Evaluating problem difficulty.** When delivering all the items to the same location, we can find a two-approximate solution in polynomial time. Are there other special cases with similar properties? How can we evaluate the difficulty of a particular problem instance?
- **Planning under uncertainty.** Items may not be ready for delivery when expected, or delays may occur. Rather than replanning to avoid delays, as we currently, could a schedule be planned that is resilient to these failures?
- **Probabilistic Travel Times.** Most current approaches to the PDP assume that the travel time between all destinations is known a priori. However, this assumption is unrealistic, as in most domains there is a variance in travel times. How could this distribution be accounted for in scheduling?
- **Additional domains.** A few other possible domains for this research could include distribution methods for the shipping industry, planning flights for airlines, and devising means of distributing water and supplies for rescue agencies. How effective would transfers be in these domains?

## 9.3 Concluding Remarks

In this thesis, we have introduced several algorithms which plan schedules with transfers for pickup and delivery problems. Transferring items makes lower cost schedules which deliver more items possible. The work presented has broad applications in robotics, logistics and transportation domains. We have evaluated our algorithms with transfers on a range of benchmark problems and transportation problems, and deployed our scheduling algorithms on real robots.

# Bibliography


[1] S. Abdel-Naby, S. Fante, and P. Giorgini. Auctions negotiation for mobile rideshare service. In *Proceedings of the International Conference on Pervasive Computing and Applications (ICPCA)*, pages 225–230, 2007. 68, 126

[2] F. Alfredo Tang Montané and R. Galvao. A tabu search algorithm for the vehicle routing problem with simultaneous pick-up and delivery service. *Computers & Operations Research*, 33(3):595–619, 2006. 123

[3] K. Alton and I. Mitchell. Efficient dynamic programming for optimal multi-location robot rendezvous. In *IEEE Conference on Decision and Control*, pages 2794–2799, 2008. 127

[4] S. Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *Journal of the ACM (JACM)*, 45(5):753–782, 1998. 122

[5] B. Baker and M. Ayechew. A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, 30(5):787–800, 2003. 123

[6] T. Balch and R. Arkin. Behavior-based formation control for multirobot teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, 1998. 127

[7] N. Bansal, A. Blum, S. Chawla, and A. Meyerson. Approximation algorithms for deadline-TSP and vehicle routing with time-windows. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 166–174, 2004. 124

[8] F. Barringer. Need a ride? There are apps for that. *The New York Times*, January 20 2011. http://green.blogs.nytimes.com/2011/01/20/need-a-ride-theres-an-app-for-that/. 126

[9] M. Batalin and G. Sukhatme. Spreading out: A local approach to multi-robot coverage. In *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems (DARS)*, pages 373–382, 2002. 127

[10] T. Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209–219, 2006. 122

[11] R. Bent and P. Van Hentenryck. A two-stage hybrid local search for the vehicle routing

problem with time windows. *Transportation Science*, 38(4):515–530, 2004. 124

[12] G. Berbeglia, J. Cordeau, I. Gribkovskaia, and G. Laporte. Static pickup and delivery problems: A classification scheme and survey. *Top*, 15(1):1–31, 2007. 123

[13] G. Berbeglia, J. Cordeau, and G. Laporte. Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202(1):8–15, 2010. 124

[14] M. Berkelaar, K. Eikland, and P. Notebaert. lpsolve, 2012. http://lpsolve.sourceforge.net. 39

[15] B. Bhattacharya and Y. Hu. Approximation algorithms for the multi-vehicle scheduling problem. *Algorithms and Computation*, pages 192–205, 2010. 124

[16] J. Biswas, B. Coltin, and M. Veloso. Corrective gradient refinement for mobile robot localization. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 73–78, 2011. 30

[17] J. Biswas and M. Veloso. Depth camera based indoor mobile robot localization and navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1697–1702, 2012. 30

[18] J. Biswas and M. Veloso. Multi-sensor mobile robot localization for diverse environments. *Proceedings of RoboCup 2013: Robot Soccer World Cup XVII*, 2013. 30

[19] J. Biswas and M. M. Veloso. Localization and navigation of the cobots over long-term deployments. *The International Journal of Robotics Research*, 32(14):1679–1694, 2013. 30

[20] J. C. Boerkoel Jr, L. R. Planken, R. J. Wilcox, and J. A. Shah. Distributed algorithms for incrementally maintaining multiagent simple temporal networks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 11–19, 2013. 128

[21] S. Botelho and R. Alami. M+: A scheme for multi-robot cooperation through negotiated task allocation and achievement. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1234–1239, 1999. 126

[22] P. Bouros, D. Sacharidis, T. Dalamagas, and T. Sellis. Dynamic pickup and delivery with transfers. In D. Pfoser, Y. Tao, K. Mouratidis, M. Nascimento, M. Mokbel, S. Shekhar, and Y. Huang, editors, *Advances in Spatial and Temporal Databases*, volume 6849 of *Lecture Notes in Computer Science*, pages 112–129. Springer, 2011. 125

[23] B. Browning, J. Bruce, M. Bowling, and M. Veloso. STP: Skills, tactics, and plays for multi-robot control in adversarial environments. *Proceedings of the Institution of Mechan-*

*ical Engineers, Part I: Journal of Systems and Control Engineering*, 219(1):33–52, 2005. 126

[24] D. Bryan and M. O'Kelly. Hub-and-spoke networks in air transportation: An analytical review. *Journal of Regional Science*, 39(2):275–295, 2002. 3, 124

[25] B. Bullnheimer, R. Hartl, and C. Strauss. An improved ant system algorithm for the vehicle routing problem. *Annals of Operations Research*, 89:319–328, 1999. 123

[26] N. Chan and S. Shaheen. Ridesharing in north america: Past, present, and future. *Transport Reviews*, 32(1):93–112, 2012. 126

[27] M. Charikar and B. Raghavachari. The finite capacity dial-a-ride problem. In *Proceedings of 39th Annual Symposium on Foundations of Computer Science, 1998*, pages 458–467, 1998. 124

[28] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Grad. School of Industrial Administration, Carnegie Mellon University, 1976. 122

[29] B. Coltin, J. Biswas, D. Pomerleau, and M. Veloso. Effective semi-autonomous telepresence. *RoboCup 2011: Robot Soccer World Cup XV*, pages 365–376, 2012. 32, 86

[30] B. Coltin, S. Liemhetcharat, C. Meriçli, J. Tay, and M. Veloso. Multi-humanoid world modeling in standard platform robot soccer. In *Proceedings of the IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 424–429, 2010. 97

[31] B. Coltin and M. Veloso. Optimizing for transfers in a multi-vehicle collection and delivery problem. In *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems (DARS)*, 2012. 51, 60

[32] B. Coltin, M. Veloso, and R. Ventura. Dynamic user task scheduling for mobile robots. In *Proceedings of the Workshop on Automated Action Planning for Autonomous Mobile Robots, AAAI*, 2011. 29

[33] J.-F. Cordeau and G. Laporte. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, 37(6):579–594, 2003. xiii, 5, 113, 117, 118

[34] C. Cortés, M. Matamala, and C. Contardo. The pickup and delivery problem with transfers: Formulation and a branch-and-cut solution method. *European Journal of Operational Research*, 200(3):711–724, 2010. 3, 124

[35] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1):61–95, 1991. 89

[36] J. Dethloff. Vehicle routing and reverse logistics: the vehicle routing problem with simultaneous delivery and pick-up. *OR Spectrum*, 23(1):79–96, 2001. 123

[37] J. Dethloff et al. Relation between vehicle routing problems: an insertion heuristic for the vehicle routing problem with simultaneous delivery and pick-up applied to the vehicle routing problem with backhauls. *Journal of the Operational Research Society*, 53(1):115–118, 2002. 123

[38] R. Dial. Autonomous dial-a-ride transit introductory overview. *Transportation Research Part C: Emerging Technologies*, 3(5):261–275, 1995. 124

[39] M. Dias, R. Zlot, N. Kalra, and A. Stentz. Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94(7):1257–1270, 2006. 127

[40] C. Duhamel, J. Potvin, and J. Rousseau. A tabu search heuristic for the vehicle routing problem with backhauls and time windows. *Transportation Science*, 31(1):49–59, 1997. 122, 123

[41] B. Eksioglu, A. Vural, and A. Reisman. The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, 57(4):1472–1483, 2009. 121

[42] J. Evans. Helpmate: An autonomous mobile robot courier for hospitals. In *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 1695–1700, 1994. 125

[43] C. Farivar. Transportation innovation: How Lyft and SideCar are changing commuting. *Ars Technica*, September 28 2012. http://arstechnica.com/business/2012/09/my-life-as-a-high-tech-part-time-not-quite-taxi-driver/. 5, 126

[44] N. Ferreira, J. Poco, H. T. Vo, J. Freire, and C. T. Silva. Visual exploration of big spatio-temporal urban data: A study of New York City taxi trips. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2149–2158, 2013. 118

[45] B. Gerkey and M. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004. 126

[46] B. P. Gerkey and M. J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9), 2004. 126

[47] M. Goetschalckx and C. Jacobs-Blecha. The vehicle routing problem with backhauls. *European Journal of Operational Research*, 42(1):39–51, 1989. 123

[48] I. Gørtz, V. Nagarajan, and R. Ravi. Minimum makespan multi-vehicle dial-a-ride. *Algorithms-ESA 2009*, pages 540–552, 2009. 125

[49] S. Grange, T. Fong, and C. Baur. Effective vehicle teleoperation on the world wide web. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 2007–2012, 2000. 125

[50] I. Gribkovskaia, Ø. Halskau, G. Laporte, and M. Vlček. General solutions to the single vehicle routing problem with pickups and deliveries. *European Journal of Operational Research*, 180(2):568–584, 2007. 122

[51] A. Gupta, M. Hajiaghayi, V. Nagarajan, and R. Ravi. Dial a ride from k-forest. *ACM Transactions on Algorithms (TALG)*, 6(2):41, 2010. 124

[52] K. Gutenschwager, C. Niklaus, and S. Voß. Dispatching of an electric monorail system: Applying metaheuristics to an online pickup and delivery problem. *Transportation Science*, 38(4):434–446, 2004. 124

[53] A. Haghani and S. Jung. A dynamic vehicle routing problem with time-dependent travel times. *Computers & Operations Research*, 32(11):2959–2986, 2005. 124

[54] G. Hasle and O. Kloster. Industrial vehicle routing. In *Geometric Modelling, Numerical Simulation, and Optimization*, pages 397–435. Springer, 2007. 117

[55] W. Herbawi and M. Weber. Evolutionary multiobjective route planning in dynamic multi-hop ridesharing. In P. Merz and J.-K. Hao, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 6622 of *Lecture Notes in Computer Science*, pages 84–95. Springer Berlin / Heidelberg, 2011. 127

[56] W. Herbawi and M. Weber. A genetic and insertion heuristic algorithm for solving the dynamic ridematching problem with time windows. In *Proceedings of the International Conference on Genetic and Evolutionary Computation*, pages 385–392, 2012. 127

[57] IBM. IBM ILOG CPLEX optimization studio, 2012. http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud/. 39

[58] A. Ivanov and A. Tuzhilin. The Steiner ratio Gilbert–Pollak conjecture is still open. *Algorithmica*, pages 1–3, 2011. 60

[59] S. Jung and A. Haghani. Genetic algorithm for a pickup and delivery problem with time windows. *Transportation Research Record: Journal of the Transportation Research Board*, 1733(-1):1–7, 2000. 124

[60] V. Kachitvichyanukul et al. A particle swarm optimization for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research*, 36(5):1693–1702, 2009. 123

[61] E. Kamar and E. Horvitz. Collaboration and shared plans in the open world: Studies of

ridesharing. In *Proceedings of the International Jont Conference on Artifical Intelligence (IJCAI)*, pages 187–194, 2009. 68, 126

[62] G. Kaminka and I. Frenkel. Integration of coordination mechanisms in the bite multi-robot architecture. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2859–2866, 2007. 97

[63] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671, 1983. 58

[64] A. Kleiner, B. Nebel, and V. Ziparo. A mechanism for dynamic ride sharing based on parallel auctions. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 266–272, 2011. 68, 126

[65] S. Koenig, C. Tovey, M. Lagoudakis, V. Markakis, D. Kempe, P. Keskinocak, A. Kleywegt, A. Meyerson, and S. Jain. The power of sequential single-item auctions for agent coordination. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 1625. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006. 127

[66] T. Kollar, V. Perera, D. Nardi, and M. Veloso. Learning environmental knowledge from task-based human-robot dialog. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 4304–4309, 2013. 34, 43

[67] M. Korein, B. Coltin, and M. Veloso. Scheduling mobile exploration tasks for environment learning. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1255–1256, 2013. 34

[68] M. Krieger, J. Billeter, and L. Keller. Ant-like task allocation and recruitment in cooperative robots. *Nature*, 406(6799):992–995, 2000. 127

[69] A. Larsen and O. Madsen. *The dynamic vehicle routing problem*. PhD thesis, Technical University of Denmark, Department of Informatics and Mathematical Modeling, 2000. 124

[70] K. Lerman, C. Jones, A. Galstyan, and M. Matarić. Analysis of dynamic task allocation in multi-robot systems. *The International Journal of Robotics Research*, 25(3):225–241, 2006. 126

[71] H. Li and A. Lim. A metaheuristic for the pickup and delivery problem with time windows. In *Proceedings of the International Conference on Tools with Artificial Intelligence*, pages 160–167, 2001. 124

[72] Y. Litus, R. Vaughan, and P. Zebrowski. The frugal feeding problem: Energy-efficient,

multi-robot, multi-place rendezvous. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 27–32, 2007. 127

[73] Q. Lu and M. Dessouky. An exact algorithm for the multiple vehicle pickup and delivery problem. *Transportation Science*, 38(4):503–514, 2004. 123

[74] Q. Lu and M. Dessouky. A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows. *European Journal of Operational Research*, 175(2):672–687, 2006. 124

[75] D. Luxen and C. Vetter. Real-time routing with OpenStreetMap data. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS)*, pages 513–516, New York, NY, USA, 2011. ACM. 118

[76] R. Masson, F. Lehuédé, and O. Péton. An adaptive large neighborhood search for the pickup and delivery problem with transfers. *Transportation Science*, 47(3):344–355, 2013. 125

[77] R. Masson, F. Lehuédé, and O. Péton. Efficient feasibility testing for request insertion in the pickup and delivery problem with transfers. *Operations Research Letters*, 41(3):211–215, 2013. 125

[78] R. Masson, F. Lehuédé, and O. Péton. The dial-a-ride problem with transfers. *Computers & Operations Research*, 41:12–23, 2014. 3, 125

[79] J. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM J. Comput.*, 28(4):1298–1309, 1999. 122

[80] S. Mitrovic-Minic and G. Laporte. The pickup and delivery problem with time windows and transshipment. *Information Systems and Operational Research*, 44(3):217–228, 2006. 3, 124

[81] G. Nagy and S. Salhi. Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries. *European Journal of Operational Research*, 162(1):126–141, 2005. 122, 123

[82] Y. Nakao and H. Nagamochi. Worst case analysis for a pickup and delivery problem with single transfer. *Numerical Optimization Methods, Theory and Applications*, 1584:142–148, 2008. 3, 124

[83] W. Nanry and J. Wesley Barnes. Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Research Part B: Methodological*, 34(2):107–121, 2000. 124

[84] OpenStreetMap. OpenStreetMap, 2012. http://openstreetmap.org. 81

[85] I. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41(4):421–451, 1993. 123

[86] L. Parker. Alliance: An architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998. 126, 127

[87] S. Parragh, K. Doerner, and R. Hartl. A survey on pickup and delivery problems: Part I, transportation between customers and depot. *Journal fur Betriebswirtschaft*, 58(1):21–51, 2008. 121, 122, 123

[88] S. Parragh, K. Doerner, and R. Hartl. A survey on pickup and delivery problems: Part II, transportation between pickup and delivery locations. *Journal fur Betriebswirtschaft*, 58(2):81–117, 2008. 121, 122

[89] M. Piorkowski, N. Sarafijanovoc-Djukic, and M. Grossglauser. A Parsimonious Model of Mobile Partitioned Networks with Clustering. In *Proceedings of the International Conference on COMmunication Systems and NETworkS (COMSNETS)*, January 2009. 81

[90] M. Polacek, R. F. Hartl, K. Doerner, and M. Reimann. A variable neighborhood search for the multi depot vehicle routing problem with time windows. *Journal of Heuristics*, 10(6):613–627, 2004. 117

[91] D. Popken. Controlling order circuity in pickup and delivery problems. *Transportation Research Part E: Logistics and Transportation Review*, 42(5):431–443, 2006. 124

[92] S. Ropke, J. Cordeau, and G. Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49(4):258–272, 2007. 123

[93] S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006. xiii, 105, 106, 107, 108, 109, 117, 118, 124

[94] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, II. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6(3):563–581, 1977. 122

[95] S. L. Rosenthal. *Human-Centered Planning for Effective Task Autonomy*. PhD thesis, Carnegie Mellon University, 2012. 30

[96] N. Roy and G. Dudek. Collaborative robot exploration and rendezvous: Algorithms, performance bounds and observations. *Autonomous Robots*, 11(2):117–136, 2001. 127

[97] Z. Rubinstein and S. Smith. Dynamic management of paratransit vehicle schedules. In *Proceedings of the International Workshop on Scheduling and Planning Applications*,

June 2011. 124

[98] Z. Rubinstein, S. Smith, and L. Barbulescu. Incremental management of oversubscribed vehicle schedules in dynamic dial-a-ride problems. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012. 124

[99] M. Samadi, T. Kollar, and M. M. Veloso. Using the web to interactively learn to find objects. In *Proceedings of AAAI*, 2012. 32

[100] P. Saucy and F. Mondada. KhepOnTheWeb: open access to a mobile robot on the internet. *IEEE Robotics Automation Magazine*, 7(1):41–47, March 2000. 125

[101] M. Savelsbergh and M. Sol. Drive: Dynamic routing of independent vehicles. *Operations Research*, 46(4):474–490, 1998. 124

[102] D. Schulz, W. Burgard, D. Fox, S. Thrun, and A. Cremers. Web interfaces for mobile robots in public places. *IEEE Robotics Automation Magazine*, 7(1):48–56, March 2000. 125

[103] J. Shang and C. Cuff. Multicriteria pickup and delivery problem with transfer opportunity. *Computers & Industrial Engineering*, 30(4):631–645, 1996. 125

[104] R. Siegwart and P. Saucy. Interacting mobile robots on the web. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1999. 125

[105] R. Simmons, J. Fernandez, R. Goodwin, S. Koenig, and J. O'Sullivan. Lessons learned from Xavier. *IEEE Robotics Automation Magazine*, 7(2):33–39, June 2000. 125

[106] R. Simmons, R. Goodwin, K. Z. Haigh, S. Koenig, and J. O'Sullivan. A layered architecture for office delivery robots. In *Proceedings of the International Conference on Autonomous Agents*, pages 245–252, 1997. 125

[107] Y. Sun, B. Coltin, and M. Veloso. Interruptible autonomy: Towards dialog-based robot task management. In *Intelligent Robotic Systems Workshop, AAAI*, 2013. 43

[108] H. Takahashi and A. Matsuyama. An approximate solution for the Steiner problem in graphs. *Math. Japonica*, 24(6):573–577, 1980. 60

[109] K. Talamadupula, D. E. Smith, W. Cushing, and S. Kambhampati. A theory of intra-agent replanning. In *Proceedings of the ICAPS 2013 Workshop on Distributed and Multi-Agent Planning*, 2013. 128

[110] S. Thangiah, A. Fergany, and S. Awan. Real-time split-delivery pickup and delivery time window problems with transfers. *Central European Journal of Operations Research*, 15(4):329–349, 2007. 125

[111] The pgRouting Project. pgRouting Project, 2012. http://pgrouting.org. 81

[112] P. Toth and D. Vigo. Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics*, 123(1):487–512, 2002. 122

[113] P. Toth and D. Vigo. *The vehicle routing problem*, volume 9. Soc. for Industrial Mathematics, 2002. 121

[114] M. Wellman, W. Walsh, P. Wurman, and J. MacKie-Mason. Auction protocols for decentralized scheduling. *Games and Economic Behavior*, 35(1):271–303, 2001. 127

[115] Z. Xiang, C. Chu, and H. Chen. The study of a dynamic dial-a-ride problem under time-dependent and stochastic environments. *European Journal of Operational Research*, 185(2):534–551, 2008. 124

[116] H. Xu, Z. Chen, S. Rajagopal, and S. Arunapuram. Solving a practical pickup and delivery problem. *Transportation Science*, 37(3):347–364, 2003. 124

[117] P. Zebrowski, Y. Litus, and R. Vaughan. Energy efficient robot rendezvous. In *Proceedings of the Canadian Conference on Computer and Robot Vision (CRV)*, pages 139–148, 2007. 70, 127

[118] R. Zlot, A. Stentz, M. Dias, and S. Thayer. Multi-robot exploration controlled by a market economy. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, pages 3016–3023, 2002. 126, 127