This document is downloaded from DR-NTU (https://dr.ntu.edu.sg)
Nanyang Technological University, Singapore.

Energy-efficient automated material handling systems

Fang, Zhou

2016

Fang, Z. (2016). Energy-efficient automated material handling systems. Doctoral thesis, Nanyang Technological University, Singapore.

https://hdl.handle.net/10356/65952

https://doi.org/10.32657/10356/65952

Downloaded on 24 Aug 2022 20:34:33 SGT

Nanyang Technological University

Energy-Efficient Automated Material Handling Systems

Ph.D Thesis

By

Fang Zhou

Supervisor: Asst. Prof. Mao Jianfeng

School of Mechanical and Aerospace Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of
Doctor of Philosophy

July, 2015

Abstract

The cost for operating equipment in Automated Material Handling Systems (AMHS)– like manufacturing facilities and distribution centers– is an ongoing cost that requires a substantial commitment in electrical power. Many facilities are extremely large, depending on the complexity of the facilities conveyor system, it could account for as much as 50% of a facilitys electrical load [1]. Increased fuel costs have pressured manufacturers to develop initiatives to find more efficient management of energy usage. In addition, emerging environmental regulations have forced companies to develop ways to reduce emissions without compromising the quality of their products. This motivates us to look into energy issues in AMHSs. However, the majority studies on the vehicle routing and dispatching problems only focused on the completion time or traveling distance issues, few of them paid energy issues any attention.

In this thesis, we investigate a minimum-energy consumption routing problem of a stacker crane vehicle, which serves a multi-story storage and retrieval system (AS/RS) inside an air cargo terminal. We name this problem Energy-efficient Stacker Crane Problem (EESCP). This problem can be formulated as a Stacker Crane Problem on a two-dimensional (2D) grid network with a cost function in L_1 norm. First, we prove this more specific problem to be NP-Complete. Still, due to the cost function in Manhattan norm, we are able to identify a special subset of 2D instances with certain arc patterns, to be polynomial-time solvable with an algorithm developed for one-dimensional problem, a.k.a Stacker Crane Problem on paths. For problem instances with more general arc patterns, we have

proposed a new exact formulation the scale of which is fixed by the underlying grid network. We have also developed two polynomial-time approximation algorithms with the grid network taken into account. One is asymptotically optimal and has the time-complexity that grows linearly with the number of requests given by the problem instance; the other has a bounded time-complexity and performs better for instances with smaller arc lengths. These two algorithms together provide an improved 5/3 theoretical worst-case bound than the 9/5 in the work of Frederickson, Hecht et al. [2].

Furthermore, we have adapted our static algorithm to the dynamic environment using a rolling-horizon approach. The dynamic algorithm has two parameters, look-ahead horizon and decision point. The algorithm performance under various combinations of these two parameters is tested via simulations. Finally, we propose an exact formulation for energy-efficient multi-capacity problem and three heuristics. The exact formulation is bilinear and we have reformulated it into the linear form. The performances of the heuristics are compared against the exact solutions.

Acknowledgments

Foremost I would like to express my sincere gratitude towards my supervisor Prof. Mao Jianfeng for his guidance and support. Throughout my four years of candidature, he has provided me with vision, showered me with encouragement and sound advice on both professional and personal growth, patiently trained me on independent research. Also, I am extremely grateful for the scholarship and education provided from Nanyang Technological University. The efficient and convenient campus life that the university has strived to provide has given me the ease of mind to focus on my studies. Furthermore, I wish to thank my friends all over the world for the friendship and companionship as well as good times. Last but not the least, I would like to thank my family and loved ones for the unconditional love and support through thick and thin. The work presented in this thesis would not have been possible without any single one of you.

Contents

Abstract i
Acknowledgments iii
List of Figures vi
List of Tables viii
List of Notations x

1 Introduction **1**
1.1 Automated Material Handling System 1
1.2 Automated Storage and Retrieval System 2
1.2.1 Definition of AS/RS 2
1.2.2 Types and Variations of AS/RS 5
1.3 Energy Efficiency in AMHS 5
1.4 Aims, Scope and Contributions 8
1.5 Organization of the Thesis 9

2 Related Work **11**
2.1 Energy-efficiency in Material Handling Systems 11
2.2 Stacker Crane Problem 12
2.3 Closely Related Combinatorial Problems 21
2.4 Summary 24

3 Pseudo-2D Problems **26**
3.1 Problem Statement 26
3.2 NP-Completeness of EESCP 32
3.3 Conditions for Free-Permutation 33
3.3.1 Proof for Theorem 3.2 41

4 Nodes Formulation and GRID Algorithm **48**
4.1 A New Exact Formulation 50
4.2 GRID-L 54
4.2.1 Minimum Cost Flow Formulation 55
4.2.2 Transportation Problem Formulation 58
4.2.3 1D Bound on 2D Problems 61
4.3 Complexity of GRID-L 63
4.4 Asymptotic Optimality of GRID-L 64
4.5 GRID-S 65
4.6 Complexity of GRID-S 68
4.7 Better Theoretical Bound 70
4.8 Numerical Results 71

5 Dynamic and Multi-Capacity Problem **79**
5.1 Dynamic Problems 79
5.1.1 Look-ahead Horizon 80
5.1.2 Decision Point 82
5.2 Multi-Capacity Problems 86
5.2.1 Multi-Capacity Exact Formulation 87
5.2.2 Solution Approach 89
5.2.3 *2AS1* 90
5.2.4 *PPDD* 90
5.2.5 *BUBBLE* 92
5.2.6 Simulation Results 93

6 Conclusions and Future Work **98**
6.1 Conclusions 98
6.2 Future Work 100
6.2.1 Integrate Time Information 100
6.2.2 Collaboration Between Multiple Vehicles 100
6.2.3 Further into the Multi-capacity Problem 101
6.2.4 Stochastic Framework 101

References **104**

List of Figures

1.1 Modern AMHS and AS/RS systems 3
1.2 Generic structure and principal constituents of an AS/RS 5
1.3 Various system concepts for AS/RSs, modified after Roodbergen and Vis 2009 [3] 6
1.4 Global Temperature, Primary energy consumption and Carbon dioxide concentration since 1965(Günthner, Tilke et al. 2010 [4]) 7
1.5 Electric Power consumption in the German industry (Kaschenz, Albert et al. 2009 [5]) 7

3.1 Two examples of storage and retrieval system in air cargo terminals 27
3.2 The EESCP instance with its underlying grid 28
3.3 The difference in energy cost when optimized in different metrics . . 31
3.4 The difference in traveling distance when optimized in different metrics 31
3.5 Problems with two different structures can be sovled with the same algorithm 34
3.6 (a) Degree-balancing and (b) connecting the graph to make an eulerian graph that contains the optimal tour 35
3.7 The 2D instance and its two corresponding 1D problem instances . 36
3.8 In circuit printing problem, the optimal solution to vertical subproblem yeilds full permutation of the request arcs 37
3.9 Some similar examples to Ball and Magazine's circuit printing problem and their free-permutation subproblems 38
3.10 Another example of "free-permutation" 1D problem 38
3.11 A demonstration of Lemma 3.3 39
3.12 A simple demonstration of Theorem 3.2 41
3.13 When all arcs are unidirectional, if they are of free-permutation, they must overlap 43
3.14 In Scenario B1), Condition 4) can be deduced 45

3.15 Scenario B2), when $n_L = 1$, Condition 1) can be deduced 46
3.16 In Scenario B2), when $n_L \geq 2$, Condition 1) can be deduced 47

4.1 No augmenting arcs from a vertex in D in the optimal eulerian graph, due to triangular inequality 53
4.2 A 3×3 grid limit decision variables number to 24 58
4.3 T_H^*,T_V^* provide extra information for the 2D problem 62
4.4 T_H^*,T_V^* can be translated into constraints to eliminate some obvious sub-tours in the 2D algorithm 63
4.5 An instance with 6 requests and 3 connected components 66
4.6 How K grows with the number of requests at different level of N . . 69
4.7 The computation time of solving ATSP Formulation and Nodes Formulation 72
4.8 The optimality gap of GRID-L narrows when the number of requests goes up 73
4.9 In a small-length arcs setting, GRID-S out performs both GRID-L and *SMALLARCS* 74
4.10 The computation time of GRID-L grows linearly, can be used instances with large M 75
4.11 The computation time of GRID-L and SPLICE 76
4.12 The computation time of GRID-S 77
4.13 The computation time of GRID-S has a high variance 78

5.1 The performance of Dynamic GRID improves as larger look-ahead horizon is adopted 81
5.2 On average, smaller decision points provide smaller gaps 82
5.3 The combined effect of look-ahead horizon and decision point 84
5.4 Relative performance of Nearest-Neighbour against Dynamic GRID 85
5.5 The relative performance between Nearest-Neighbour and Dynamic GRID has a high variance 86
5.6 The optimality gaps of the three heuristics 95
5.7 The relative performances of the three heuristics 96
5.8 The computation time of PPDD and BUBBLE 97

6.1 The deterministic equivalent of an uncertain request 102

List of Tables

2.1 The state of art for static unit-capacity SCPs 18
2.2 The state of art for static multi-capacity SCPs 18

3.1 The difference in energy cost when optimized in different metrics (%) 30
3.2 The difference in traveling distance when optimized in different metrics (%) 32
3.3 Six conditions in Theorem 3.3 40

4.1 Algorithm: GRID-L(GF) 57
4.2 Algorithm: GRID-L(TF) 60
4.3 The time complexity of GRID-L 64
4.4 Algorithm: GRID-S 67
4.5 The time complexity of GRID-S 68
4.6 The computation time of solving ATSP and Nodes formulation (s) 72
4.7 Maximum optimality gap at different instance size for GRID-L(GF) and (TF), over 100 instances (%) 73
4.8 The optimal gaps of *SMALLARCS*, GRID-L, and GRID-S, over 50 instances (%) 74
4.9 The computation time of GRID-L and SPLICE(s) 76

5.1 The performance of Dynamic GRID with different look-ahead horizon (%) 81
5.2 The performance of Dynamic GRID with different values of decision point (%) 83
5.3 Algorithm: 2AS1 91
5.4 Algorithm: *PPDD* 92
5.5 Algorithm: Multi-capacity Nearest-Neighbour 93
5.6 Algorithm: *BUBBLE* 94

5.7 The relative performances of the *2AS1* and *PPDD* against *BUBBLE* (%) 96

List of Notations

Sets

R	Set of all the requests
A	Set of all the arcs representing the requests
V	Set of all the vertices on the grid
E	Set of all the edges on the grid
S	$S = \{i \in V \mid G_i > 0\}$
D	$D = \{i \in V \mid G_i < 0\}$
U	U is the set of vertices that are visited by at least one arc in A, whose final net in-degree is zero
P	$P = S \cup D \cup U$ is the set of all the vertices that are visited at least once in the instance
NB_i	Set of neighboring cells of cell i
NB_i^H	The horizontal neighboring cells of cell i
NB_i^V	The vertical neighboring cells of cell i
A_1	Set of arcs that represents the requests in a 1D SCP istance
T_s	The set of arcs that forms the trajectory (eulerian graph) of the vehicle if it performs A_1 in sequence s
$P_H(T_s), P_V(T_s)$	The horizontal and vertical projections of T_s. They themselves are 1D eulerian graphs
I_H, I_V	The horizontal and vertical 1D subproblem instances of a 2D instance I;

T_H^*, T_V^*	The optimal eulerian graph obtained by respectively solving I_H and I_V
S_H^*, S_V^*	The respective sets of all the sequences that can be obtained from T_H^* and T_V^*

Parameters

M	The total number of all requests
N_V	The number of rows of the storage rack
N_H	The number columns of the storage rack
N	$N = N_V \times N_H$, is the total number of storage cells on the rack
K	The number of connected components
c_H, c_V	The energy consumption for the stacker crane vehicle to perform a single move horizontally/ vertically when empty-loaded
G_i	The net in-degree of cell i induced by the arcs in A
x_i^O, x_i^D	The respective coordinates of the pickup and drop-off positions of arc $i \in A_1$
n	The number of arcs in A_1
a_{ij}	The number of arcs starting from vertex i to vertex j in A, $\forall i, j \in P$
c_{ij}	The cost of arc (i, j)

Others

$c(s)$	The cost incurred by sequence s;
$c(A)$	The cost incurred by the set of arcs A
C^*	The optimal tour cost for the *EESCP* instance

Chapter 1

Introduction

This chapter provides some background knowledge for this thesis, which includes a general description of Automated Material Handling System (AMHS), one of its most widely adopted equipment, Automated Storage/Retrieval System (AS/RS), and the rising awareness of energy-efficiency. The research aims, scope, contributions and the organization of the thesis are also summarized.

1.1 Automated Material Handling System

Expressed in simple language, material handling is moving and controlling of materials. The current widely used definition of material handling is presented by Tompkins et al. [6] as the function of "providing the right amount of the right material, in the right condition, at the right place, at the right time, in the right position, in the right sequence, and for the right cost, by using the right methods". And as the American Society of Mechanical Engineers(ASME) defines it: "Materials handling is the art and science involving the moving, packaging and storing of substances in any form." Materials handling serves as an essential part of production, but is a process of cost rather than value adding. Inefficient materials handling practices have put many enterprises out of business. Thus it is vital to reduce the cost of material handling to the minimum. And companies

try to stay ahead of their competitors by adopting advanced material handling systems.

A well-designed material handling system attempts to achieve the following [7]:

- Ensure the right quantity of materials delivered at the right place at the right time most economically;
- Cut down indirect labor cost and overall cost;
- Improve the customer service by reducing the response and thereby waiting time;
- Maximize space utilization by proper storage of materials and thereby reduce storage and handling cost;
- Reduce damage of materials during storage and movement;
- Minimize accidents during materials handling.

1.2 Automated Storage and Retrieval System

1.2.1 Definition of AS/RS

Since its introduction in 1950s, AS/RS has been one of the major tools used for warehouse materials handling and inventory control. It plays an essential role in many integrated manufacturing systems and is widely used in automated production and distribution centers. AS/RSs provide many advantages such as improved utilization of time and space, improved efficiency and inventory control at lower cost [8] [9].

In a general sense, AS/RSs can be defined as a combination of equipment and controls, which automatically handle, store and retrieve materials with great speed and accuracy, without direct handling by human workers [9]. Though this

is a rather broad definition, the term AS/RS somehow has come to mean a common type of system consisting of one stacker crane (also referred to as storage and retrieval (S/R) machine), one or multiple parallel aisles of storage racks, input/output stations (I/O stations, also pickup/delivery stations or docks), conveyors, a communication system and a central supervisory computer (Figure 1.1).

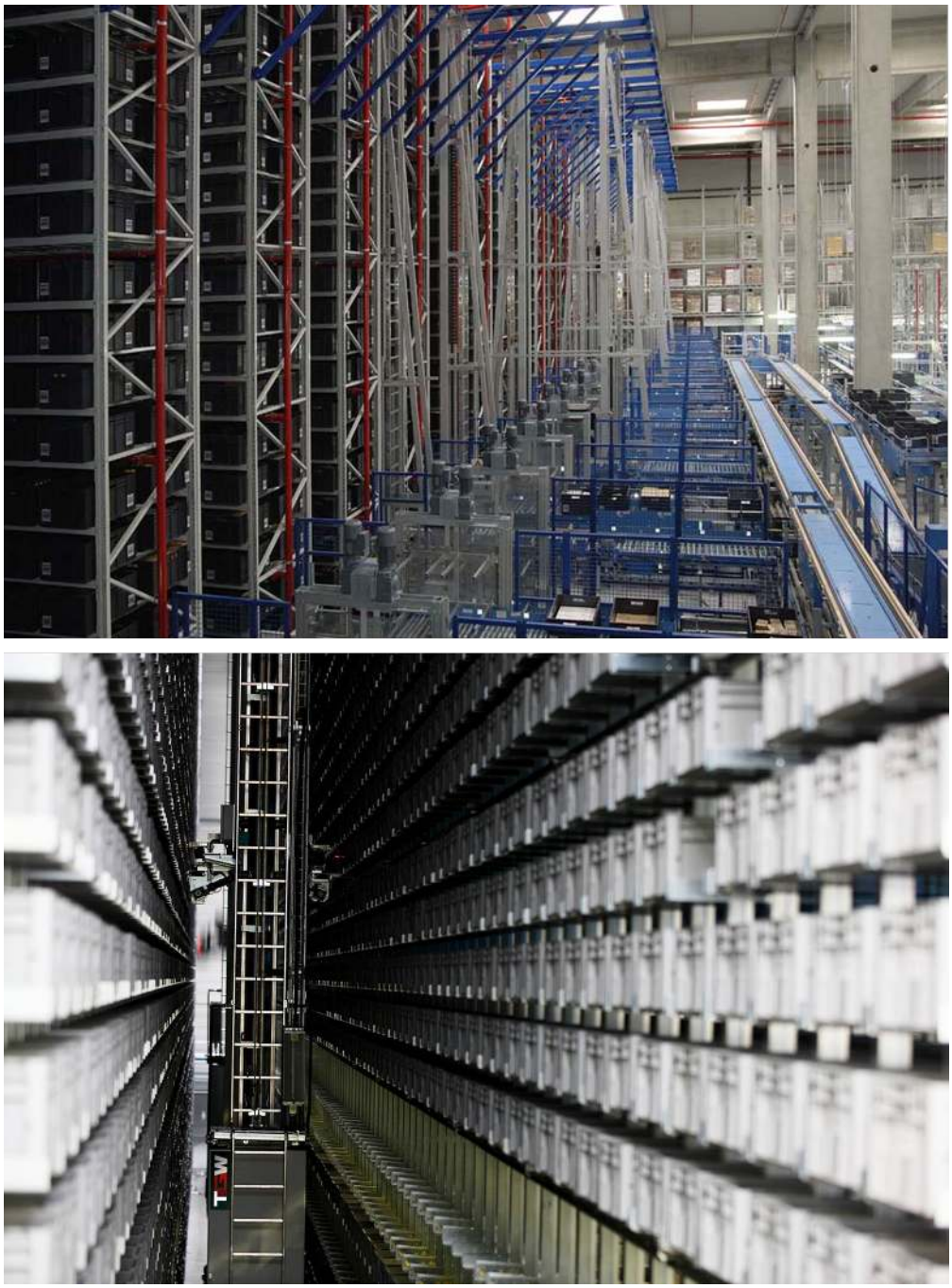

Figure 1.1: Modern AMHS and AS/RS systems

The stacker crane (S/R machine) in an AS/RS is a geometric robot that moves

rectangularly, and is used to store and retrieve loads into/from storage cells. Two drives, one for horizontal motion and the other for vertical drive this autonomous vehicle. The horizontal drive moves the items back-and-forth along the aisle and the vertical one lifts and lowers the load. And they can move simultaneously for the sake of efficiency. Due to their different functions, these two drives often consist of motors of different power. Normally the vertical motor is of higher power than the horizontal one, for it is used to balance the weight of the load and the cart, while the horizontal one only needs to balance the friction. Typically the vehicle is also equipped with one or two shuttle drives that perform the storing or retrieving action.

Empty spaces between the racks form the aisles, along which the stacker cranes can move. An I/O station (P/D station) is usually located at the end or the beginning of the aisle. It is where the incoming loads for storage are picked up and retrieved outgoing loads are dropped off. Figure 1.2 illustrates the generic structure and principle constituents of an AS/RS. A conventional AS/RS operates as follows: first the incoming items for storage are sorted and put into containers or pallets within load size and weight limits. Then they arrive at I/O stations and are registered to central computer. This computer then assigns the items to different storage locations on the storage racks based on the content of the items and the inventory level of the storage racks. Then the stacker cranes pick the items up, move and drop them off at their assigned locations. Upon receipt of a request to retrieve an item, the computer will look through its database and find the storage location of the requested item, then direct the stacker crane to retrieve the load and drop it at the I/O station [10]. Our studied AS/RS, however, is of tremendous size and volume, and highly integrated into the terminal building. The system stretches over several levels, and serves as the transportation hub, linking the storage space to various functional areas. So instead of a single I/O station, our studied systems have multiple entries across the storage racks, so that a request does not always start or end at the I/O station.

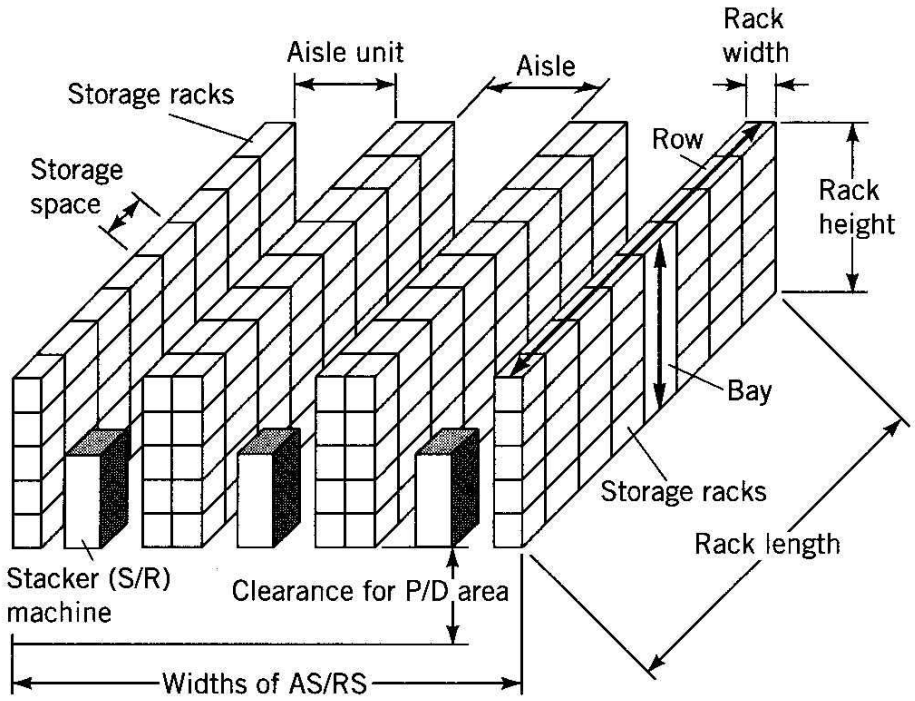

Figure 1.2: Generic structure and principal constituents of an AS/RS

1.2.2 Types and Variations of AS/RS

As diverse as materials handling itself, numerous options and configurations for AS/RSs can be found. We focus on the most basic version of an AS/RS. A basic version of AS/RSs has one stacker crane in each aisle, which is aisle-captive and can only transport one unit-load at a time (single shuttle). In this case, no human interaction is involved, and the storage racks are single-deep, which means that the stacker crane can directly access every load. This AS/RS type is referred to as a single unit-load aisle-captive AS/RS. Numerous AS/RSs vary from this basic version, and an overview is presented in Figure 1.3. The shaded blocks represent the basic version of AS/RS.

1.3 Energy Efficiency in AMHS

The topic of energy efficiency is particularly popular today. Reports and researches have shown that, since the beginning of the industrialization, the carbon

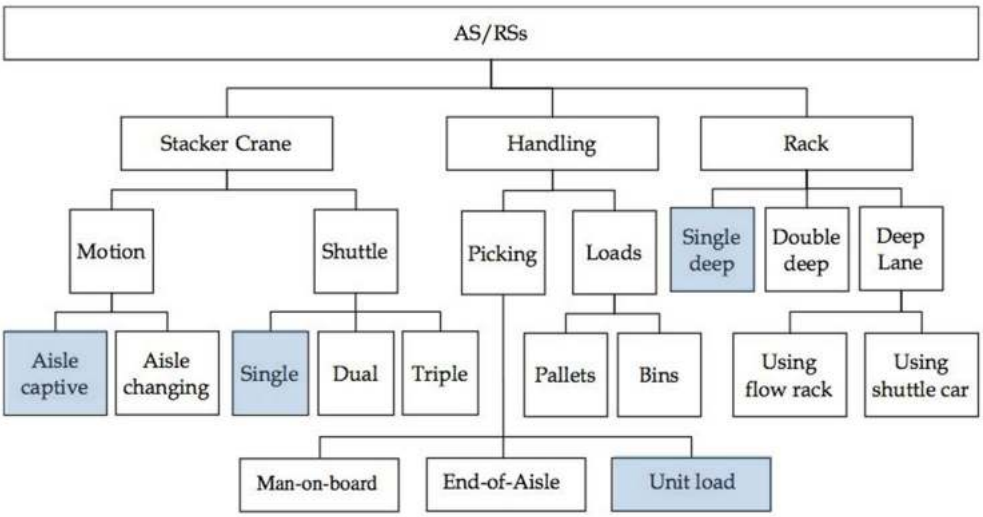

Figure 1.3: Various system concepts for AS/RSs, modified after Roodbergen and Vis 2009 [3]

dioxide percentage in the atmosphere has increased significantly, mainly during the last 50 years (Figure 1.4). This shows that an increase of the concentration of carbon dioxide in earth atmosphere contributes directly to global warming. The disastrous consequence of uncontrolled energy consumption (fossil fuel mostly), which may threaten the very existence of us mankind, has brought the topic of energy efficiency to the global-politic level. Developments in this area are currently getting an additional boost from political resolutions and decisions. The EU's energy policy action plan requires a reduction in CO_2 emissions by 30 percent by the year 2020 – in relation to 1990 levels. As a critical measure for the implementation of this goal, the increase in energy efficiency was set at 20 percent [11]. In summary, responsible and efficient energy consumption is the most important contribution to climate protection.

The cost for operating equipment in automated material handling systems (AMHS)– like manufacturing facilities and distribution centers– is an ongoing cost that requires a substantial commitment in electrical power (Figure 1.5). Many facilities are extremely large, depending on the complexity of the facilities conveyor system, it could account for as much as 50% of a facilitys electrical load [12]. Increased raw material costs and fuel costs have pressured manufacturers to develop initiatives to reduce their material waste and find ways to more efficiently manage

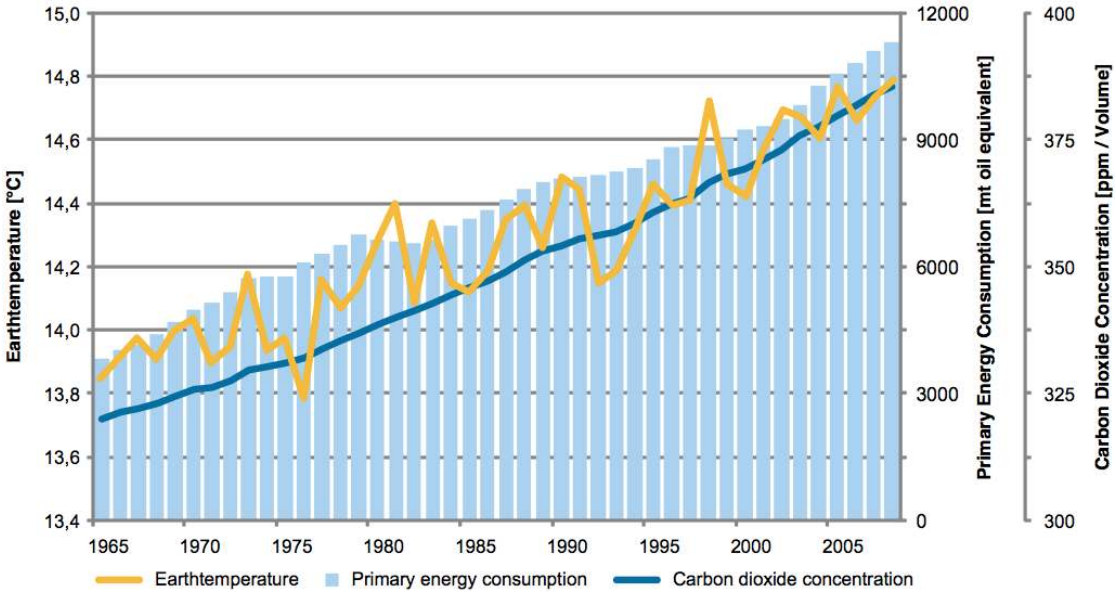

Figure 1.4: Global Temperature, Primary energy consumption and Carbon dioxide concentration since 1965(Günthner, Tilke et al. 2010 [4])

energy usage. In addition, emerging environmental regulations have motivated companies to develop ways to reduce emissions without compromising the quality of their products.

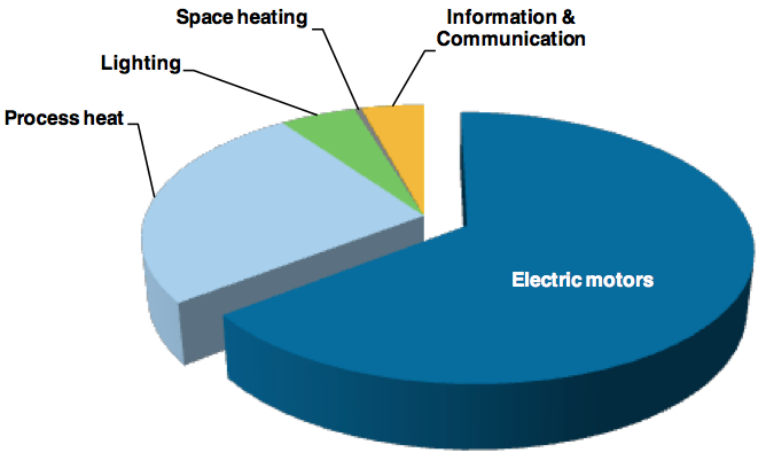

Figure 1.5: Electric Power consumption in the German industry (Kaschenz, Albert et al. 2009 [5])

1.4 Aims, Scope and Contributions

Energy-Efficiency

The first of our objectives is to adopt energy efficiency in a well-studied classic routing problem. To be specific, we try to address the energy efficiency issue in the stacker crane vehicle routing in an automated material handling system located in a large air cargo terminal in Singapore. We name the studied problem Energy-efficient Stacker Crane Problem (EESCP). Unlike most of previous work in which a lot attention has been paid to enhance time efficiency, our problem is set in a context where time is of less significance and energy matters. In this new perspective, for the same instance, the optimal solution in the traveling distance metric can deviate a lot from the optimal solution in the energy consumption metric (See Section 3.1). The energy metric further allows us to make other discoveries about the problem structures.

Arc Patterns and Grid Network

Stacker crane routing problem instances that arise in practice usually come with structured underlying networks or specific arc patterns. In the studied problem, a grid network and the energy cost metric impose constraints and structural properties that have an impact on the complexity of finding solutions, which effect enables us to devise more efficient and accurate models and algorithms than in the general problems.

Our work focuses on two themes about stacker crane routing problems generalized from real-life systems: (1) the request patterns that can be found in the practice; (2) the underlying network of the real systems; and to utilize them to provide better algorithms. On these two topics, our contribution is fourfold: Firstly, we can show that EESCP is NP-Complete, by proving its special case, TSP on solid grid graph in Manhattan norm, is NP-Complete. Secondly, we have discovered a

subset of EESCP with special arc patterns that can be solved as Stacker Crane Problem on linear track, and proposed the sufficient and necessary conditions to identify them. Thirdly, we propose a new exact formulation for our EESCP, the scale of which does not grow with the number of requests in the instance. Finally, for more general cases of EESCP, with the introduction of the grid network in our formulation, we propose the algorithm GRID, which consists of two approximation algorithms: GRID-L and GRID-S. GRID-L is asymptotically optimal, and has a complexity that grows linearly with the number of requests. GRID-S performs well with small-length arcs instances. When GRID-L and GRID-S are combined, they provide a worst-case bound of 5/3.

Dynamic Problem and Multi-capacity Problem

We extend our work to the dynamic problem where not all requests information are given beforehand. We develop a dynamic algorithm using a rolling-horizon approach. The effect of two parameters, look-ahead horizon and decision point, on the performance of the dynamic algorithm are tested via simulations.

An exact formulation is proposed for the multi-capacity problem. The formulation is naturally a bilinear programming, we have alternatively reformulated it into linear form. Given solving the exact formulation is still time consuming, several time-efficient heuristic algorithms are proposed. The performance of these algorithms are tested via simulations.

1.5 Organization of the Thesis

This thesis is organized into six chapters. The rest five are as follows:

- Chapter 2 introduces some highly related work to our studied problem.
- In Chapter 3, we start by stating our studied problem, and point out its NP-Completeness. Then we discuss the Pseudo 2D problem, a peculiar subset of

2D Stacker Crane Problem instances in Manhattan norm that can be solved to optimal in polynomial-time with 1D algorithm.

- For general EESCP instances, we propose a new exact formulation in Chapter 4, and later present two approximate algorithms, GRID-L and GRID-S in Section 4.2 and 4.5. The claimed properties of the two algorithms, such as asymptotic optimality, time complexity and worst-case theoretical bound, are duly examined.

- In Chapter 5, we extend our work to both dynamic and multi-capacity problems. A dynamic algorithm is proposed using the rolling-horizon approach. The algorithm performance under different combinations of its two parameters is discussed. For multi-capacity problem, an exact formulation and three heuristic algorithms are proposed. The performances of the heuristics are tested via simulations.

- In Chapter 6, the conclusions and future work are discussed.

Chapter 2

Related Work

In this chapter, we summarize the existing work that is related to our research issues. Firstly, we will review some general literature on energy-efficiency in material handling systems and warehouses. Secondly, we discuss the most closely related combinatorial optimization problem to our studied problem, the Stacker Crane Problem, and its many variants. Then we look at literature on some other relevant combinatorial optimization problems.

2.1 Energy-efficiency in Material Handling Systems

The topic of energy-efficiency in material handling systems and warehouses are still scarce. Economic and environmental requirements are rarely taken into account concerning power consumption.

W.A. Gnthner, Ch. Tilke et al. [4] did an empirical study on the topic of energy efficiency in the bulk materials handling industry. They brought up some interesting points: that according to statistics, the electric motors make up 60% of the energy consumption of the whole system; and that if you use an electric motor with an annual service life of more than 3,000 hours, 95% of the entire costs during the durability fall on energy consumption, less than 3% on acquisition; it is

therefore short-sighted to make decisions only dependent on the acquisition price. Then they went ahead and did a comparison on transportation energy consumption between using trucks and conveyor belts. They also identified that unloading a ship had the greatest potential of reduction in energy consumption in the context of port handling.

P. Christian, K. Andreas et al. designed a monitoring system in a testing facility [13]. They also discussed the possibilities to save energy by changing the constant speed running conveyor belt motors to on-demand mode; or lowering the drives rotating speed while they are not conveying any goods.

Finally, P.A. Makris, A.P. Makri et al. addressed an energy-efficient order picking problem in ware housing environment [14]. Their problem is directly related to the Traveling Salesman Problem. They concluded that a relatively small loss of service time in many cases may lead to a significant decrease of consumed energy without any additional cost. They solved this problem by changing the cost function in the k-interchange heuristics.

2.2 Stacker Crane Problem

Our studied problem can be classified as a special case of the Stacker Crane Problem (**SCP**), in which objects (such as containers or pallets) need to be transported from their origins to their assigned destinations via a unit-capacity vehicle. The vehicle performs a set of movements (traverse a set of arcs) to load and unload objects at their origins and destinations respectively. The objective is to plan the vehicle movements so as to minimize the total tour cost. Usually this cost is represented by the total tour time, or in some other cases, the Euclidean distance traveled. The SCP can be considered an arc routing problem, particularly a special case of the Directed Rural Postman Problem, a Pickup and Delivery Problem, or a special case of the Asymmetric Traveling Salesman Problem.

The SCP is defined as follows: Let $G = (V, E, A)$ be a mixed graph, where V is the set of vertices with a distinguished initial vertex v_s. E is the set of

edges and A the set of arcs. Let c be a cost function from (E, A) to the set of nonnegative integers, such that for every arc there is a non-negative cost. The corresponding optimization problem is to find a tour starting at v_s and traversing each arc in A and ending at v_s for the unit-capacity vehicle, such that the cost of the tour is minimized [2]. The SCP, being a generalization of the Traveling Salesman Problem, is NP-complete on general graphs [15].

There has been fair amount of significant work on this problem. In SCPs, a single vehicle with unit-capacity is the most common assumption. Systems with more than one vehicle, and vehicles with multi-capacity have also been investigated despite their hardness. Common SCPs that arise from various literature can be generally classified based on 4 criteria other than the number of vehicle: Preemptive or Non-Preemptive; Static or Dynamic; Unit-capacity or Multi-capcity; Underlying graph being on Paths/Trees/Euclidean Space/General Graph etc. In non-preemptive problems, once a request is initiated, the item on the vehicle cannot be dropped off until the delivery is made; whereas in preemptive problems, the item can be dropped off at any intermediate nodes and to be picked up later for delivery. If all the requests information is given before hand, we say the problem is static; If the information of incoming requests is revealed over time, we say the problem is dynamic. In the following sections, we will use $[\cdot|\cdot|\cdot]$ to indicated how each literature falls into these categories, for example, [pre|mc|trees] indicates a preemptive and multi-capacity SCP on trees (pre stands for preemptive, while npre for non-preemptive); and [npre|uc|general] indicates a non-preemptive and unit-capacity SCP on general graphs. We start with static problems.

Static Problems

Atallah and Kosaraju studied a SCP problem with a unit-load vehicle that is confined to linear track with no other constraints imposed on the requests [npre|uc|paths]. They showed that the problem can be solved in $O(m + n\alpha(n))$ time, where m is the number of requests, n the number of vertices, and $\alpha(n)$ the

extremely slowly growing functional inverse of Ackermann's function [16]. They also developed an algorithm that runs in $O(m+n\log n)$ for SCP on circular tracks [npre|uc|circle]. For preemptive problems, they were able to develop an $O(m+n)$ time algorithm, either on linear or circular track [pre|uc|paths/circles]. Michael O. Ball and M. Magazine solved an insertions sequencing problem that arises in a printed circuit board assembly [npre|uc|paths]. They developed a polynomial-time "Balance and Connect" heuristic for their studied RPP, and proved that the heuristic can find the optimal solution when the Manhattan metric is used [17]. Though the application is very different from Atallah and Kosaraju's work, the insights and developed algorithms for their problem are innately the same. We notice the fact that, though being an SCP in the 2D plane, Ball and Magazine's studied problem can be solved with Atallah and Kosaraju's algorithm for SCP on paths. For preemptive problems, Atallah and Kosaraju found an $O(m+n)$ time algorithm for either circular or linear track [pre|uc|paths/circles] [16].

Frederickson and Guan proved the problem to be NP-complete on trees [npre|uc|trees] [18]. They presented several approximation algorithms for SCP on tree graphs with different worst-case bound ranging from 1.21 to 1.5 in [18]. One of these algorithms, which has a 4/3 approximation ration, has been shown to provide optimal solutions on almost all inputs [19]. Frederickson and Guan are able to developed two polynomial-time exact algorithms for preemptive SCP on trees [pre|uc|trees] [15]. One of them runs in $O(k+qn)$ time, where k is the number of objects, n is the number of vertices in the tree, and q is the number of nontrivial connected components. The other algorithm runs in $O(k+n\log n)$ time.

Treleaven, Pavone et al. recently provided an efficient $O(n^{2+\varepsilon})$ and asymptotically optimal (almost surely) class of algorithms to solve stochastic SCPs [npre|uc|Euclidean] in Euclidean norm and uncertain environments [20]. They also had investigated the SCP on roadmaps to address SCPs that arise in small and densely traveled context [npre|uc|general], such as modern robotics and transportation contexts. In their work an asymptotically optimal algorithm was developed for this special case of SCPs [21].

For general graphs, the best approximation ratio is 9/5 and is achieved by the algorithm *CRANE* developed by Frederickson, Hecht et al. [npre|uc|general] [2]. The algorithm is a combination of two algorithms. One consists of a minimum-cost matching and then a minimum spanning tree. It works better if the cost of the arcs is large compared to the cost of the optimal solution. The other is based on the transformation of the SCP instance into a TSP one, then using Christofides algorithm [22], which is more suitable for the opposite situation. Each SCP instance is solved by both algorithms and the better solution is chosen.

Recently, T. Ávila, Á. Corberán addressed the polyhedral description and the resolution of directed general routing problem (DGRP) and the SCP [npre|uc| Euclidean] [23]. They have described large families of facet-defining inequalities and implemented a branch-and-cut algorithm for these problems. They observe that removing non-required vertices may not always help solve the problem, and there are instances where working with the original graph seems to be the only successful way of solving the problem.

It is worth noting that the only known method to solve the SCP on a general graph [npre|uc|general] to optimality, by far, is by generalizing an SCP instance into an Asymmetric Travelling Salesman Problem (ATSP) instance and solve it. Larporte formulated and solved six classes of Arc Routing Problem as TSP, SCP was one of them [24]. Cirasella, Johnson et al. [25] presented a study about ATSP heuristics on eleven types of instances, one of which corresponds to the SCP. Zheng and Zheng used genetic algorithm to solve the ATSP formulation of SCP and its multi-vehicle variant [26]. Jordan Srour and Steef van de Velde [27] have presented a statistical study comparing the difficulty of the resolution of SCP instances with that of general ATSP instances. Their conclusion is that SCPs are not necessarily easier than other ATSPs, but a special subset of SCPs, termed drayage problems, are more readily solved.

Multi-Capacity Problems

Though the most literature on SCP assumes a unit-load vehicle, multi-capacity vehicle systems widely exist in practice. If the vehicle has multi-capacity, the problem is NP-hard even if the underlying graph is a simple path [npre|mc|paths] [28]. For preemptive multi-capacity cases, Guan D.J. et al. proved it P on paths [pre|mc|paths] and NP-complete on trees [pre|mc|trees] [28]. He proved that for the multi-capacity problem on paths, the optimal solution can be found in $O(k + n)$ time if the starting vertex is at one of the endpoints of the path. Despite the hardness to get theoretical results, in practice it is shown that increasing the capacity of the vehicle will yield better system throughput [29], so multi-capacity problems never lack attention. In fact, in most literature, SCPs with multi-capacity are often referred to as the Single Vehicle Routing Problem with Pickup and Delivery (SVRPPD).

For uncapacitated SVRPPD (unlimited capacity), one of the first studies was made by Stein [30]. The author proposed a simple heuristic with an asymptotic performance ratio of 1.06. Psaraftis developed a $O(N^2)$ heuristic for this problem in Euclidean plane, though by the name of single vehicle many-to-many dial-a-ride problem with no time constraints. The heuristic's worst-case perfformance is 300% over optimal, if the TSP heuristic of Christofides [22] is used in the first step [31]. Psaraftis also introduced one of the first local search methods for the SVRPPD, the k-interchange procedure [32].

For the capacitated SVRPPD, Kubo and Kasugai made an experimental comparison of classical construction heuristics and local search methods [33]. Van der Bruggen et al. have constructed a 2-phase local-search heuristic for the capacitated SVRPPD with time windows [34]. In the first phase, a feasible solution is obtained according to each request's time window, while taking load and precedence constraints into account. If the solution still violates some time windows, in the second phase an arc-exchange procedure is applied to obtain feasibility. Desrosiers et al. introduced an exact dynamic programming algorithm for the capacitated problem

with time windows [35]. Their algorithm was able to efficiently eliminates infeasible and dominated states,and solve instances with up to 40 vertices in a very short time. Kalantari et al. developed a branch-and-bound algorithm for both capacitated and uncapacitated SVRPPD. Their algorithm eliminates the branches on the search tree that would lead to solutions violating precedence constraints. Ruland and Rodin presented an exact branch-and-cut algorithm using four classes of inequalities as cuts [36].

Multi-capacity SCPs are also studied in practical context as multi-load AGV dispatching problem for specified environments. Grunow, Gnther et al. studied the dispatching of multi-load AGVs in container terminals [37]. In contrast to single load vehicles, a multi-load vehicle can still be partially available to load another container if one of its many item-holding positins is free. This results in a new decision which can be made in the assignment procedure, namely how to insert a new request in the already scheduled route. A mixed integer linear programming model has been formulated to dispatch multi-load AGVs such that the lateness of AGVs is minimized. Furthermore, the authors showed that the heuristic, in contrast to the optimal approach, can efficiently be applied to real-life container terminals with large numbers of AGVs and high workloads for the AGV system. Sinriech and Palni developed an optimal scheduling algorithm for a single multi-load vehicle travelling in a closed loop during a finite planning horizon [38]. The arrival time and processing time of each job are known. Sinriech and Kotlarski extended this algorithm such that it can be used to schedule dynamically multi-load vehicles in a single loop while minimizing transfer times of jobs and the number of loops travelled by the vehicles [39]. Due to the dynamic nature of the algorithm, changes in the scheduling plan can be made to react to unexpected events. It has been shown that this dynamic algorithm outperforms existing commonly used non-dynamic scheduling rules from a perspective of cycle times and work in progress. In Table 2.1 and 2.2 we summarize the state of art on the research of unit-capacity and multi-capacity static SCPs.

Unit-Capacity	Non-preemptive	Preemptive
Path	P:$O(m + n\alpha(n))$[16]	P:$O(m + n)$[16]
Circle	P:$O(m + n \log n)$[16]	P:$O(m + n)$[16]
Tree	NPC:$\frac{4}{3}C^*$[18]	P:$O(k + n \log n)$[15]
Euclidean	NPC:Asymptotically Optimal almost surely[20]	NPC
General	NPC:$\frac{9}{5}C^*$[2]	NPC

Table 2.1: The state of art for static unit-capacity SCPs

Multi-Capacity	Non-preemptive	Preemptive
Path	NPC[28]	P:$O(m + n)$[28]
Tree	NPC	NPC[28]
General	NPC	NPC

Table 2.2: The state of art for static multi-capacity SCPs

Multi-Vehicle Problems

SCPs with a fleet of vehicles rather than a single vehicle are usually referred to as multi-vehicle routing problem with pickup and deliveries (full-form VRPPD).

Dumas et al. studied a VRPPD with time windows, and developed an exact column generation algorithm that can handle multiple depots and different type of vehicles [40], and can solve instances with 55 requests. Xu et al. studied a practical VRPPD with time windows and many restrictions from real-world logistics, such as multiple time windows, compatibility between carriers, and driver work rules [41]. They proposed a column generation algorithm to solve this problem. Their algorithm was able to provide near-optimal solutions to instances as large as 200 requests within a reasonable time. Ropke and Cordeau proposed a branch-and-cut-and-price algorithm for a VRPPDTW. They formulated their studied problem using two new formulations, and introduced several families of valid constraints to strengthen these two formulations [42].

Lau and Liang presented a two-phase heuristic method for VRPPD, in which the vehicle number is unlimited. In the first phase, they combine construction and a sweep procedure to generate an initial solution. Then the second phase uses a tabu search with three different neighborhood moves to improve the solution [43]. Another two-stage hybrid heuristic for a VRPPDTW problem was developed by Bent and Van Hentenryck [44]. The first stage of the algorithm uses a simulated annealing algorithm to decrease the number of routes, while the second stage uses Large Neighborhood Search (LNS) to decrease total travel cost. The neighborhood in this LNS is defined as the set of solutions that are reachable by relocating at most p requests.

Some other relevant studies can be found in practical contexts such as Automated Guided Vehicle Systems. The literature on this topic focuses more on the design issues such as collision avoidance [45] [46] [47], or AGVs dispatching. Most of the research efforts on the AGV dispatching problem have been designed for specific environments, and especially flexible manufacturing systems. Lee tested the performance of three composite AGV dispatching rules on a simulated multi-vehicle system based on discrete event simulation [48]. And Schouwenaars, De Moor et al. presented a mixed linear integer programming model, to solve the problem of multi-vehicle moving to different positions avoiding each other and stationary obstacles [49].

Dynamic Problems

In this section we review some literature on the Dynamic SCP, with single or multiple vehicles. A Dynamic SCP is an SCP where transportation requests arrive dynamically and stochastically. Some of the requests are revealed or updated during the period of time in which operations take place. Research on dynamic problems assumes the requests arrive in a known or unknown stochastic process. A commonly used strategy to address the dynamic component of the problem is to adapt an algorithm that solves the static version of the problem.

The main application of the Dynamic SCP is the problem of operating a trucking fleet to move full truckloads between pickup and delivery locations. For this reason the Dynamic SCP is sometimes referred to as the dynamic full truckload pickup and delivery problem. For a survey of dynamic models for this problem see [50]. Powell studied a dynamic SCP that arose in the truckload motor carrier industry [51] [52]. A fleet of vehicles serve regions all over the country. One or a few vehicles may be requested to transport a load of freight from one city to another, with only little advance knowledge about future requests. Information on probability distribution of the requests between any pair of regions, and the average net revenue per request is assumed known. He used a network flow model with two types of arcs to calculate an approximate marginal value of an additional vehicle in the future. Then with this calculated information he generated a standard network to be optimized and give dispatching decisions for a day.

Swihart and Papastavrou [53] analyzed the performance of three developed policies on both single-vehicle and multi-vehicle dynamic PDPs under different traffic intensity, by means of Queuing Theory and simulation. Yang, Jaillet et al. [54] studied a real-time multi-vehicle truckload Pickup and Delivery problem. To handle the dynamic component, they used a rolling-horizon framework to solve this problem many times, each time as a static problem. Their computation result showed that though their approach outperformed all others, similar results could be achieved by some simple policies with much less computation effort. Later in [55], they examined a similar problem with two new policies. These new policies involve re-optimizing every time a new request arrives and assume some knowledge about the probability distribution of incoming requests. They concluded that knowing the distribution indeed improved the result.

A study was carried out by Tjokroamidjojo, Kutanoglu et al. [56]. Their study examined the effects of two parameters (τ, T) in the routing policies, in which a re-optimizing is carried out every time a request becomes known. The parameter τ represents how many days in advance requests are known, while T represents how many days in advance the decision was made. The results showed that policies

(3,1),(5,1),(5,3) performed significantly better than the policies (3,3),(5,5),(5,5) respectively, suggesting that instant decision will lead to worse result.

Tabu search and annealing heuristic were used in a complex Dynamic SCP arising from a real-world dispatching problem of an electric monorail system [57]. Their solution strategy consists of solving a static problem several times without any knowledge about the future. The objective of the static problem was modified to improve the overall performance of the online algorithm. Mes, van der Heijden et al. developed an agent-based approach for a version of the Dynamic SCP. An agent-based model attempts to model complex phenomena using a set of individual agents with specific tasks and goals. Their simulation experiments showed that the agent-based approach behaved generally better than simple scheduling heuristics [58].

2.3 Closely Related Combinatorial Problems

In addition to the SCP, we review a selection of important and related combinatorial problems.

Pickup and Delivery Problem

Among others, Pickup and Delivery Problems (PDPs) constitute an important class of vehicle routing problems in which commodities or people are transported between locations in a physical environment. PDPs arise in many contexts such as logistics, transportation systems and material handling systems, etc. PDP generally consists of a fleet of vehicles and a set of customer requests. Each request specifies the locations of the pickup point and delivery point and the size of the load to be transported. A request is considered served if the load of commodities that the request specifies is picked up at the pickup point and then the same load is dropped at the delivery point. Furthermore, the same vehicle must visit the pickup location before the delivery location. Each vehicle in the fleet has a given

capacity and a start location. The objective is to minimize the total cost, which may include the fixed vehicle cost and the travel cost, while satisfying all customer demand.

PDPs are generally classified into 3 different groups based on the perspective of the commodities [59]. 1) Many-to-many PDP, characterized by several origins and destinations for each commodity/customer; 2) one-to-many-to-one PDP, where commodities are first located at the depot to be transported to many customers and then back to the depot; 3) one-to-one PDP, where each commodity (which can be seen as a request) has a given origin and a given destination. Comprehensive reviews of PDP can be found in [60] [61] and [62].

Two important problems belong to the family of one-to-one PDP, one is Vehicle Routing Problem with Pickup and Delivery (VRPPD), and the other is Dial-A-Ride Problem (DARP). The former deals with transportation with objects, while the DARP applies to the transportation of people, with customers convenience taken into account [63] [59]. When one adds the unit capacity constraint to the fleet of vehicles, the one-to-one PDP is referred to as the Stacker Crane Problem (SCP). In this sense, we could say that SCP is a unit-load VRPPD. Also in many studies of the SCP, single-vehicle constraint is assumed.

The Dial-a-Ride Problem

The Dial-a-Ride Problem (DARP) is a particular case of the VRPPD arising in contexts where passengers are transported, either in groups or individually, between specified origins and destinations. Recently there is a surge in the DARP popularity due to internet transportation companies like Uber. DARP arises in several practical applications [64], such as shared taxi services [65] [66], the transportation of elderly and/or disabled persons [67] [68], Telebuses [69], and providing transportation services in regions with low population density [70]. Generally, these problems contain several constraints that control user inconvenience, which also distinguish itself from the basic VRPPD. Examples of such constraints are

tight time windows and maximum ride times [71] [72] [73]. The minimization of user inconvenience often has to be balanced with operation costs since these objectives usually conflict. Models and algorithms for the static and the dynamic versions of the DARP were surveyed by [63].

The Vehicle Routing Problem

The Vehicle Routing Problem (VRP) is reducible to the aforementioned PDP and a generalization of the Traveling Salesman Problem. The VRP can be defined as a problem of finding the optimal routes of delivery or collection from one or several depots to a number of cities or customers, while satisfying some constraints. Collection of household waste, gasoline delivery trucks, goods distribution, snow plough and mail delivery are the most used applications of the VRP. It is one of the most important and studied combinatorial optimization problems. First introduced by Dantzig and Ramser [74], more than 50 years have passed, and hundreds of models and algorithms have been proposed for the optimal and approximate solution of the different versions of the vehicle routing problems. The constant interest in vehicle routing problems is motivated by its practical relevance as well as its considerable difficulty [75]. There are dedicated books on the VRP such as [75] [62] and many state-of-the-art reviews like [76].

The Traveling Salesman Problem

The well-known Traveling Salesman Problem (TSP), formulated on a weighted graph, is to find a minimum-weight simple closed exhaustive tour that visits all nodes in the graph. The TSP is reducible to the VRP, and is NP-Hard even under a number of limiting assumptions. A common such assumption is that either the problem graph is undirected, or else the edge weights are symmetric. In the more general case, of directed, asymmetric problem graphs, the TSP is often called instead the Asymmetric Traveling Salesman Problem.

In both the symmetrical and asymmetrical cases, the TSP is made hard specifically because of sub-tour elimination. Consider instead a relaxation of the TSP: to find a minimum-weight set of mutually disjoint simple closed tours together visiting all nodes. The union of such tours is sometimes referred to as a cycle factor. The Cycle Factor Problem (CFP) has a straightforward reduction to the Assignment Problem (AP) [77]. The AP has a concise linear programming formulation, and there are a number of exact and approximate algorithms to solve it in polynomial time [78].

The AP features in a number of exact and approximate algorithms for the TSP and ATSP [79]: it is a common starting point for branch-and-cut approach algorithms; in approximation algorithms, the idea is to identify a minimum-weight cycle factor, and then use good heuristics to combine the sub-tours into a single feasible tour.

As every SCP instance can be generalized into an ATSP instance, there have been studies on solving SCP instances (as well as some other arc routing problems) as ATSP instances [24]. The author concluded that this approach works well on low-density graphs containing few edges. And to our knowledge, this is the only known approach to solve SCP to optimality by far.

In all, the TSP has been one of the most widely studied combinatorial optimization problems and occupies a central place in operation research, ever since its introduction around 1930s [80]. For extensive methodologies, case studies on this problem, see [80] [81] [82] [83].

2.4 Summary

Further, Eiselt, Gendreau et al. [84] presented a survey on the Rural Postman Problem and devoted a section to the SCP. For an extensive review on static and dynamic pickup and delivery problem, see [59] and [85] respectively. In summary, the SCP, as an important problem that generalizes many practical issues, as well

as an important class of Arc Routing Problem, has been extensively studied. However, the structured underlying networks or special arc patterns that the practical problem instances usually come with have not been paid enough attention to. In this thesis, we study a problem with a grid network and special cost metric that impose constraints and structural properties that have an impact on the complexity of solving the problems, which effect enables us to devise more efficient and accurate models and algorithms.

Chapter 3

Pseudo-2D Problems

First we give a formal definition and introduction to our studied problem.

3.1 Problem Statement

We investigate a problem that arises in an automated material handling system located in a large air cargo terminal in Singapore (as shown in Figure 3.1). Everyday, large batches of outbound/ inbound/ transfer cargoes are transported to the terminal to be processed. Usually when they arrive, they are temporarily stored inside a storage and retrieval system inside the terminal, and to be retrieved some time later. The storage and retrieval system, which is served by a stacker crane vehicle, is of several stories high. In our studied system, the transhipment cargoes usually stay for a couple of days before getting picked up. Consequently, the time window of each cargo request and the total finishing time are no longer of the highest priorities, and energy-efficiency of the operations becomes important. This motivates us to look for solutions to the stacker crane routing with the minimal energy consumption. We call this problem *Energy-Efficient Stacker Crane Problem*(*EESCP*).

Our studied problem can be classified as a special case of the Stacker Crane Problem (**SCP**), in which objects (such as containers or pallets) need to be transported from their origins to their assigned destinations via a unit-capacity vehicle.

Figure 3.1: Two examples of storage and retrieval system in air cargo terminals

The vehicle performs a set of movements (traverses a set of arcs) to load and unload objects at their origins and destinations respectively. The objective is to plan the vehicle movements so as to minimize the total tour cost. Usually this cost is represented by the total tour time, or in some other cases, the Euclidean traveled distance. In this work, the tour cost is the total operating cost measured in energy consumption, which can be equivalently converted to the weighted traveled distance in Manhattan norm (explained in Section 3.1). Also in this problem, the items come in batches, thus all pickup and delivery requests information are collected and known in advance, consequently our studied problem is considered static.

The *Energy-Efficient Stacker Crane Problem*(*EESCP*): In this problem, a unit-capacity stacker crane vehicle serves a storage rack with N_V rows and N_H columns (as shown in Figure 3.2). Every cell on this storage rack is of the same size, h in width and v in height. The vehicle itself has a net mass of m_0 when empty-loaded, and can move freely to any cell on this rack to perform either a pickup action or a

drop-off action. A set of pickup and delivery requests R is given. Once a request is initiated, the stacker crane cannot perform another task before the current one is finished (non-preemptive). The cost induced is measured in terms of energy consumption. A solution, or a tour, is a sequence in which the stacker crane can perform all the given requests and return to its starting position. The objective is to find such a tour with the minimum energy cost.

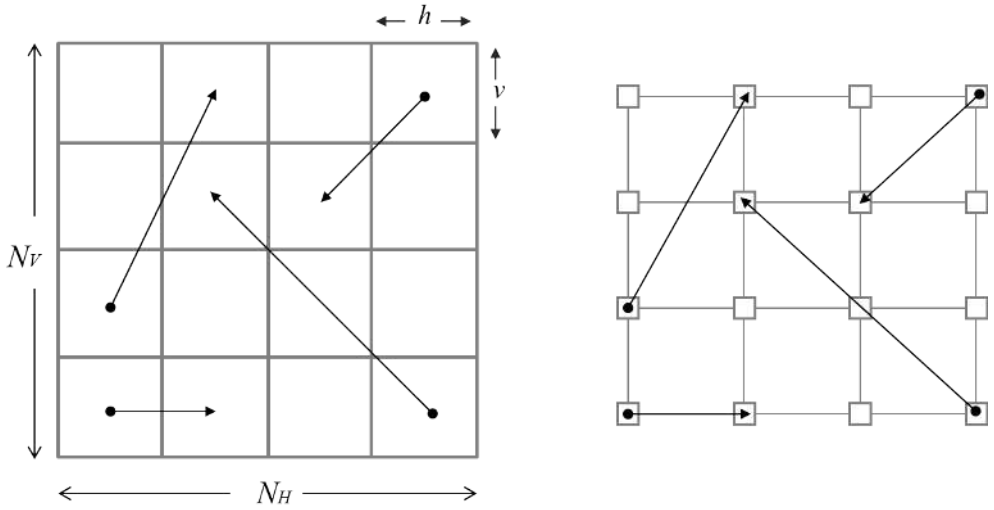

Figure 3.2: The EESCP instance with its underlying grid

This problem setting is more specific than the general SCP:

- The vehicle is traveling on a fixed, connected and undirected grid graph;
- The cost is in Manhattan norm and associated with the horizontal and vertical displacement the stacker crane travelled rather than the Euclidean distance. Consequently, in this problem, while the vehicle can move freely in any diagonal direction, the cost that a diagonal movement incurs always equals to the cost incurred by its corresponding horizontal and vertical components combined;

Observe that for unit-capacity and non-preemptive problems, the given requests arcs are always part of the optimal tour. Thus the part of energy consumption incurred by the given requests is in fact fixed and irrelevant to neither the

vehicle tour decisions nor the weight of carried items. So the optimization only applies to the tour cost in-between given requests, where the vehicle is empty-loaded. The energy consumption for the empty vehicle to travel from one cell to the one next to it is calculated as follows:

energy cost per horizontal move:

$$c_H = k_H \cdot m_0 g\mu \cdot h \tag{3.1}$$

energy cost per vertical move:

$$c_V = k_V \cdot m_0 g \cdot v \tag{3.2}$$

where k_H and k_V are constant coefficients when the stacker crane is working under certain condition, e.g., fixed cruising and rising speed. g and μ are the constant of gravitation and friction coefficient respectively. A "move" is defined as a unit horizontal or vertical movement from one cell to one of its neighboring cells for the vehicle.

In practice, the energy cost function of movement actually depends on the type of equipment. Detailed energy cost calculations can be given according to the diagrams of specific equipment. However varied they are, they are all more or less linear functions of the weight. This is understandable, as what governs the energy costs of horizontal and vertical displacements are still the fundamental principles of physics. Thus we used an extremely simplified version of cost functions, and aggregated all the variations and uncertainties into parameters k_H,k_V. This way, our analysis can be simplified, and be applied to a wide variety of cases. For literature to support this assumption, please refer to [86]. Furthermore, we can find similar use of linear energy cost function in other literature such as [14].

The difference between minimizing the total traveling distance and the total energy consumption is mainly affected by the different ratio of $k = \frac{k_H \cdot \mu}{k_V}$. In reality, k is expected to be smaller than 1. In Figure 3.3 and Table 3.1 we see the difference

in incurred energy cost between the distance-minimal optimal solution and the energy-minimal optimal solution under different values of k. A smaller k implies a larger energy-cost difference between minimizing the two different objectives, as much as 83%. On the other hand, we also run a simulation to show how much distance cost of an energy-optimal tour would deviate from a distance-optimal tour (if the vehicle velocity is assumed constant, we can also interpret this result in terms of time). In Figure 3.4 and Table 3.2 we see that the largest deviation happens when k is close to zero, and smallest when k is close to 1, similar to the trend shown in Figure 3.3. Comparing Figure 3.3 and Figure 3.4, we also notice that, at the same k, the percentages of deviations are not the same. For example, in Figure 3.3, the largest deviation is around 20% when $k = 0.1$. Compare this to the largest energy deviation, which is 30%, we can draw the conclusion that for the current k, we can averagely sacrifice around 20% of time to gain 30% of energy, which is a good tradeoff in some situations. Also, we find that the standard deviation of the difference in Table 3.2 is much smaller than that in Table 3.1. This phenomenon also implies that for the same loss of time, there can be strategies to maximize the gain of energy.

	Difference in Energy Cost (%)				
k	1:10	3:10	5:10	7:10	9:10
Max.	82.99	21.16	13.48	6.80	5.73
Ave.	**29.51**	**9.68**	**5.53**	**3.16**	**2.19**
Min.	5.58	1.23	1.12	0.75	0

Table 3.1: The difference in energy cost when optimized in different metrics (%)

Then an EESCP instance is defined by $I = G(V, E, A)$ and its cost function $c(\cdot)$. $G(V, E)$ is the graph representation of the grid that the vehicle serves, $V = \{v = (x_v, y_v)\}$ being the set of vertices, E the set of edges. Arc set A represents the requests in R on the gird. For any arc $a = (v_1, v_2)$ on the grid, $c(a) = c_H|x_{v_1} - x_{v_2}| + c_V|y_{v_1} - y_{v_2}|$. The EESCP is essentially an SCP on a solid grid in Manhattan norm.

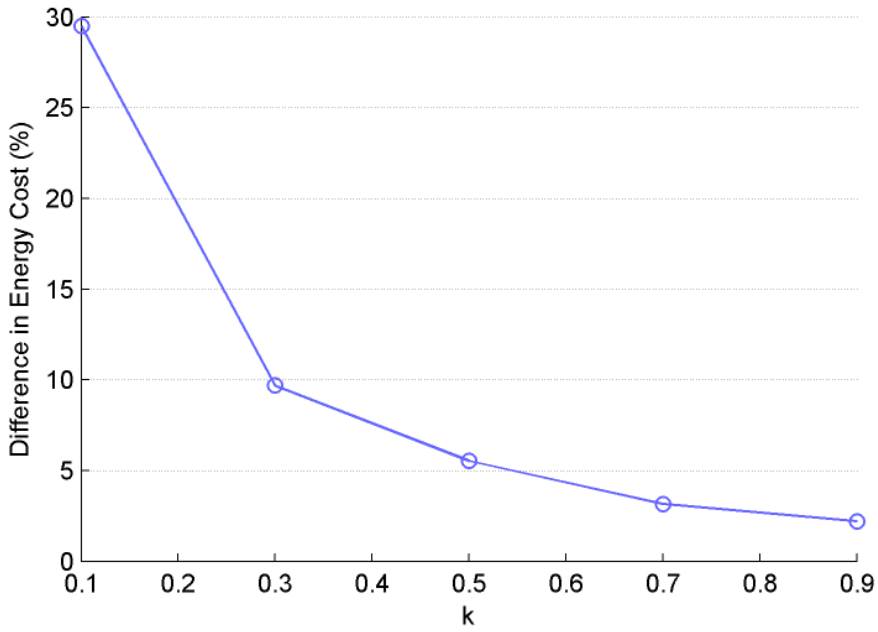

Figure 3.3: The difference in energy cost when optimized in different metrics

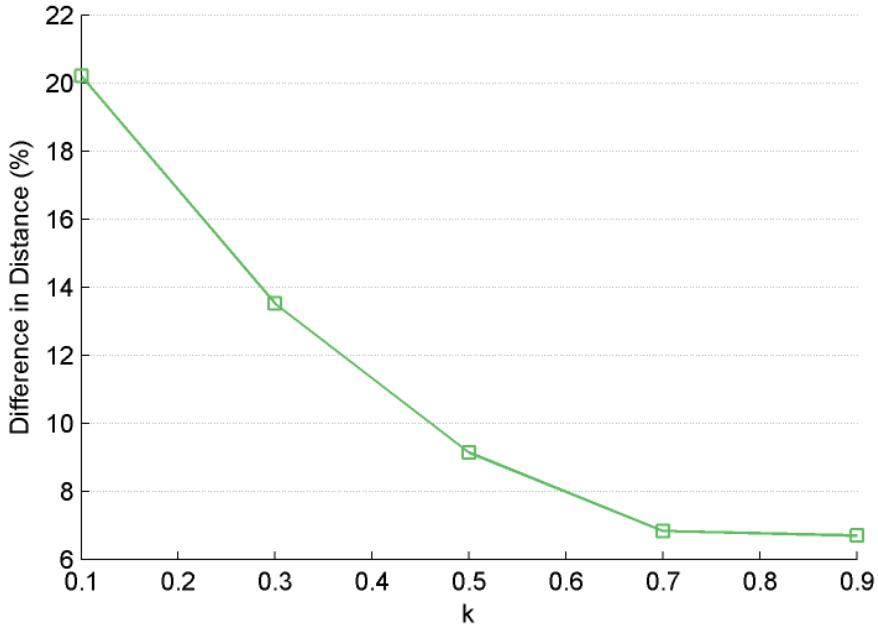

Figure 3.4: The difference in traveling distance when optimized in different metrics

	Difference in Traveling Distance (%)				
k	1:10	3:10	5:10	7:10	9:10
Max.	35.45	25.85	16.55	10.45	9.92
Ave.	**20.20**	**13.51**	**9.13**	**6.83**	**6.69**
Min.	7.49	4.57	4.02	2.11	1.97

Table 3.2: The difference in traveling distance when optimized in different metrics (%)

A solution to I is a sequence to traverse all the arcs in A. Let S^* be the optimal sequence set. Every sequence s, when viewed on the graph, can be translated into the trajectory of the vehicle. This trajectory, denoted by T_s, is a degree-balanced and connected (eulerian) graph that starts and ends at the same position in the 2D plane. $c(s) = c(T_s)$ is the cost incurred by $s \in S^*$, excluding the fixed cost of arcs in A.

3.2 NP-Completeness of EESCP

EESCP is a essentially an SCP on solid grid graph in Manhattan norm (SCPSM). For definition on solid grid graph, see [87].

Theorem 3.1 *The SCP in Manhattan norm on solid grid graph (SCPSM) is NP-Complete.*

Proof: Note that the Traveling Salesman Problem on solid grid graph in Manhattan norm (TSPSM) is a special case of SCPSM. In an SCPSM instance $I = G(V, E, A)$, if we restrict that $\forall a = (v_i, v_j) \in A, v_i = v_j$, then we have reduced A to a node set $A \subseteqq V$ on the grid graph. Thus TSPSM is a special case of SCPSM.

C.H. Papadimitriou [88] has shown that the Traveling Salesman Problem with rectilinear structure and in Manhattan norm is NP-Complete. With a similar method to his, the NP-Completeness of TSPSM can be obtained. The NP-Completeness of TSPSM then implies the NP-Completeness of SCPSM. Thus we have proved that EESCP is NP-Complete. □

3.3 Conditions for Free-Permutation

Back in 1988, Atallah and Kosaraju developed an algorithm that runs in $O(m + n\alpha(n))$ time for one-dimensional(1D) SCP on linear tracks, when studying to minimize the movement of robot arms [16]. Almost at the same time, Ball and Magazine developed the exact same algorithm to solve the SCP in Manhattan norm that arises in circuit printing problem [17]. What interesting is, the SCP which Ball and Magazine studied is a 2D problem that does not fit the description of "SCP on linear tracks" (Figure 3.5). As we investigate, the implication that a subset of 2D problems can be simply solved by 1D problem algorithms, is true. We call this subset of 2D problems Pseudo-2D problems, since they seem to exist somewhere in between these two dimensions. But how to identify these problem instances among others? And to what extent can we push the 1D algorithm to apply to the 2D problems? In this chapter we try to answer these two questions.

The following is a list of frequently used notation in the following sections.

Notation in this chapter:

Sets

A_1	Set of arcs that represents the requests in a 1D SCP istance;
T_s	The set of arcs that forms the trajectory (eulerian graph) of the vehicle if it performs A_1 in sequence s;
$P_H(T_s), P_V(T_s)$	The horizontal and vertical projections of T_s. They themselves are 1D eulerian graphs;
I_H, I_V	The horizontal and vertical 1D subproblem instances of a 2D instance I;
T_H^*, T_V^*	The optimal eulerian graph obtained by respectively solving I_H and I_V;
S_H^*, S_V^*	The respective sets of all the sequences that can be obtained from T_H^* and T_V^*;

Parameters

x_i^O, x_i^D	The respective coordinates of the pickup and drop-off positions of arc $i \in A_1$;
n	The number of arcs in A_1;

Others

$c(s)$	The cost incurred by sequence s;
$c(A)$	The cost incurred by the set of arcs A.

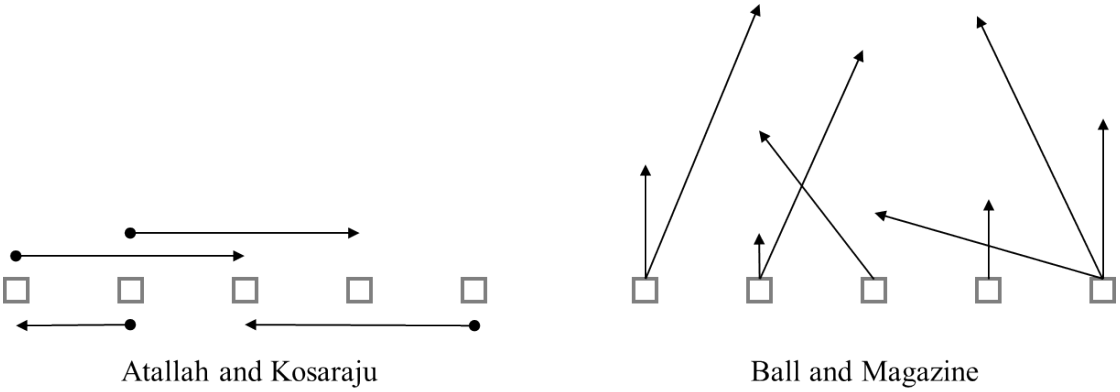

Figure 3.5: Problems with two different structures can be sovled with the same algorithm

A general instance of SCP on paths I_1, can be defined by an arc set $I_1 = G(A_1), A_1 = \{a_i = (x_i^O, x_i^D), i = 1, 2, ..., n\}, |A_1| = n$, $x_i^O \in \mathbb{R}$ is the coordinate of the pickup position of arc i on the axis, and x_i^D the drop-off position.

We briefly introduce some key points about Atahhah and Kosaraju's algorithm for solving 1D SCP. See Figure 3.6 for demonstration. The algorithm starts by constructing the optimal 1D eulerian graph. The construction is done by the following two steps:

Step (1): Degree-balancing every node with minimum number of augmenting arcs; (illustrated by dashed arrows in (a) of Fig. 3.6);

Step (2): If the resulting graph has disconnected sub-tours, use minimum pairs of arcs in opposite directions to connect the sub-tours. Then any euler tour of the resulting eulerian graph gives an optimal cost.

Note that the resulting graph of step (1) is unique, while that of step (2) is not necessarily unique. Due to multiple solutions of step (2), we can identify multiple optimal 1D eulerian graphs with this algorithm. Also note that within the same optimal eulerian graph, usually multiple optimal sequences can be identified. For more details, see [16].

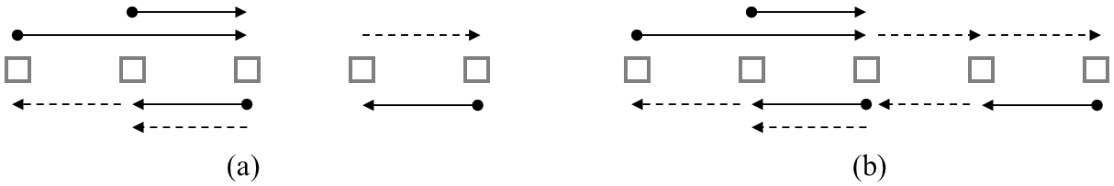

Figure 3.6: (a) Degree-balancing and (b) connecting the graph to make an eulerian graph that contains the optimal tour

Assume that we have an EESCP instance $I = G(V, E, A)$, S^* its optimal sequence set, and C^* its optimal cost. Then we project $T_{s^*}, \forall s^* \in S^*$ horizontally and vertically, and let $P_H(T_{s^*})$ and $P_V(T_{s^*})$ be the respective horizontal and vertical projections. Note these two projections are also two 1D eulerian graphs on their own. As the cost is measured in Manhattan norm, $c(P_H(T_{s^*})) + c(P_V(T_{s^*})) = c(T_{s^*})$.

On the other hand, by projecting the arc set A of I horizontally and vertically, we generate two 1D subproblem instances $I_H = G(A_H)$ and $I_V = G(A_V)$, respectively, as seen in Figure 3.7. A_H and A_V are the respective horizontal and vertical "shadow" of arcs in A while retaining their own identities in A. The optimal eulerian graphs to these two 1D instances can be obtained independently by aforementioned algorithm for SCP on paths, let them be T_H^* and T_V^* respectively. Let S_H^* and S_V^* be the sets of all the optimal sequences that can be deduced respectively from T_H^* and T_V^*. Now let us first introduce a lemma:

Lemma 3.1 *Let* $S' = S_H^* \bigcap S_V^*$ *If* $S' \neq \varnothing$, *then* $S' \subseteqq S^*$.

Proof: Let us assume on the contrary, that $S' = S_H^* \bigcap S_V^* \neq \varnothing$, but $S' \nsubseteqq S^*$. Then there must $\exists s' \in S'$ and $C^* < c(s')$, where $c(s')$ is the cost that incurred

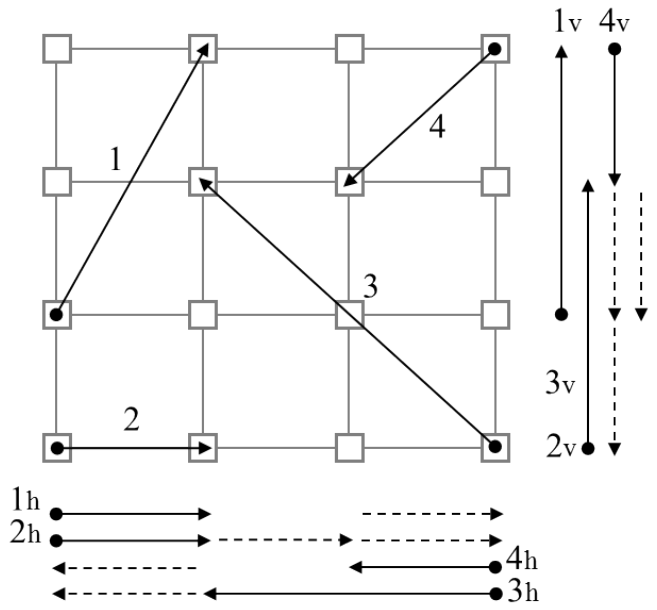

Figure 3.7: The 2D instance and its two corresponding 1D problem instances

by sequence s'. Let s^* be any sequence in S^*, $c(s^*) = C^*$. We then examine the projections of T_{s^*}, $P_H(T_{s^*})$ and $P_V(T_{s^*})$. Since the problem is in Manhattan norm, so we have:

$$c(s^*) = c(P_H(T_{s^*})) + c(P_V(T_{s^*})) = C^* < c(T_H^*) + c(T_V^*) = c(s') \qquad \forall s' \in S' \quad (3.3)$$

But according to the definition of T_H^*, $c(T_H^*)$ is the minimum cost possible for any 1D eulerian trajectory to exist at all. Thus $c(P_H(T_{s^*})) \geq c(T_H^*)$. The similar holds for $c(T_V^*)$. Thus $c(P_H(T_{s^*})) + c(P_V(T_{s^*})) \geq c(T_H^*) + c(T_V^*)$. Hence the contradiction, and we have the conclusion that if s' is a sequence that simultaneously minimizes the horizontal and vertical subproblems, then $s' \in S^*$. □

Lemma 3.1 gives us an interesting perspective. If we work on the two 1D subproblems independently and their solutions agreed on some optimal sequences, then these optimal sequences are optimal to the original 2D problem as well. However, this insight is not very helpful to directly solve the 2D problem for the following reasons: (1) For a given optimal 1D eulerian trajectory, there can be an exponential number of optimal sequences, generating all of them is impractical;

(2) Often the prerequisite $S' = S^*_H \bigcap S^*_V \neq \varnothing$ simply does not hold.

However, Lemma 3.1 does help us understand what happened in the 2D problem of Ball and Magazine. In Figure 3.8 we have an instance of the circuit printing problem. We proceed to project the instance into two 1D instances and had their optimal eulerian trajectory drawn. We immediately notice that, in the 1D eulerian trajectory on the vertical axis, (1) the sequence does not matter: this eulerian graph allows full permutation of the request arcs to be optimal and (2) the optimal cost (excluding the cost of request arcs) is a constant $c(A_V)$. We call this 1D instance is of "free-permutation", which refers to the fact that any sequence is optimal. Now that S^*_V is full permutation, we know $S' = S^*_H \bigcap S^*_V = S^*_H \neq \varnothing$. This means that the solution to the horizontal 1D subproblem I_H alone, can determine the optimal sequences for the 2D problem. And this is exactly what happened in Ball and Magazine's circuit printing problem.

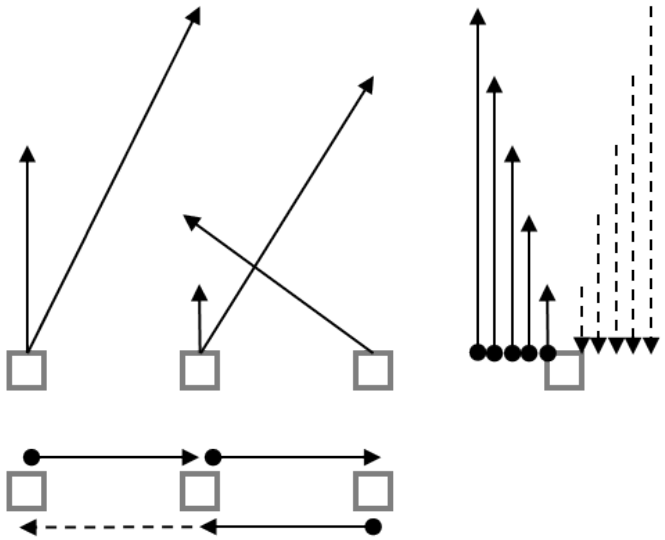

Figure 3.8: In circuit printing problem, the optimal solution to vertical subproblem yeilds full permutation of the request arcs

This observation implies that, for a 2D SCP in Manhattan norm, if we can identify that one of its two 1D subproblems is of free-permutation, then this 2D problem can be solved to optimal in $O(m + n\alpha(n))$ time by solving the other subproblem with Atallah and Kosaraju's algorithm. One can point out some extra similar examples, like those listed in Figure 3.9, and draw the conclusion that "all

the request arcs starts from/ ends at the same straight line" is what guarantees an free-permutation 1D subproblem. While this condition is indeed sufficient, but not necessary. There are some other free-permutation 1D instances that have different structures, though not obvious, like the one shown in Figure 3.10. Also note that all these examples are given in continuous space, it is true that the space does not necessarily need to be discrete or continuous. Now we answer the question of under what precise circumstances can we identify an free-permutation 1D instance.

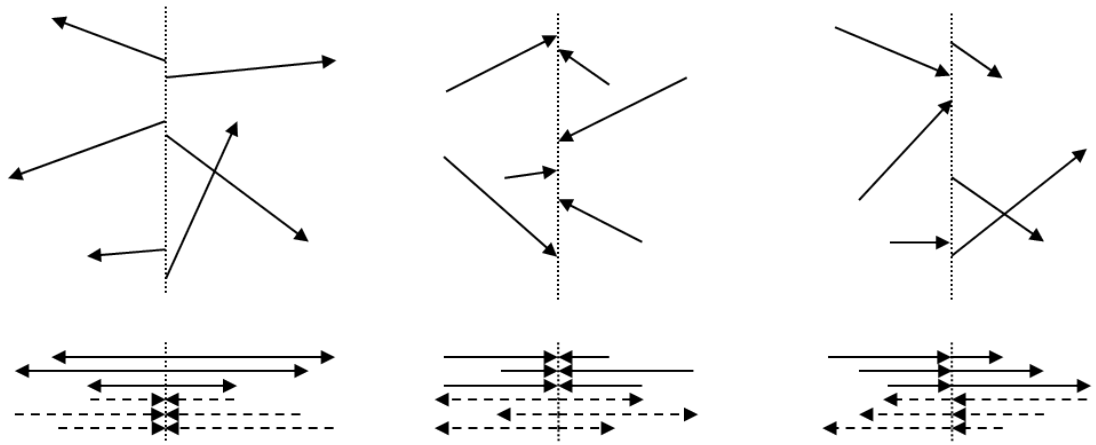

Figure 3.9: Some similar examples to Ball and Magazine's circuit printing problem and their free-permutation subproblems

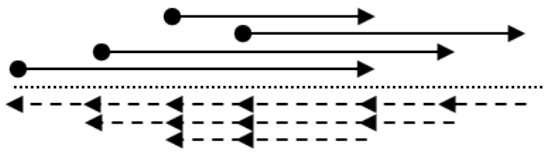

Figure 3.10: Another example of "free-permutation" 1D problem

First we introduce two necessary conditions for a free-permutation 1D instance.

Lemma 3.2 *Assume that the eulerian graph contains no disconnected sub-tours after the degree-balancing process (step (1) in Atahhah and Kosaraju's algorithm). If the instance is of free-permutation, then in its eulerian graph* T_H^**, for any arc* $i = (x_i^O, x_i^D) \in A_1$*, starting from* x_i^D*, there must exist an "augmenting path"* (x_i^D, x_j^O)*, a path that consists of solely augmenting arcs, to reach the pickup position* x_j^O *of any other arc* $j \neq i \in A_1$*.*

Proof: Under the no sub-tours assumption, T_H^* is unique. So any optimal sequence of the 1D instance can only be deduced from T_H^*. Suppose in T_H^*, $\exists i, j \in A_1$, that there is no such path as described above between x_i^D and x_j^O. Any solution sequence that has arc j right after arc i, such as $s = \{s_1, s_2, ...i, j, ..., s_{n-1}, s_n\}$, cannot be deduced from T_H^*. This contradicts the condition that the instance is of free-permutation. Thus we complete the proof for this lemma. □

Lemma 3.3 *Let $Y = \{y_1, y_2, ...y_m\}$ be the set of the coordinates of all the vertices of arcs in A_1. Since some of the elements in Y are equal, $m = |Y| \leq 2n$. On the axis, mark these coordinates in ascending order as $(y_1, y_2, ..., y_{m-1}, y_m)$. Under the no sub-tours assumption, if there exist augmenting arcs over any interval $[y_i, y_{i+1}], i = 1, ..., m-1$ of T_H^*, they are in the same direction, i.e. either to the left or to the right, not both (Figure 3.11).*

Proof: This lemma is an observation from the the degree-balancing process under the no sub-tour assumption. If the resulting graph T_H^* is optimal and eulerian after the degree-balancing, and there is an interval over which it has augmenting arcs in both directions. Then we can remove the pair of augmenting arcs over that interval without affecting the degree-balance or the connectivity of T_H^*, this contradicts that T_H^* is optimal. □

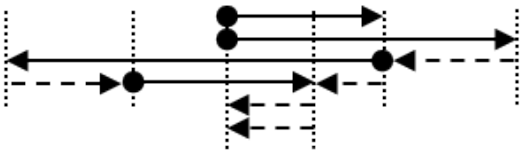

Figure 3.11: A demonstration of Lemma 3.3

Theorem 3.2 *In a 1D SCP problem instance, suppose its arc set $A_1 = \{a_i = (x_i^O, x_i^D), i = 1, 2, ..., n\}$ contains n arcs in total. Let $c(a), a \in A_1$ be the cost of arc a, and $c(A_1)$ be the total cost of all the arcs in A_1. If the resulting graph contains no disconnected sub-tours after degree-balancing, the 1D instance is of free-permutation, if and only if the arc set A_1 satisfies one or more of the following conditions 1)-6); otherwise, the 1D instance is of free-permutation if and only if the arc set A_1 satisfies one of the two conditions 7)-8) (Table 3.3 and Figure 3.12):*

1)	$x_i^O = x_j^O$	$\forall i, j = 1, 2, ..., n$
2)	$x_i^D = x_j^D$	$\forall i, j = 1, 2, ..., n$
3)	$x_i^O \leq x_j^D$	$\forall i, j = 1, 2, ..., n$ (the other direction symmetrical)
4)	$\max_{i=1,...,n;i\neq l}\{x_i^O\} \leq x_l^D \leq x_l^O \leq \min_{i=1,...,n;i\neq l}\{x_i^D\}$	$\exists$ only one $l \in \{1, 2, ..., n\}$
	$x_i^O \leq x_j^D$	$\forall i \neq l \vee j \neq l; i, j = 1, 2, ..., n$ (symmetrical)
5)	$x^O \leq x_l^O \leq \min_{i=1,...,n;i\neq l}\{x_i^D\}$	$\exists$ only one $l \in \{1, 2, ..., n\}$
	$x_i^O \leq x_i^D$, $x_i^O = x_j^O = x^O$	$\forall i, j \in \{1, 2, ..., n\}; i, j \neq l$ (symmetrical)
6)	$\max_{i=1,...,n;i\neq l}\{x_i^O\} \leq x_l^D \leq x^D$	$\exists$ only one $l \in \{1, 2, ..., n\}$
	$x_i^O \leq x_i^D$, $x_i^D = x_j^D = x^D$	$\forall i, j \in \{1, 2, ..., n\}; i, j \neq l$ (symmetrical)
7)	$x_l^O, x_l^D \leq x^O$	$\exists$ only one $l \in \{1, 2, ..., n\}$
	$x_i^O \leq x_i^D$, $x_i^O = x_j^O = x^O$	$\forall i, j \in \{1, 2, ..., n\}; i, j \neq l$ (symmetrical)
8)	$x_l^O, x_l^D \geq x^D$	$\exists$ only one $l \in \{1, 2, ..., n\}$
	$x_i^O \leq x_i^D$, $x_i^D = x_j^D = x^D$	$\forall i, j \in \{1, 2, ..., n\}; i, j \neq l$ (symmetrical)

Table 3.3: Six conditions in Theorem 3.3

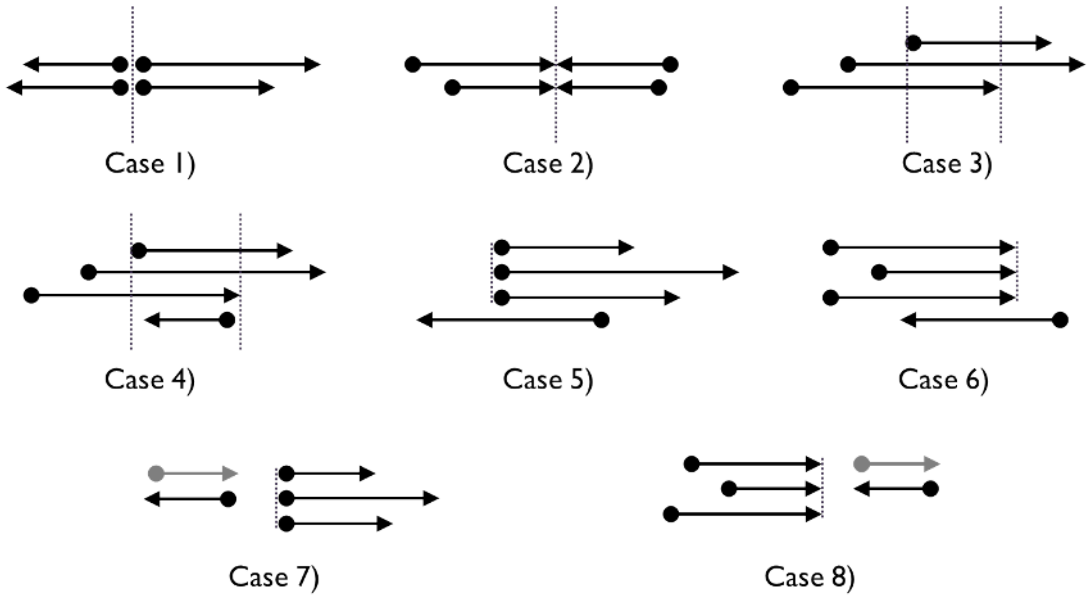

Figure 3.12: A simple demonstration of Theorem 3.2

3.3.1 Proof for Theorem 3.2

Proof:

Necessity: Pick any sequence of A_1, $seq = s_1, s_2, ..., s_n, s_i = 1, 2, ...n$. Then the cost of the tour completing the sequence is:

$$c(seq) = |x^O_{s_2} - x^D_{s_1}| + |x^O_{s_3} - x^D_{s_2}| + |x^O_{s_4} - x^D_{s_3}|... + |x^O_{s_n} - x^D_{s_{n-1}}| + |x^O_{s_1} - x^D_{s_n}| \quad (3.4)$$

Condition 1): If $x^O_i = x^O_j = x^O, \forall i, j = 1, 2, ..., n$, then equation (3.4) becomes:

$$\begin{aligned} c(seq) &= |x^O - x^D_{s_1}| + |x^O - x^D_{s_2}| + |x^O - x^D_{s_3}|... + |x^O - x^D_{s_{n-1}}| + |x^O - x^D_{s_n}| \\ &= |x^O_{s_1} - x^D_{s_1}| + |x^O_{s_2} - x^D_{s_2}| + |x^O_{s_3} - x^D_{s_3}|... + |x^O_{s_n-1} - x^D_{s_{n-1}}| + |x^O_{s_n} - x^D_{s_n}| \\ &= c(a_1) + c(a_2) + c(a_3)... + c(a_{n-1}) + c(a_n) \\ &= c(A_1); \end{aligned}$$

Condition 2): Can be proved similarly to Condition 1);

Condition 3): If $x_i^O \leq x_j^D, \forall i, j = 1, 2, ..., n$, then equation (3.4) becomes:

$$\begin{aligned} c(seq) &= (x_{s_1}^D - x_{s_2}^O) + (x_{s_2}^D - x_{s_3}^O) + (x_{s_3}^D - x_{s_4}^O)... + (x_{s_{n-1}}^D - x_{s_n}^O) + (x_{s_n}^D - x_{s_1}^O) \\ &= (x_{s_1}^D - x_{s_1}^O) + (x_{s_2}^D - x_{s_2}^O) + (x_{s_3}^D - x_{s_3}^O)... + (x_{s_{n-1}}^D - x_{s_{n-1}}^O) + (x_{s_n}^D - x_{s_n}^O) \\ &= c(a_1) + c(a_2) + c(a_3) + ... + c(a_{n-1}) + c(a_n) \\ &= c(A_1); \end{aligned}$$

Condition 4): Suppose that arc l is the s_lth in the sequence. Notice how Condition 4) is just Condition 3) with one modification: allowing one and only one arc to be in the opposite direction to the others. In Condition 4), since $\max_{i=1,...,n;i\neq l}\{x_i^O\} \leq x_l^D \leq x_l^O \leq \min_{i=1,...,n;i\neq l}\{x_i^D\}$ holds, the proof follows exactly the same as that of Condition 3) with only a minor difference:

$$\begin{aligned} c(seq) &= (x_{s_1}^D - x_{s_2}^O) + (x_{s_2}^D - x_{s_3}^O)... + (x_{s_{l-1}}^D - x_{s_l}^O) + (x_{s_l}^D - x_{s_{l+1}}^O)... + (x_{s_n}^D - x_{s_1}^O) \\ &= (x_{s_1}^D - x_{s_1}^O) + (x_{s_2}^D - x_{s_2}^O)... + (x_{s_l}^D - x_{s_l}^O)... + (x_{s_{n-1}}^O - x_{s_{n-1}}^D) + (x_{s_n}^O - x_{s_n}^D) \\ &= c(a_1) + c(a_2) + ... + c(a_{n-1}) + c(a_n) - c(a_l)(Constant) \end{aligned}$$

Condition 5): Under the conditions of Condition 5), suppose that arc r is the s_rth in the sequence, then equation (3.4) becomes:

$$\begin{aligned} c(seq) &= |x^O - x_{s_1}^D| + |x^O - x_{s_2}^D|... + |x_l^O - x_{s_{l-1}}^D| + |x_{s_{l+1}}^O - x_l^D|... + |x^O - x_{s_n}^D| \\ &= c(a_1) + c(a_2)... + x_{s_{l-1}}^D - x_l^O + x_{s_{l+1}}^O - x_l^D... + c(a_n) \\ &= c(a_1) + c(a_2)... + (x_{s_{l-1}}^D - x_{s_{l-1}}^O) - x_l^O - x_l^D + x_{s_{l+1}}^O + x_{s_{l-1}}^O... + c(a_n) \\ &= c(a_1) + c(a_2)... + c(a_{l-1}) - x_l^O - x_l^D + 2 \times x^O... + c(a_n)(Constant) \end{aligned}$$

Condition 6): The proof for Condition 6) is similar to Condition 5).

Hence we proved the necessity.

Sufficiency:

We will prove that, under the assumption that the resulting graph T_H is optimal and contains no sub-tours after the degree-balancing process, if the 1D instance is of free-permutation, then the arc set A_1 satisfies one or more of the above six conditions.

We divide all the 1D instances into 2 major categories: (1) Unidirectional: where all arcs are in only one direction; (2) Bidirectional: where there can exist arcs in both directions. In this proof, we will automatically assume that $n = |A_1| \geq 3$ since the cases where $n \leq 2$ are trivial.

(1) Unidirectional (Scenario U1):

When all arcs in A_1 are of the same direction (without loss of generality, we assume they are to the right), and are of free-permutation. Assume on the contrary that there exist arcs i, j that do not overlap, that is, $x_i^O \leq x_i^D < x_j^O \leq x_j^D$ (See Figure 3.13). According to Lemma 3.2, we know that augmenting paths (x_j^D, x_i^O) and (x_i^D, x_j^O) must both exist, and their directions must be the way shown in Figure 3.13. Then the interval between x_i^D and x_j^O clearly contradicts Lemma 3.3. Applied to both directions, we have the "overlapping property":

$$\begin{aligned} & x_i^O \leq x_j^D \quad \forall i, j = 1, 2, ..., n \\ \text{or} \quad & x_i^O \geq x_j^D \quad \forall i, j = 1, 2, ..., n \end{aligned}$$

These conclusions are exactly Case 3).

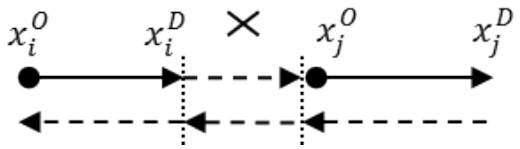

Figure 3.13: When all arcs are unidirectional, if they are of free-permutation, they must overlap

(2) Bidirectional:

When bidirectional arcs are allowed, there are a few scenarios to consider. Let A_L be the set of arcs that point to the left, A_R the set of arcs to the right,

$A_L \cup A_R = A_1$. $|A_L| = n_L, |A_R| = n_R$. Locate four critical coordinates on the axis for A_R:

$$\begin{aligned} x^O_{max} &= \max_{i \in A_R}\{x^O_i\} \\ x^O_{min} &= \min_{i \in A_R}\{x^O_i\} \\ x^D_{max} &= \max_{i \in A_R}\{x^D_i\} \\ x^D_{min} &= \min_{i \in A_R}\{x^D_i\} \end{aligned}$$

Since arcs in A_1 are of free-permutation, it can be inferred that, arcs in A_R and A_L are also of free-permutation. Hence the overlapping property must still hold for arcs in A_R and A_L respectively. For any pair of arcs $i, j \in A_R/A_L$, $[x^O_i, x^D_i] \cap [x^O_j, x^D_j] \neq \varnothing$. As a result, when $n \geq 3$, all situations can be divided into three mutually exclusive and collectively exhaustive scenarios. Since $n_L + n_R = n \geq 3$, without loss of generality, we will assume that $n_R \geq 2, n_L \geq 1$.

Scenario B1): Nonhomogeneous pickup and drop-off positions in A_R

In this scenario, $\exists i, j \in A_R, x^O_i \neq x^O_j$ and $\exists a, b \in A_R, x^D_a \neq x^D_b$. We will prove that in this scenario, under the assumption that A_1 is of free-permutation, if there exist arcs in A_L, then $n_L = 1$, and this only arc $l = (x^O_l, x^D_l)$ satisfies $x^O_{max} \leq x^D_l \leq x^O_l \leq x^D_{min}$.

Note the fact that, according to Lemma 3.2, for the arc in A_R with the largest drop-off position (x^D_{max}) to reach all other arcs in A_R, and for all the other arcs in A_R to reach the arc with the smallest pickup position (x^O_{min}), there will always exist an augmenting path (x^D_{max}, x^O_{min}), see Figure 3.14. This augmenting path alone enforces very strong limitations on arcs in A_L.

Assume on the contrary, there exists arc $l = (x^O_l, x^D_l) \in A_L$, $x^O_l > x^D_{min}$. See Figure 3.14 (a), for the arc in A_R with the smallest drop-off position to reach arc l, according to Lemma 3.2 there should be an augmenting path (x^D_{min}, x^O_l). Now over the interval $[x^D_{min}, x^O_l]$ there are opposing augmenting paths, this contradicts Lemma 3.3. Thus we have $x^O_l \leq x^D_{min}$.

Again assume on the contrary, there exists arc $l \in A_L$, $x_l^D < x_{max}^O$. In Figure 3.14 (b), for arc l to reach the arc with the largest pickup position in A_R, according to Lemma 3.2, there should exist an augmenting path (x_l^D, x_{max}^O). Now over the interval $[x_l^D, x_{max}^O]$, there exist augmenting paths in both directions, this contradicts Lemma 3.3. We have $x_{max}^O \leq x_l^D$.

Furthermore, assume on the contrary, that $n_L \geq 2$. Demonstrated in Figure 3.14 (c), pick any pair of arcs $l, m \in A_L$ (they will overlap), for them to reach each other, say, m to reach l, there should be an augmenting path (x_m^D, x_l^O). Over the interval $[x_m^D, x_l^O]$ there are augmenting paths in both directions, hence contradicts Lemma 3.3. So we have $n_L = 1$.

Thus We have deduced Condition 4).

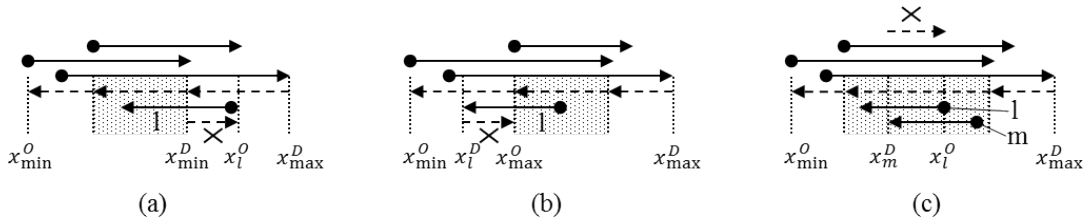

Figure 3.14: In Scenario B1), Condition 4) can be deduced

Scenario B2): Homogeneous pickup positions in A_R

In this scenario, all arcs in A_R share the same pickup position x^O (See Figure 3.15 for a comprehensive demonstration). Now we consider the possible positions and the number of arcs in A_L.

Case a): $n_L = 1$

In this case, we will prove that, under the conditions that A_1 is of free-permutation and all arcs in A_R has homogeneous pickup positions x^O. If there exists any arc $l \in A_L$, then $x^O \leq x_l^O \leq x_{min}^D$.

Assume on the contrary that there exists an arc $l = (x_l^O, x_l^D) \in A_L$, $x_l^O > x_{min}^D$ or $x_l^O < x^O$. When $x_l^O > x_{min}^D$, as interpreted in Figure 3.15 (a), according to Lemma 3.2, for the arc that has the minimum drop-off coordinate in A_R, m, to reach l, there exists an augmenting path (x_{min}^D, x_l^O). On the other hand, for the arc

that has the maximum drop-off coordinate in A_R, n to reach any other arc in A_R, there exists an augmenting path $(x^D_{max}, x^O_{min} = x^O)$. Over the interval $[x^D_{min}, x^O_l]$, now we have augmenting arcs in both directions, thus contradicts Lemma 3.3.

When $x^O_l < x^O$ (Figure 3.15 (b)), according to Lemma 3.2, for arc l to reach any arc in A_R, there must exist an augmenting path (x^D_l, x^O). On the other hand, for arcs in A_R to reach l, there also must exist an augmenting path (x^D_{min}, x^O_l). We see over the interval $[x^O_l, x^O]$, there are opposing augmenting paths that violate Lemma 3.3.

Thus we have the contradiction.

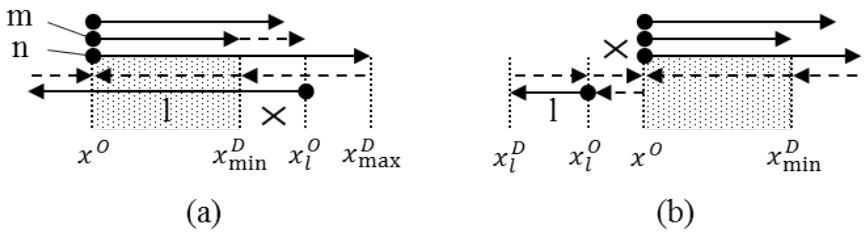

Figure 3.15: Scenario B2), when $n_L = 1$, Condition 1) can be deduced

Case b): $n_L \geq 2$

In this case, we will prove that, if A_1 is of free-permutation and all arcs in A_R has homogeneous pickup positions x^O. If there exists $n_L \geq 2$ arcs in A_L, then $x^O_l = x^O, \forall l \in A_L$.

Assume on the contrary that $\exists l = (x^O_l, x^D_l) \in A_L, x^O_l \neq x^O$. From Case a) we already know that $x^O_l \in [x^O, x^D_{min}]$, so $x^O_l \neq x^O$ means that $x^O_l \in (x^O, x^D_{min}]$. We can identify the arc in A_L with the largest pickup position, $n = (x^O_n, x^D_n), x^O_n = \max_{i \in A_L}\{x^O_i\}$. Since x^O_n is the largest, we have $x^D_l \leq x^O_l \leq x^O_n \forall l \in A_L$. Demonstrated in Figure 3.16, according to Lemma 3.2 for the same reason in Case a), there exists augmenting path (x^D_{max}, x^O); on the other hand, for any other arc $m \in A_L$ to reach arc n, there should be an augmenting path (x^D_m, x^O_n). Over the interval $[x^D_m, x^O_n]$, we will observe opposing augmenting paths that violate Lemma 3.3. Thus we have the contradiction.

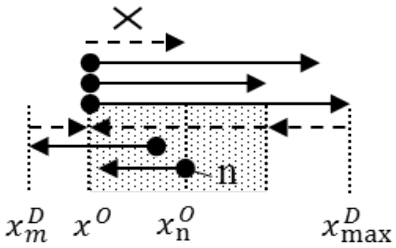

Figure 3.16: In Scenario B2), when $n_L \geq 2$, Condition 1) can be deduced

Case a) and b) combined, we have exactly deduced Conditions 1) and 5).

Scenario B3): Homogeneous drop-off positions in A_R

Pickup and drop-off locations are essentially symmetrical. Thus in this scenario, with similar proof in Scenario B2), we can deduce Conditions 2) and 6).

Scenario U1, B1, B2, B3 are mutually exclusive and collectively exhaustive, thus beginning from the condition of free-permutation, we have deduced all six conditions of Theorem 3.2. This completes the proof for sufficiency. □

Chapter 4

Nodes Formulation and GRID Algorithm

Our work in the previous section only identifies a small subset of polynomial-time solvable EESCP, with certain arc patterns. In most cases, as we have shown, the problem is NP-Complete and its optimal solutions are resource-demanding to obtain. A common exact solution is to formulate and solve it as an ATSP problem, which has an exponential time complexity. In this section, we propose a new exact formulation for EESCP based on its fixed network with finite number of vertices, the solving time of which does not grow exponentially with the number of requests in the instance. And then we introduce two approximation algorithms GRID-L and GRID-S.

Notation in the following sections:

Sets

NB_i	Set of neighboring cells of cell i
NB_i^H	The horizontal neighboring cells of cell i
NB_i^V	The vertical neighboring cells of cell i
R	Set of all the requests
A	Set of all the arcs representing the requests
V	Set of all the vertices on the grid
E	Set of all the edges on the grid
S	$S = \{i \in V \mid G_i > 0\}$
D	$D = \{i \in V \mid G_i < 0\}$
U	U is the set of vertices that are visited by at least one arc in A, whose final net in-degree is zero
P	$P = S \cup D \cup U$ is the set of all the vertices that are visited at least once in the instance

Parameters

M	The total number of all requests
N_V	The number of rows of the storage rack
N_H	The number columns of the storage rack
N	$N = N_V \times N_H$, is the total number of storage cells on the rack
K	The number of connected components
c_H, c_V	The energy consumption for the stacker crane vehicle to perform a single move horizontally/ vertically when empty-loaded
G_i	The net in-degree of cell i induced by the arcs in A

a_{ij}	The number of arcs starting from vertex i to vertex j in A, $\forall i, j \in P$
c_{ij}	The cost of arc (i, j)
Decision Variables	
x_{ij}/y_{ij}	$x_{ij}/y_{ij} \in \mathbb{N}$, $x_{ij}/y_{ij} = n$ means that there would be n moves from cell i to cell j in the final vehicle tour
z_{ij}	$z_{ij} = n$ if there are n units of commodity flow through arc (i, j)
Others	
C^*	The optimal tour cost for the *EESCP* instance
C_A	The cost of the given requests in A

4.1 A New Exact Formulation

Given an instance $G(V, E, A)$For each vertex in V, do:

1) For each cell i on the grid, calculate its "net in-degree" G_i by subtracting the number of requests that origin at cell i from the number of requests that end at cell i;

2) Define set $S = \{i \in V | G_i > 0\}$ and $D = \{i \in V | G_i < 0\}$. Next we observe that all the unbalanced vertices are included in S and D. Define a set of vertices U, $u \in U$ is a vertex that is visited by at least one arc in A, but its final net in-degree is zero;

3) Let $S \cup D \cup U = P$, then P is the set of all the vertices that are relevant (visited at least once) in this problem instance;

4) Let a_{ij} be the number of arcs starting from vertex i to vertex j in A, $\forall i, j \in P$. Also introduce another set of "commodity flow" variable $Z = \{z_{ij} | z_{ij} = 1, 2..., |P|\}$ for sub-tour elimination.

5) Let c_{ij} be the cost of arc (i, j). For general graphs, c_{ij} would often need to be solved by a shortest path algorithm, and $c_{ij} = c_{ji}$. In a solid grid, however, c_{ij} is easy to obtain;

6) Notice that to make every vertex in P degree-balanced, is to find a single tour to "transport" positive in-degrees from vertices in $S \cup U$ to vertices in $D \cup U$, with minimum energy cost.

Then the formulation is as follows:

Decision Variables:

$Y = \{y_{ij} \in \mathbb{N}\}$, $y_{ij} = n$ if the solution contains n augementing arcs starting from vertex i to j, $\forall i, j \in P$

$Z = \{z_{ij} | z_{ij} = 1, 2..., |P|\}$, $z_{ij} = n$ if there are n units of commodity flow through arc (i, j)

Objective Function:

$$\min . \sum_{i \in P} \sum_{j \in P} c_{ij} y_{ij} \tag{4.1}$$

Constraints:

$$\sum_{i \in S \cup U} y_{ij} = -G_j \qquad \forall j \in D \tag{4.2}$$

$$\sum_{j \in D \cup U} y_{ij} = G_i \qquad \forall i \in S \tag{4.3}$$

$$\sum_{j \in P} y_{ij} = 0 \qquad \forall i \in D \tag{4.4}$$

$$\sum_{i \in P} y_{ij} = 0 \qquad \forall j \in S \tag{4.5}$$

$$\sum_{j \in D \cup U, j \neq i} y_{ij} = \sum_{j \in S \cup U, j \neq i} y_{ji} \qquad \forall i \in U \tag{4.6}$$

$$\sum_{j \in P} z_{ji} - \sum_{j \in P} z_{ij} = 1 \qquad \forall i \in P, i \neq 1 \tag{4.7}$$

$$\sum_{j \in P, j \neq 1} z_{1j} = |P| \tag{4.8}$$

$$z_{ij} \geq 1 \ \ if \ \ a_{ij} \geq 1 \ \ or \ \ y_{ij} \geq 1 \qquad \forall i, j \in P \tag{4.9}$$

$$y_{ij} \geq 0 \quad \forall i \in S \cup U, j \in D \cup U$$

We call this formulation "Nodes Formulation", for the scale of this formulation is bounded by the number of vertices of the grid network. This formulation is a mixed integer programming. It has $2|P|^2$ decision variables, $2N^2$ at most. It has roughly $2P^2 + 2P$ constraints, $2N^2 + 2N$ at most. The objective function is to use arcs with the least amount of cost to construct an eulerian graph. The objective function (4.1) is to find a set of augmenting arcs with the minimum cost to be combined with A to form an eulerian graph. a_{ij} represents the part of the graph that is given by the request set A; Constraint (4.2) says that the vertices with a negative net in-degree is going to be balanced by the vertices with positive or zero net in-degrees; Constraint (4.3) says that the vertices with a positive net in-degree is going to supply its surplus to those with a negative or zero in-degree. Constraint (4.6) means that for vertices in U, the number of arcs that go in must equal the number that go out; The rest of the constraints (4.7) to (4.9) are using "flow variables" to guarantee there will be no sub-tour in the solution [89].

Note that in Constraints (4.4) and (4.5), we have made a claim that, in the optimal eulerian graph, there would neither necessarily be an augmenting arc starting from a vertex with a negative net in-degree, nor one ending at a vertex already with a positive net in-degree. We provide a brief proof that this necessary condition is true.

Theorem 4.3 *In an EESCP instance $I = G(V, E, A)$, let $P \subseteqq V$ be the set of vertices that are visited (i.e. being the pickup or drop-off node) by at least one arc in A. Calculate the net in-degree G_i for each vertex $i \in P$, and categorize them into three sets: $S = \{i \in P | G_i > 0\}$, $D = \{i \in P | G_i < 0\}$, $U = \{i \in P | G_i = 0\}$. Then there exists an optimal eulerian graph $G(A, A')$ for this instance I, the augmenting arcs A' of which satisfies that: $\forall (i, j) \in A'$, $i \notin D$, $j \notin S$.*

Proof: First we prove the $i \notin D$ part. In the optimal eulerian graph $G(A, A')$, identify any vertex $v_1 \in D$, the net in-degree of which is negative before adding

the augmenting arcs. As shown in Figure 4.1 (a), since it is in D, there will be at least one augmenting arc r ending at v_1. Assume on the contrary, that in the optimal eulerian graph, there must exist an augmenting arc n that starts from v_1, and ends at some other vertex v_2. For the graph to be eulerian, there must exist an augmenting arc m, from some other vertex v_3, that ends in v_1 (Figure 4.1 (a)). We can replace m, n with a single augmenting arc b that directly goes from v_2 to v_3, without affecting the degree-balance (Figure 4.1 (b)). On a grid graph, even in Manhattan norm, the triangular inequality holds, we have that $c(b) \leq c(m)+c(n)$. So if replacing m, n with b does not break the connectivity of the graph, we already have the contradiction.

Now consider the case where replacing m, n with b does break the connectivity of $G(A, A')$. In this case, the topology of $G(A, A')$ must be in the form shown in Figure 4.1 (c), where m, n are the only arcs that linking v_2, v_3 to v_1, and the vertices and arcs that are not directly associated with v_1, v_2, v_3 are represented by black boxes. This is true because if there are other arcs between either v_1, v_2 or v_1, v_3, replacing m, n with b would not break connectivity in the first place. As we already understand the existence of arc r, in Figure 4.1 (d), we see that in this situation, we can always replace r, n with a direct augmenting arc d, without affecting either the degree-balance or the connectivity of the optimal eulerian graph. Since we have $c(d) \leq c(r) + c(n)$, hence the contradiction.

The $j \notin S$ part can be proved similarly. This completes the proof. □

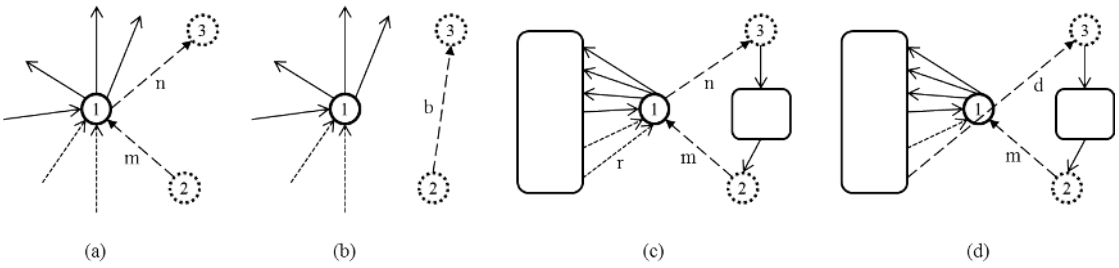

Figure 4.1: No augmenting arcs from a vertex in D in the optimal eulerian graph, due to triangular inequality

On a given fixed network, the computation time for solving the new formulation does not grow exponentially with the number of requests in A like that of ATSP

formulation, but is instead bounded, as shown later in Section 4.8. However, for an instance with a large underlying network, it can still be very time consuming to solve the instance to optimal. So we turn to approximation algorithms with both good time complexity and theoretical bound. Frederickson, Hecht et al. proposed the algorithm *CRANE* to solve general SCP, which consists of two polynomial-time approximation algorithms, *LARGEARCS* and *SMALLARCS*, and it has the best proven worst-case bound to date [2]. Treleaven, Pavone et al. recently combined Bipartite Matching with Euclidean SCP and provided a asymptotically optimal (almost surely) class of algorithms with the complexity of $O(n^{2+\varepsilon})$ [20].

Next we will propose two algorithms for EESCP, GRID-L and GRID-S, that utilize the underlying grid network. We will show that, when the grid size is fixed, GRID-L is asymptotically optimal, and has a complexity of $O(M)$, M being the number of requests. GRID-S makes up for situations where GRID-L has a large optimality gap, and also takes the fixed grid size into account. When these two combined, they provide a worst-case theoretical bound of 5/3.

4.2 GRID-L

GRID-L has a similar structure to that of *LARGEARCS*, which consists of two steps, degree-balancing and sub-tours connecting. In the $SPLICE$ algorithm of Treleaven, Pavone et al., they used Bipartite Matching ($O(n^{2+\varepsilon})$ time complexity) to replace the degree-balancing process, n being the number of requests. This complexity is also the complexity for their algorithm, *SPLICE*. However in GRID-L, with the introduction of the grid network, we use two network flow models to complete the degree-balancing, making the time complexity of this process almost irrelevant to number of requests. One of them is a Minimum Cost Flow model, which has a better optimality gap when used in algorithm, while the other is Transportation Problem model, which has smaller computation time in general. The structure of GRID-L is as follows: First it produces disconnected sub-tours by degree-balancing the given arcs in A (solving a linear programming model), then

use Minimum Spanning Tree algorithm to connect resulting sub-tours. Finally the algorithm ends by identifying a feasible Euler tour in the constructed eulerian graph, and a post-process is called to eliminate the unnecessary detours.

4.2.1 Minimum Cost Flow Formulation

We use the Minimum Cost Flow Problem formulation on the grid network for a relaxed version of EESCP as the first step by leaving out the sub-tour elimination constraints in the formulation. Observe that when Manhattan norm is used, no matter what the final tour is, all the arcs that represent empty-loaded movements must be along the edges of the grid; and that if the final tour is degree-balanced, for every vertex on the grid, the number of arcs goes into it must equal the number that goes out. Thus gives us the formulation as follows:

1) The problem instance has M requests in total.

2) The pickup and drop-off vertices of each request are both generated randomly over the grid. $R = \{1, 2, 3, ..., M\}$ is the set of all the requests.

3) Consider the storage rack as a $N_V \times N_H$ grid. So the grid contains $|V| = N = N_V \times N_H$ rectangular cells in total.

4) Define "neighboring cell" NB_i for each cell $i \in V$: a cell on the grid has normally four neighboring cell that are directly adjacent to itself in four directions. Cells along the four edges of the grid have 3 neighboring cells while those at the four corners have only 2. Since moving to a horizontal adjacent neighboring cell incurs different energy consumption from moving to a vertical adjacent one, we denote the horizontal and vertical neighboring cells with NB_i^H, NB_i^V respectively.

5) Define a "move" as a unit movement of the stacker crane vehicle from one cell to one of its neighboring cells.

6) Cost parameter c_H, c_V equal the respective energy consumption for the stacker crane to perform a single move horizontally and vertically when empty-loaded.

7) Define G_i for each cell $i \in V$, G_i is the net in-degree of cell i induced by the arcs in A. It is calculated by subtracting the number of requests that origin at cell i from the number of requests that end at cell i.

Then the relaxed problem will be presented as follows:

Decision Variables:

$x_{ij} \in \mathbb{N}$, $x_{ij} = n$ means that there would be n moves from cell i to cell j in the solution

Objective Function:

$$\min . \sum_{i=1,2...N} \sum_{j \in NB_i^H} c_H x_{ij} + \sum_{i=1,2...N} \sum_{j \in NB_i^V} c_V x_{ij}$$

Constraints:

$$\sum_{j \in NB_i} x_{ij} - \sum_{j \in NB_i} x_{ji} = G_i \qquad \forall i \in V \tag{4.10}$$

$$x_{ij} = 0 \qquad \forall j \notin NB_i, \forall i \in V \tag{4.11}$$

$$x_{ij} \geq 0 \qquad \forall i, j \in V \tag{4.12}$$

We call this formulation GRID Formulation (GF) for the fact that the decision variables are set on the grid network. Also note that this formulation works on any type of network. Its solution, A', is the set of added unit augmenting arcs (moves) to make a degree-balanced graph. This formulation, objective and constraints combined, means to add the fewest moves possible—along the grid edges both horizontally and vertically—to make the resulting graph (V, A, A') degree-balanced. Constraint (4.10) means every vertex on the grid must be strictly degree-balanced. Constraints (4.11) and (4.12) mean that every move is unit and confined to the edges of the grid; and that for each vertex on the grid, we only consider its relationship with its neighboring cells.

This formulation is a linear integer programming; it has $2N_H(N_V - 1) + 2N_V(N_H - 1)$ nonnegative integer variables and N constraints. The matrix (M_A) specified by constraint (4.10) is actually totally unimodular. To see this, one need only see that M_A is the incidence matrix of the grid graph. Since the grid graph is degree-balanced, according to [90] and [91], M_A must be totally unimodular. Thus the integer decision variables in this formulation can be relaxed to continuous variables and solved by simplex methods and Linear Programming solvers. We propose the complete GRID-L in Table 4.1.

Algorithm: GRID-L(GF)	
Step 1.	Formulate the problem using GF;
Step 2.	Solve the formulation with LP solver. For each $x_{ij} = k$ in the solution, add k unit arcs from cell i to cell j. Let A' be the set of arcs specified by the solution to GF, and $S = \{S_1, S_2...S_p\}$ the connected sub-tours in graph (V, A, A');
Step 3.	Define a undirected network (N_S, E_S), where: $N_S = 1, 2, ...p$ is a node set, each node stands for a sub-tour in S, $E_S = \{(i, j) : i, j \in N_S\}$, $w_{ij} = min\{e_{ab} : a \in S_i, b \in S_j\}$, where e_{ab} is the energy cost associated with arc (a, b);
Step 4.	Find the minimum spanning tree in (N_S, E_S). Let $\bar{E}$ be the solution. For each undirected edge $(a, b) \in \bar{E}$, generate two directed arcs (a, b) and (b, a). Let T be the set of these arcs;
Step 5.	(V, A, A', T) is now strongly connected and degree-balanced. Find an Euler tour to traverse it by means of any Euler tour construction algorithm;
Step 6.	Post-process to eliminate unnecessary detours.

Table 4.1: Algorithm: GRID-L(GF)

GRID-L(GF) is an approximate algorithm thus does not guarantee optimality. However, it stands out in that:

- Degree-balancing is completed by solving the network flow model GF. Since GF is independent of the number of the requests M, unlike *LARGEARCS*

or *SPLICE*, the computational time of Step 1 does not grow with number of requests at all (fixed by the size of grid) which fact further contributes to the linear time complexity of GRID-L (See Section 4.3);

- In GF we break any empty-loaded trajectory of the vehicle into unit moves, which fact further benefits Step 3. In *LARGEARCS*, after the degree-balancing, the linking of sub-tours can only happen at a pickup or delivery location. However, the solution of *GF* gives a lot more options for the linking algorithm to choose from—the linking of sub-tours can happen at any vertex that A' bypasses, which is quite intuitive since all arcs in A' stand for empty-loaded vehicle movement. This improvement results in smaller optimality gaps;
- GRID-L is asymptotically optimal (see the proof in Section 4.4);
- The number of decision variables can be further reduced by limiting the movement to a Hanan grid [92] [93], and removing redundant vertices in the problem instance.

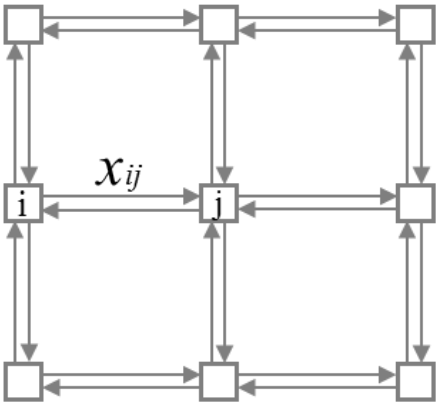

Figure 4.2: A 3×3 grid limit decision variables number to 24

4.2.2 Transportation Problem Formulation

Some might argue that the GRID formulation can waste computation resource when the number of requests M is small while the grid is large. We propose another formulation that is more suitable for a small number of requests:

1) For each cell i on the grid, calculate its "net in-degree" G_i by subtracting the number of requests that origin at cell i from the number of requests that end at cell i;

2) Define set $S = \{i \in V | G_i > 0\}$ and $D = \{i \in V | G_i < 0\}$. Next we observe that all the unbalanced vertices are included in S and D, and that to make them degree-balanced is to find a solution to "transport" positive in-degrees from vertices in S to vertices in D, with minimum energy cost.

This defines a Transportation Problem relaxation for the original problem. The details are as follows:

Decision Variables:

$x_{ij} \in \mathbb{N}$ is the number of augmenting arcs that start from vertex i and end in vertex j, $\forall i \in S, j \in D$

Objective Function:

$$\min . \sum_{i \in S} \sum_{j \in D} c_{ij} x_{ij}$$

Constraints:

$$\sum_{i \in S} x_{ij} = -G_j \qquad \forall j \in D \tag{4.13}$$

$$\sum_{j \in D} x_{ij} = G_i \qquad \forall i \in S \tag{4.14}$$

$$x_{ij} \geq 0 \qquad \forall i \in S, j \in D$$

We call this formulation Transportation Formulation (TF). It is only a relaxation because again this formulation does not include sub-tour elimination constraints. In this formulation, the decision variables are different from the GF in the sense that each x_{ij} is not restricted to its neighboring cells, and c_{ij} must be calculated for each $i \in S$ and $j \in D$. Constraint (4.13) means that for every cell that has a negative net in-degree, we will "demand" exactly the same amount of augmented movements that end in it from "suppliers". Constraint (4.14) means

that for each cell that has a positive net in-degree, we will "supply" all of its net in-degree to those "customers" with negative net in-degrees. In essence, this formulation is just another way to describe "degree-balance" necessary condition. TF has the following features:

- It has $|S| \times |D|$ positive integer variables, small when M is small, but can increase to as much as $N^2/4$, and it only has $|S| + |D|$ constraints in total.
- In the current problem setting, where c_{ij} is very straightforward to determine, this approach is very advantageous. But when trying to extend this method onto a more general network, determining every c_{ij} is actually a Shortest Path problem and should be dealt with accordingly.
- When used in algorithm, the algorithm is still asymptotically optimal in the same sense as GRID-L(GF) and can be proved similarly.
- The algorithm run-time also scales linearly with number of requests M.

The algorithm is as in Table 4.2:

Algorithm: GRID-L(TF)
Step 1. Formulate the relaxed problem with TF;
Step 2. Follow Step 2 to 6 in GRID-L(GF)

Table 4.2: Algorithm: GRID-L(TF)

A final word on GRID-L(GF) and GRID-L(TF) is that, they provide similar efficiency. They both have linear time complexity with respect to the number of requests M. GRID-L(GF) has a smaller optimality gap than that of GRID-L(TF), for it searches a larger space for better solution for the MST algorithm (all augmenting arcs in unit length). For the same reason, GRID-L(GF) generally has a longer computation time, for its input size to the Euler Tour finding algorithm is usually a lot larger than than of GRID-L(TF).

4.2.3 1D Bound on 2D Problems

Suppose we have a 2D EESCP instance I, and its two projections instances, I_H, I_V. In Chapter 3, we discussed about the relationship between T_H^*, T_V^*, the optimal eulerian graph of I_H, I_V, and $P_H(T_{s^*}), P_V(T_{s^*})$, the horizontal and vertical projections of the 2D optimal eulerian graph for I. We understand that in term of cost, we have:

$$c(T_H^*) \leq P_H(T_{s^*}); c(T_V^*) \leq P_V(T_{s^*}) \tag{4.15}$$

Equation (4.15) means that T_H^* and T_V^* provide lower bounds for the optimal 2D solution. This inspires us to use the solution of 1D instances as constraints in the 2D algorithm to help improve its performance. In fact, the degree-balancing procedure in algorithm GRID often results in sub-tours, whose 1D projections often are also disconnected; On the other hand, the solution to the projection 1D instances are always connected. Intuitively, we can use the latter to eliminate some obvious sub-tours for the former. This concept is demonstrated in Figure 4.3: the degree-balancing procedures result in sub-tours, whose projections are visually disconnected. If we can feed the extra information provided by T_H^* and T_V^* (highlighted with a box) to the degree-balancing procedures, some of the sub-tours can be avoided.

As demonstrated in Figure 4.3, the only extra information that T_H^* and T_V^* can provide is in fact produced by the Minimum Spanning Tree connecting procedures in the 1D algorithm from Atallah and Kosaraju (See Chapter 3). This information can be translated into valid constraints for the 2D problem. See in Figure 4.4, as the "shadow" T_H^* indicates that in the middle interval, there should be at least one augmenting arc goes from left to right, and vice versa. Using Transportation Formulation, this can be translated into:

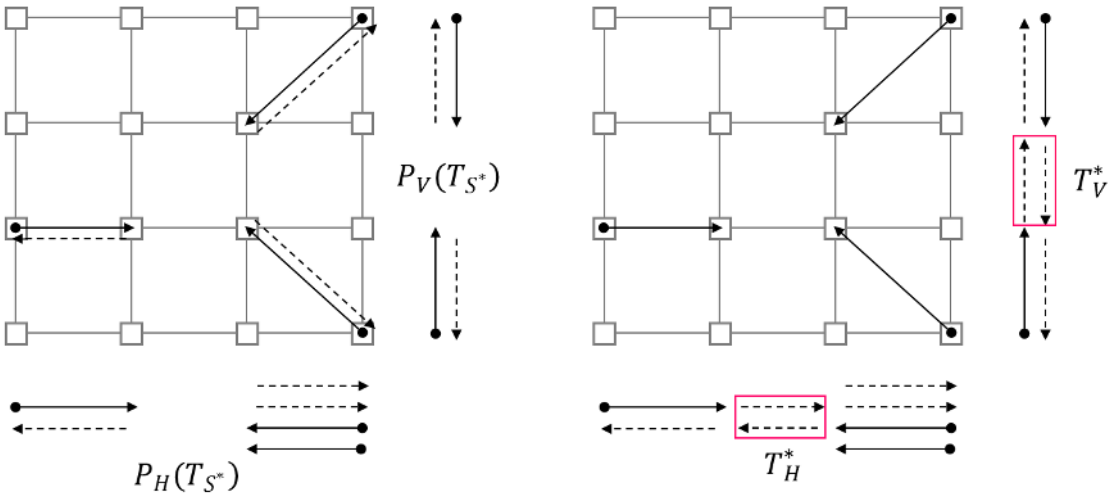

Figure 4.3: T_H^*,T_V^* provide extra information for the 2D problem

$$x_{d_1p_3} + x_{d_1p_2} \geq 1$$

$$x_{d_3p_1} + x_{d_2p_1} \geq 1$$

Likewise, the extra information T_V^* provides can be translated to:

$$x_{d_1p_3} + x_{d_2p_3} \geq 1$$

$$x_{d_3p_1} + x_{d_3p_2} \geq 1$$

With the constraints above, the degree-balancing procedures can directly jump to the true optimal solutions without sub-tours as shown in Figure 4.4. In practical simulations, with these extra constraints, the degree-balancing procedures produce a lot less sub-tours than without, improving the optimality gap tremendously. However, there are two points to take note of: (1) These constraints are only valid if T_H^* or T_V^* is unique, i.e., T_H^* and T_V^* fail to provide precise constraints when there are multiple optimal eulerian graphs for 1D projection sub-problems. In fact, occasionally these constraints can even bring the algorithm further away from the true optimal. (2) In instances with large number of requests arcs, there are often

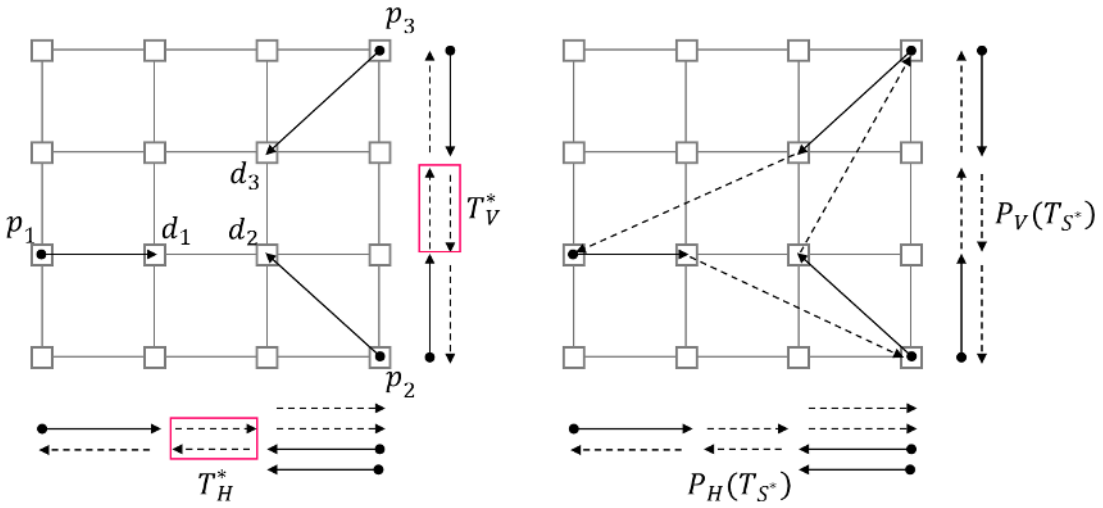

Figure 4.4: T_H^*,T_V^* can be translated into constraints to eliminate some obvious sub-tours in the 2D algorithm

no such empty intervals, reducing these constraints insignificant.

4.3 Complexity of GRID-L

First we examine how the computation time of GRID-L grows linearly with number of requests M. To see this, we understand that this algorithm consists of three procedures:

1) Degree-balancing (DB): solving a linear programming;

2) Linking sub-tours (LS): use Minimum Spanning Tree algorithm;

3) Tour construction (TC): any Euler tour finding algorithm.

Table 4.3 show the complexity for each part of GRID-L algorithm under different formulation. Recall that N is the number of vertices on the grid and M the number of requests. These two tables show clearly that the total run-time of GRID-L grows linearly with the number of requests. The observation is that, by setting the decision variables on the fixed grid graph rather than individual requests: the DB run-time is fixed by, or, in the case of TF, bounded by N; the LS run-time is averagely fixed by N; and that the TC run-time grows linearly with M and will eventually outweigh the sum of the former two, becoming dominant.

Procedure	Time Complexity	
	GRID-L(GF)	**GRID-L(TF)**
DB	$O(N^3)$	$\min\{O(N^6), O(M^6)\}$
LS	$O(N)$ in worst case	$O(N)$ in worst case
TC	$O(M \cdot N)$ in worst case	$O(M \cdot N)$ in worst case

Table 4.3: The time complexity of GRID-L

4.4 Asymptotic Optimality of GRID-L

In this section we will prove the asymptotic optimality for GRID-L.

In GRID-L, the cost of its solution is decided by DB and Linking LS procedures. The solution after DB process is the solution to GF, A' . At this stage (V, A, A') is already degree-balanced but may contain sub-tours. $c(A') + c(A)$ is the total cost.

Since GF is a relaxation of the original SCP problem, $c(A') + c(A)$ is a lower bound for the original optimal value C^*(the cost of A included). The second part LS is the linking process that links all the sub-tours in (V, A, A'). Its solution is T at the cost of $c(T)$. The objective value algorithm GRID-L yields is $C_L = c(A') + c(A) + c(T)$. We will first show that $c(T)$ is bounded.

Lemma 4.4 *For a given graph $G = (V, E)$, a fixed set of linking arcs can be found to make any sub-tours on G connected, which means that for any given graph, $c(T)$ is always bounded.*

Proof: Suppose $T(G)$ is an arbitrary spanning tree of G. Then replace every edge in $T(G)$ with a pair of arcs in opposite direction. We call the resulting graph $T_A(G)$. Note that $T_A(G)$ is a directed graph that connects every vertex in G. Then we combine (V, A, A') and $T_A(G)$, the resulting graph $(V, A, A', T_A(G))$ is a connected graph with the introduction of $T_A(G)$. Since T is obtained from the minimum spanning tree of a subset of vertices of G, we know $c(T)$ is bounded, and should not exceed $c(T_A(G))$. □

Theorem 4.4 *Algorithm GRID-L is asymptotically optimal when the number of requests* $M \to +\infty$

Proof: The proof is quite straightforward, we already know that $c(A') + c(A)$ is a lower bound for the optimal value C^*, thus $c(A') + c(A) \leq C^*$ holds.

(V, A, A', T) is a feasible solution for the original problem, so $C^* \leq c(A') + c(A) + c(T)$ must hold. Thus

$$\begin{aligned} c(A') + c(A) \quad &\leq C^* \leq c(A') + c(A) + c(T) \\ \Rightarrow \frac{c(A') + c(A)}{c(A') + c(A) + c(T)} &\leq \frac{C^*}{C_L} \leq 1 \end{aligned}$$

When $M \to +\infty$, $c(A') + c(A) \to +\infty$ while $c(T)$ is bounded(Lemma 4.4), so

$$\lim_{M \to +\infty} \frac{c(A') + c(A)}{c(A') + c(A) + c(T)} = 1$$

We have:

$$\lim_{M \to +\infty} \frac{C_L}{C^*} = 1$$

This completes the proof. □

In fact, $c(T)$ is far less than $c(T_A(G))$ in most cases, thus the optimality gap can be narrowed down to 5% when M is as small as 30.

4.5 GRID-S

It is worth noting that, when the requests on the grid are quite sparse and the lengths of the request arcs are small, GRID-L usually has unsatisfying approximation rate. GRID-S, like *SMALLARCS* [2], is designed to deal with this situation. However, in reality *SMALLARCS* often yields very unsatisfying results even in situations that it supposedly can handle well (which can be seen in Section 4.8).

On the grid, unlike that of a continuous Euclidean space, the size of vertices set is limited. In this case, pickup and drop-off points of the requests often overlap

on the same vertex. In graph theory, a connected component (or component) of an undirected graph is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the supergraph. For our directed graph (V, A), we define "connected component" as below:

Definition 4.1 *Connected Component (for directed graph): Let A be a arc set. All the arcs in $A_S \subseteqq A$ form a "connected component" if their corresponding undirected graph is also a connected component (Figure 4.5).*

The connected component is a partition of A. Let $CP = \{C_1, C_2, ..., C_c\}$ be the set of connected components in A, then $C_1 \cup C_2 \cup ... \cup C_c = A$ and $C_i \cap C_j = \varnothing \; \forall i, j = 1, ..., c$. In GRID-S, we will be dealing with connected components instead of individual requests. Observe that arcs in the same connected component will always end up in the same sub-tour if degree-balancing procedure is ever applied to the problem instance, so using connected components provides a certain level of aggregation.

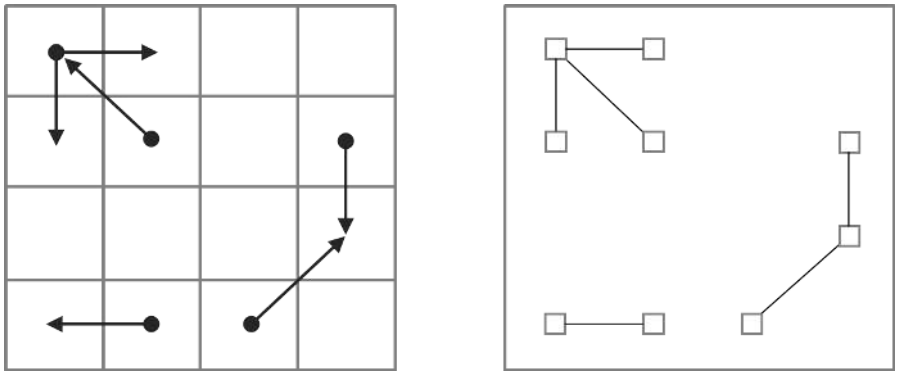

Figure 4.5: An instance with 6 requests and 3 connected components

In GRID-S, a preprocess is called to transform the problem instance into an Asymmetric Traveling Salesman Problem instance. To do this, we first identify all the connected components (CC) in A. Inside each CC, there would be vertices with positive, zero or negative net in-degree. Then, treating every CC as a node, we define the inter-node distance for node i and j to be the minimum energy cost from a vertex with positive net in-degree in i, to one with negative net in-degree in j. We present the full GRID-S in Table 4.4.

Algorithm: GRID-S

Step 1. Identify all the connected components in A. Let $CP = \{C_1, C_2, ..., C_c\}$ be the set of connected components in A;

Step 2. Define a directed network (N_C, A_C), where:

$N_C = \{1, 2, ..., c\}$ is a node set, in which each node represents a CC in CP,

$A_C = \{(i, j) : i, j \in N_C\}$, $d_{ij} = \min\{e_{mn} : m \in C_i, n \in C_j\}$

$m \in C_i$ is a vertex with a positive net in-degree

$n \in C_j$ is one with a negative net in-degree

e_{mn} is the energy cost associated with arc (m, n);

Step 3. Solve this transformed ATSP defined by (N_C, A_C). Let the solution be A_T, combine it with (V, A) to get a connected graph (V, A, A_T);

Step 4. Use minimum matching on (V, A, A_T) to get a strongly connected and degree-balanced graph (V, A, A_T, A_M). Find an Euler tour by means of any Euler tour construction algorithm;

Step 5. Post-process to eliminate unnecessary detours.

Table 4.4: Algorithm: GRID-S

GRID-S involves solving a transformed ATSP where each node in the ATSP is a connected component. This approach can be justified by its strengths:

- It automatically solves the problem to optimality when the number of requests is small.
- The number of connected component K is bounded by the layout of the grid. In fact, assuming a uniform distribution of the pickup and delivery vertices, K will reduces drastically when the number of requests is sufficiently large, which makes solving the ATSP very efficient even with large number of requests M (See Figure 4.6).
- Combined with GRID-L it provides a better theoretical worst-case bound (See Section 4.8).

4.6 Complexity of GRID-S

The computation time for GRID-S is not exponential in the number of requests M. To be specific, the GRID-S algorithm consists of 4 steps, the time complexity of each step is shown in Table 4.5:

Procedure	**Complexity**
Identify Connected Components (CCs)	$O(M)$
Solving the ATSP Defined by CCs	$T_C(K(M))$
Degree Balancing	$O(N^3)$
Identify an Euler Tour	$O(M \cdot N)$

Table 4.5: The time complexity of GRID-S

In the table above, we have denoted the computation time of solving the ATSP defined by CCs as T_C. T_C is a function of M. Intuitively, the larger the M, the larger the T_C. However, T_C is not a monotonic increasing function of M.

The solving time of the transformed ATSP instance in GRID-S is determined by the number of connected components, K. By the definition of connected components, $N/2$ is a natural upper bound for K; Thus we know for sure that K is bounded by the size of the underlying graph, so is T_C. Furthermore, in practice, assuming that pickup and delivery vertices are independently randomly distributed over the grid, the expected number of connected component can be calculated by means of simulation. In Figure 4.6, how the number of connected component K grows with the number of requests at different level of N is shown.

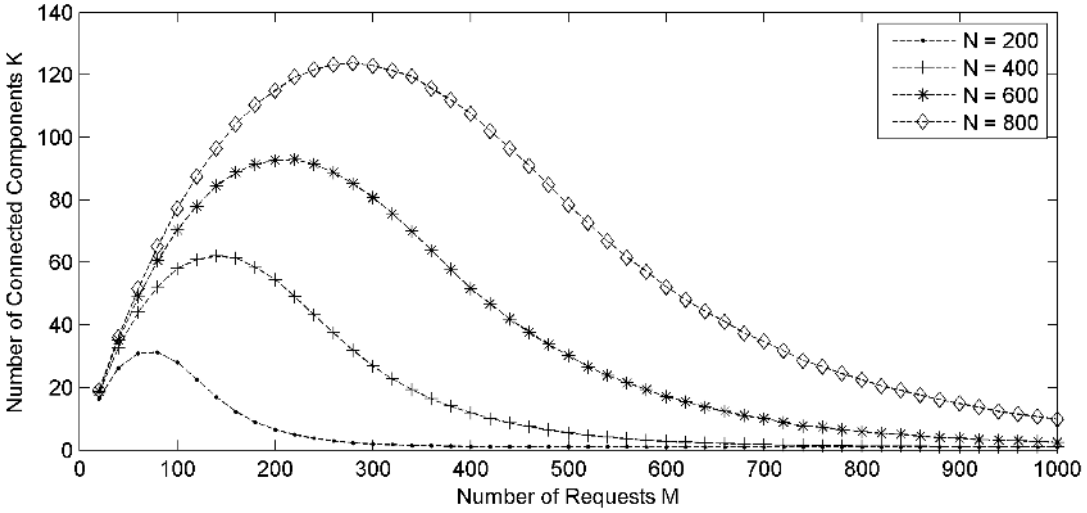

Figure 4.6: How K grows with the number of requests at different level of N

We can see that on a fixed-size grid, the expected number of K is not only bounded, but also decreases when number of requests M passes certain threshold. The peak number of K and the threshold is only determined by the size of the grid N. Thus we can treat this part of computation time as a bounded constant (as long as N and M are not simultaneously too large). In this sense, we say that GRID-S is "polynomial time solvable with respect to M". And this is how the time-efficiency of GRID-S is retained.

4.7 Better Theoretical Bound

Theorem 4.5 *Let C^* be the optimal tour cost for the EESCP, and C_L, C_S the cost generated by algorithm GRID-L and GRID-S respectively. $C_G = \min\{C_L, C_S\}$, then:*

$$C_G/C^* \leq 5/3$$

Proof:

The cost of GRID-L is like that of *LARGEARCS*, which provides an upper bound of $C_L \leq 3C^* - 2C_A$ [2].

The cost of GRID-S is at most $C^* + C_A$. The cost of the first three steps of GRID-S is at most $C^* - C_A$, this is due to that the transformed ATSP is essentially performed on a subset of arcs in A. Now consider step 4. Observe CP before step 3, we see that in every CC, the total number of in-degree equals the total number of out-degree. The resulting graph after step 3 is $G' = (V, A, A_T)$. Since A_T is the solution to an ATSP, observe that the effect of adding A_T to G is that it takes exactly one in-degree and one out-degree from every CC in CP. Let $CP' = \{C'_1, C'_2, ..., C'_c\}$ be the set of effected CCs, and $M(\cdot)$ be the result of matching operation. The matching process on G' is effectively applied on CP'. We know that the number of in-degree still equals out-degree in each $C'_i, \forall i = 1, ..., c$, thus $\{M(C'_1), M(C'_2), ..., M(C'_c)\}$ is a feasible solution to the matching over CP'. Let $c_M(\cdot)$ be the cost of the matching, then

$$c_M(CP') \leq c_M(C'_1) + c_M(C'_2)... + c_M(C'_c) \leq c_M(C_1) + c_M(C_2)... + c_M(C_c) \leq C_A$$

Thus the total cost of GRID-S is $C_S \leq C_A + C^* - C_A + C_A = C^* + C_A$.

Now if $C_A \geq \frac{2}{3}C^*$, we use GRID-L and:

$$C_G/C^* \leq (3C^* - 2C_A)/C^* \leq \frac{3C^* - 2(\frac{2}{3}C^*)}{C^*} = \frac{5}{3}$$

If $C_A \leq \frac{2}{3}C^*$, we apply GRID-S, then:

$$C_G/C^* \leq (C^* + C_A)/C^* \leq \frac{C^* + (\frac{2}{3}C^*)}{C^*} = \frac{5}{3}$$

□

4.8 Numerical Results

In this section, we present some simulation results for our proposed algorithms and models. Our simulations are run on a solid grid graph with 400 vertices unless pointed out otherwise, and assume uniform and independent distributions of pickup and drop-off vertices by default.

Exact Models

First, we see the computation time for solving the two exact formulations for EESCP (Figure 4.7) and Table 4.6: ATSP and Nodes Formulation. Due to memory shortage, the simulation is run on a grid structure with 100 vertices in total. The observation is that, Solving Nodes Formulation does tend to be more storage-demanding, but its required computation storage and the computation time are bounded by the grid size. On the other hand, the computation time and memory needed for solving the ATSP formulation, will go up exponentially with the number of requests.

Optimality Gap

We showcase the average optimality gaps of GRID-L under two different formulations in Figure 4.8, and the maximum optimality gaps in Table 4.7 (The minimum gaps are not shown because they are all zeros). We see that when the number of requests is small, these two algorithms can have optimality gaps as large as 20%. However, due to the postprocess and uniformly distributed arc lengths,

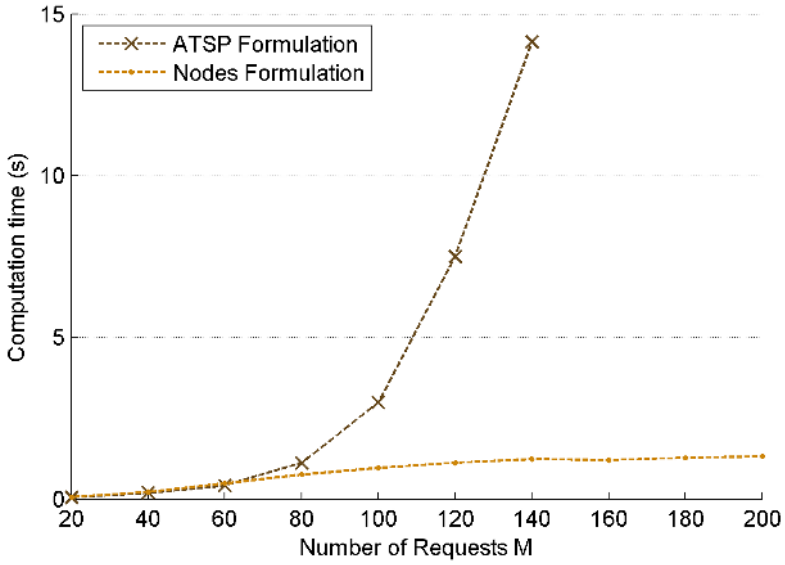

Figure 4.7: The computation time of solving ATSP Formulation and Nodes Formulation

	Computation Time (s)					
	ATSP			NODES		
M	Max.	**Ave.**	Min.	Max.	**Ave.**	Min.
20	0.16	**0.05**	0.02	0.20	**0.06**	0.03
40	0.35	**0.19**	0.07	0.31	**0.21**	0.13
60	1.04	**0.41**	0.19	0.76	**0.48**	0.30
80	3.21	**1.11**	0.45	1.18	**0.75**	0.53
100	8.43	**2.99**	1.02	1.38	**0.96**	0.71
120	16.31	**7.50**	1.83	1.72	**1.12**	0.84
140	26.64	**14.13**	2.97	1.52	**1.23**	0.92
160	-	-	-	1.39	**1.20**	0.99
180	-	-	-	1.46	**1.28**	1.12
200	-	-	-	1.59	**1.31**	1.11
220	-	-	-	1.82	**1.35**	1.19

Table 4.6: The computation time of solving ATSP and Nodes formulation (s)

these two algorithms have very small optimality gaps overall. In both cases the cost gap between GRID-L and exact solution diminishes as the number of requests M grows. And the average optimality gap can be as small as 2% when the number of requests is 30. The lower optimality gap under GRID Formulation than Transportation Formulation can be explained by the fact that the solution to GRID Formulation provides more and better options for the later linking process.

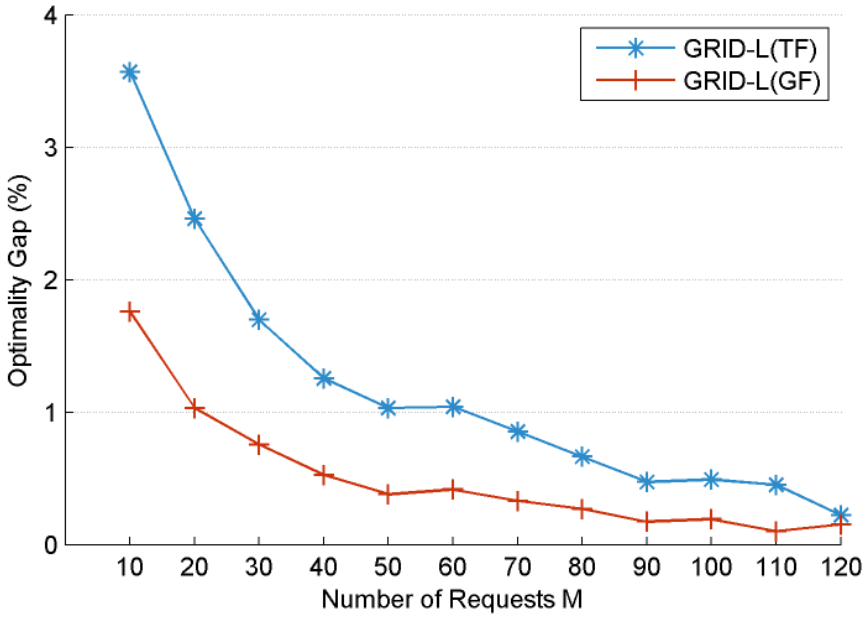

Figure 4.8: The optimality gap of GRID-L narrows when the number of requests goes up

	Maximum Optimality Gap (%)											
M	10	20	30	40	50	60	70	80	90	100	110	120
TF	21.8	14.0	10.1	8.5	6.3	4.8	4.7	3.3	2.6	2.8	3.1	1.8
GF	17.0	10.7	5.3	5.9	3.1	2.8	3.4	2.3	2.3	1.9	1.1	1.7

Table 4.7: Maximum optimality gap at different instance size for GRID-L(GF) and (TF), over 100 instances (%)

In Figure 4.9 we compare the optimality gaps of GRID-L, GRID-S and *SMALLARCS*. In this simulation we generated 50 instances, whose arcs in A are set small in length, putting GRID-S and *SMALLARCS* into their supposed advantage against GRID-L. From the result we see that GRID-S indeed performs better

than GRID-L when the number of requests is not large. Later as the optimality gap of GRID-L diminishes, it tends to merge with that of GRID-S. Also we see that the *SMALLARCS* could not outperform either GRID-L or GRID-S, often resulting with optimality gap as large as above 50% to 80%. See Table 4.8 for more detailed numerical results.

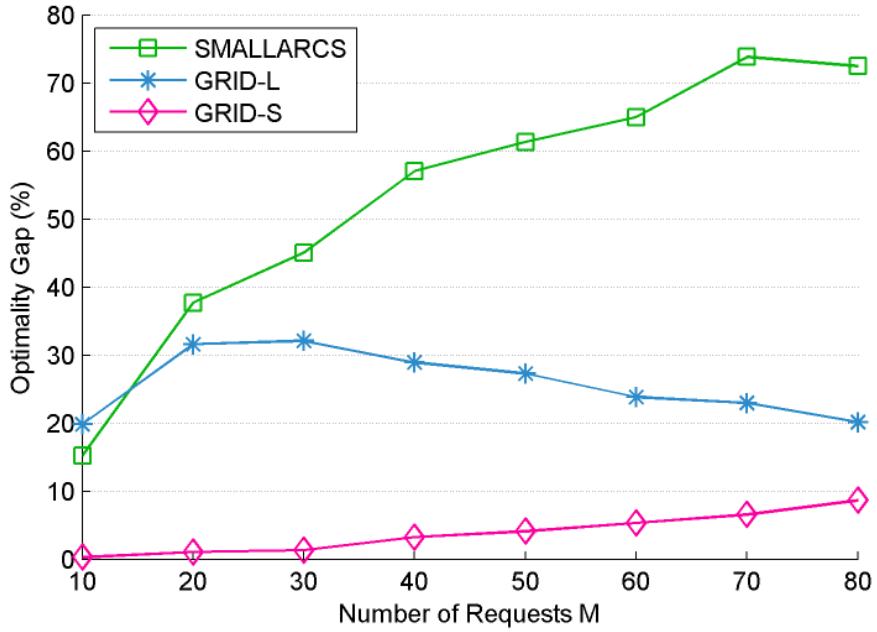

Figure 4.9: In a small-length arcs setting, GRID-S out performs both GRID-L and *SMALLARCS*

	Optimality Gaps (%)								
	SMALLARCS			GRID-L			GRID-S		
M	Max.	**Ave.**	Min.	Max.	**Ave.**	Min.	Max.	**Ave.**	Min.
10	47.06	**15.21**	0	53.13	**19.82**	0	8.33	**0.28**	0
20	79.01	**37.70**	4.26	56.76	**31.59**	8.08	5.33	**1.09**	0
30	86.87	**45.07**	9.26	45.36	**32.10**	13.08	10.10	**1.36**	0
40	90.76	**57.05**	23.42	47.54	**28.93**	16.95	9.52	**3.26**	0
50	86.61	**61.33**	39.39	44.09	**27.30**	7.94	13.22	**4.10**	0
60	102.9	**64.99**	34.59	37.96	**23.86**	11.11	23.26	**5.33**	0
70	115.9	**73.84**	36.60	46.51	**22.99**	8.70	23.08	**6.56**	0
80	101.3	**72.76**	40.00	36.00	**20.30**	8.43	34.44	**8.73**	1.18

Table 4.8: The optimal gaps of *SMALLARCS*, GRID-L, and GRID-S, over 50 instances (%)

Computational Time

The computation time of GRID-L under two formulations is shown in Figure 4.10. The average computation time of GRID-L is linear as predicted. And due to this linearity, the algorithm can easily compute up to instances with M as large as thousands in seconds. The variances of the computation time of these two algorithms are actually very small, see Table 4.9 for more numerical details.

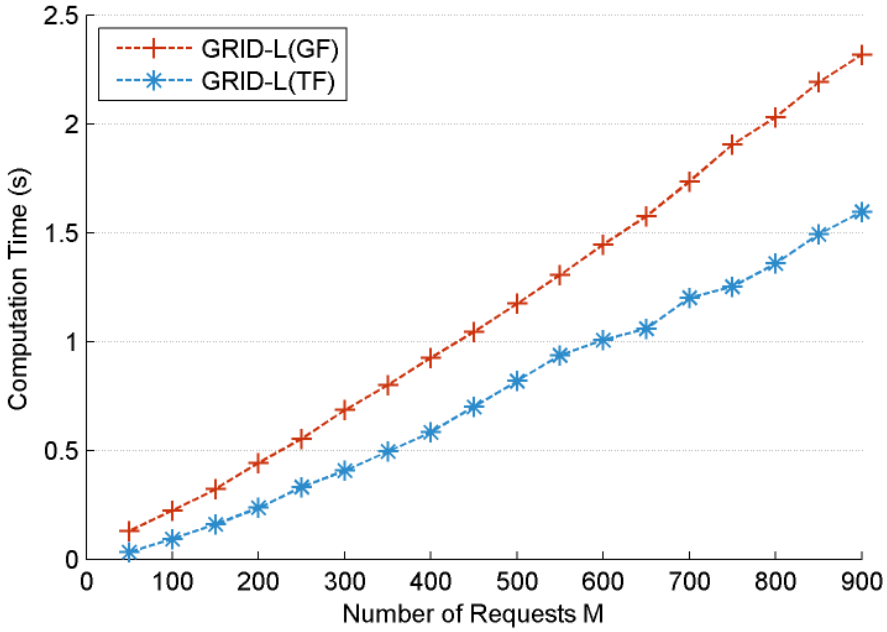

Figure 4.10: The computation time of GRID-L grows linearly, can be used instances with large M

In Figure 4.11 and Table 4.9 we compare the computation time of GRID-L and *SPLICE*. In GRID-L, we replaced the Bipartite Matching in *SPLICE* with solving a network-flow model. We do observe the advantage of doing this in our problem. The computation time of GRID-L grows linearly, while on the other hand, that of *SPLICE* still has a complexity of $O(n^{2+\varepsilon})$.

Finally we observe the computation time GRID-S in Figure 4.12. It is noted that since GRID-S involves solving an ATSP to optimal, the computation time depends a lot on the instances structure (large arcs or small arcs). And the computation time for an instance with mixed arc lengths is a lot less than that of one

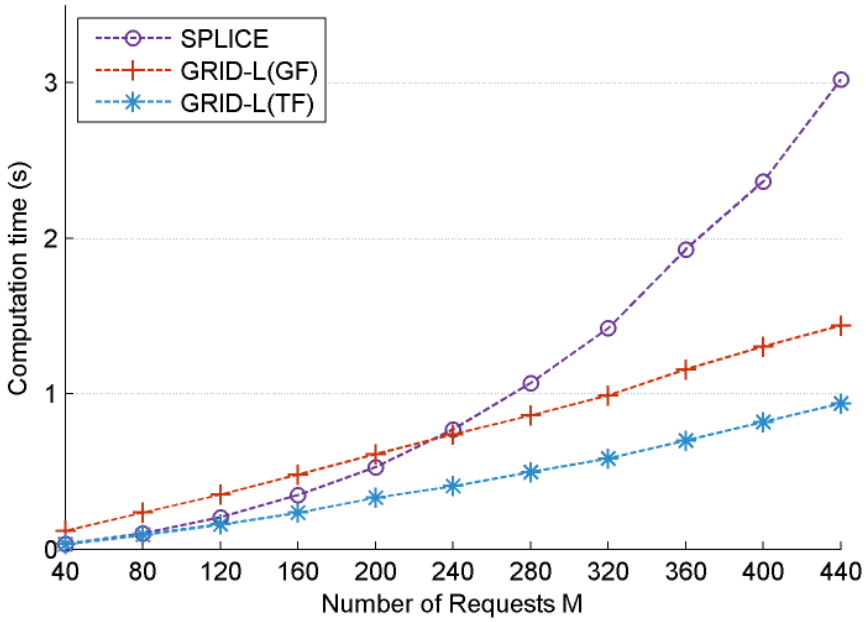

Figure 4.11: The computation time of GRID-L and SPLICE

	Computation Time (s)								
	GRID-L(GF)			GRID-L(TF)			SPLICE		
M	Max.	**Ave.**	Min.	Max.	**Ave.**	Min.	Max.	**Ave.**	Min.
40	0.21	**0.12**	0.08	0.08	**0.03**	0.02	0.06	**0.03**	0.02
120	0.53	**0.35**	0.22	0.32	**0.16**	0.13	0.34	**0.21**	0.18
200	0.91	**0.61**	0.45	0.56	**0.33**	0.27	0.66	**0.53**	0.46
280	1.20	**0.86**	0.65	0.65	**0.50**	0.43	1.41	**1.07**	0.91
360	1.68	**1.16**	0.92	0.95	**0.70**	0.59	2.93	**1.93**	1.51
440	2.29	**1.44**	1.14	1.61	**0.94**	0.81	4.47	**3.02**	2.36

Table 4.9: The computation time of GRID-L and SPLICE(s)

with only small arc lengths. To show its capability in worst case, we only focus on the computation time of GRID-S on small arc instances. And we see that the average computation time of GRID-S is within half a minute, bounded and display similar growth trend with that of the expected number of connected components K (Figure 4.6), which is quite expected. On the other hand, due to the ATSP solving procedure, the variance in computation time of GRID-S is high. See a box graph for the computation time of GRID-S in Figure 4.13.

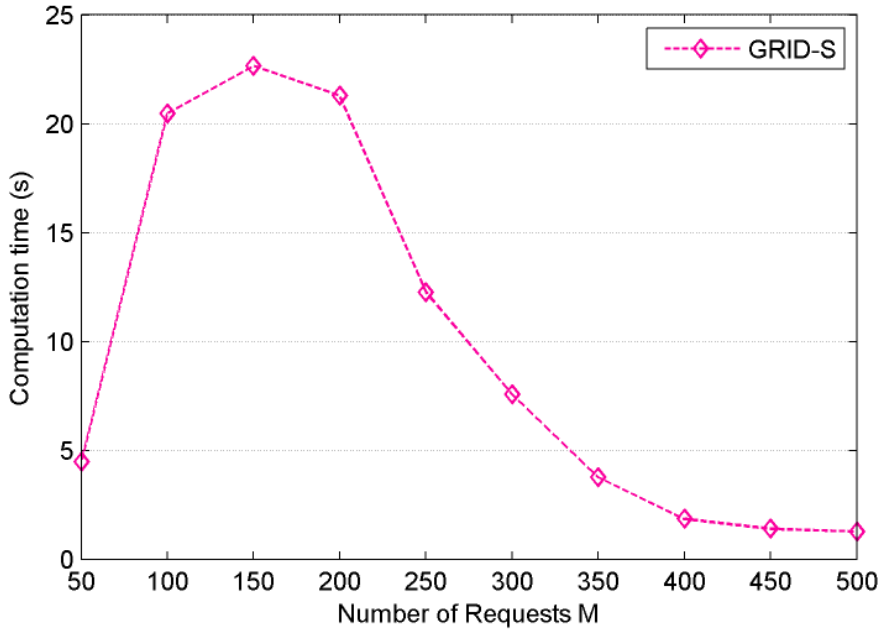

Figure 4.12: The computation time of GRID-S

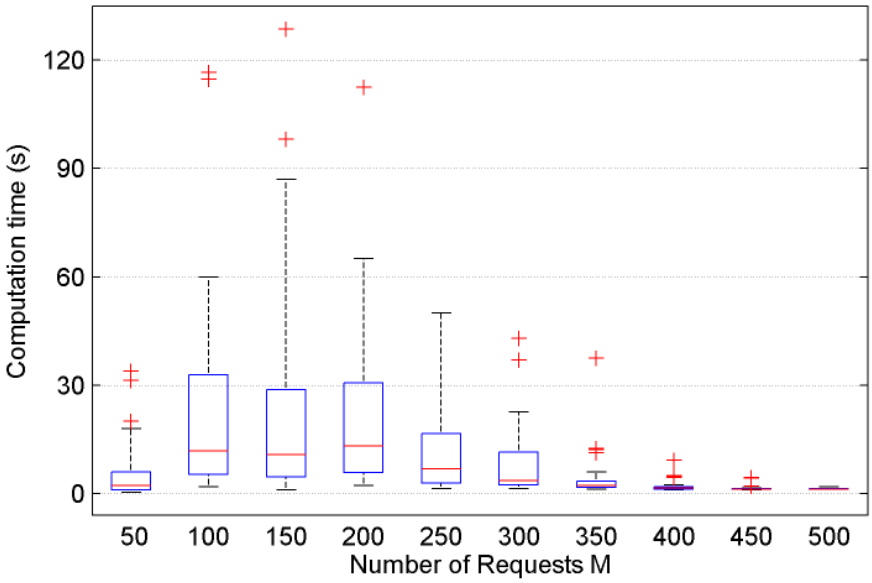

Figure 4.13: The computation time of GRID-S has a high variance

Chapter 5

Dynamic and Multi-Capacity Problem

5.1 Dynamic Problems

All previous chapters discuss static problems, where all information about pickup and delivery requests is given beforehand. In this chapter, we apply GRID algorithm to a dynamic environment, where the requests information is revealed over time. To adapt the algorithm to the dynamic environment, we develop a dynamic algorithm (Dynamic GRID) that applies the static algorithm repeatedly on the set of known requests that keeps updating. This dynamic algorithm is simple: (1) It first gathers(also wait for) a reasonable amount of requests information and start scheduling the sequence s with the static algorithm GRID; (2) After carrying out a certain number of requests in s, as new requests come, the algorithm initiates a re-optimization to update its scheduling. Dynamic GRID has two parameters: (1) Look-ahead horizon (h): how much request information the algorithm tries to gather before making a decision. This is measured by the number of requests the algorithm takes into account as input; (2) Decision point (dp): after an initial decision making, the number of requests the vehicle actually carries out before a re-optimization is initiated by the algorithm. We use simulations to investigate

the effect of these two parameters on the performance of the dynamic algorithm in this section.

5.1.1 Look-ahead Horizon

Intuitively, the larger the look-ahead horizon, the better the overall performance of the dynamic algorithm.

In the first simulation, 60 problem instances are randomly generated. In each instance, a total number of 300 pickup and delivery requests are uniformly distributed over a grid with 400 cells. The look-ahead horizon ranges from 10 to 100, which means in each iteration the algorithm tries to optimize a subset of requests of the size of 10 to 100 (To the algorithm, the remaining requests are still unknown). The decision-point is fixed at 10, i.e. a re-optimization is called after the vehicle finishes 10 requests in the solution to its last optimization. We assume that, before a new round of optimization, the vehicle will stop at the drop-off position of the last request of each iteration, instead of its initial position. Since the simulation is only for analysis purposes, we can assume that the arrival rate of the requests is high enough, so that in each iteration the number of requests waiting in queue is always larger or equal to the look-ahead horizon. To see purely the effect of look-ahead horizon on the dynamic algorithm performance, for each instance, the cost output by the Dynamic GRID at each h is normalized by the cost at $h = 100$ (supposedly the best result). The relative performance when $h = h_i$ is calculated as

$$relative\ \ performance = \frac{cost\ \ at\ \ h_i}{cost\ \ at\ \ h = 100} - 1$$

The average result over 60 instances are shown in Figure 5.1 and Table 5.1, we see that not surprisingly, the shorter the look-ahead horizon, the worse the cost of the dynamic solution. The gap is as large as 230% when the dynamic algorithm only tries to optimize 10 requests each iteration, then gradually reduces as the look-ahead horizon becomes larger.

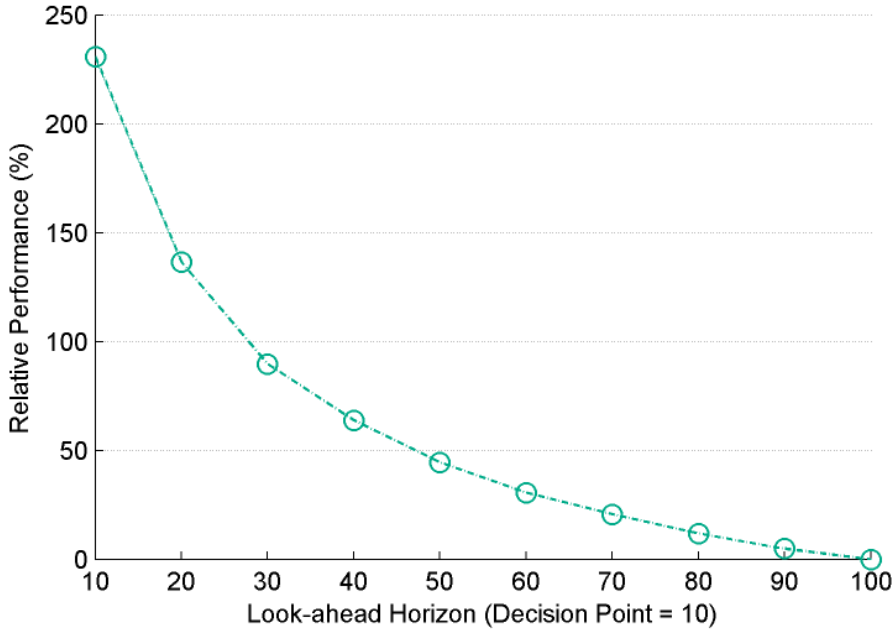

Figure 5.1: The performance of Dynamic GRID improves as larger look-ahead horizon is adopted

	Relative Performance (%)									
h	10	20	30	40	50	60	70	80	90	100
Max.	297.2	180.6	126.6	99.4	68.8	47.4	44.4	25.4	18.4	0
Ave.	**230.9**	**136.4**	**89.7**	**63.9**	**44.5**	**30.7**	**20.7**	**11.9**	**4.8**	**0**
Min.	186.2	103.9	62.0	40.7	21.7	10.7	0	-1.6	-5.8	0

Table 5.1: The performance of Dynamic GRID with different look-ahead horizon (%)

5.1.2 Decision Point

On the other hand, given the look-ahead horizon, reducing decision point (more frequent re-optimization) also seems to have an positive effect on the algorithm performance. In another 60 randomly generated instances, we fix the look-ahead horizon at 50, then change the decision point from 5 to 45 (1/10 to 9/10 of h). We do not examine when the decision point equals the look-ahead horizon for when $dp = h$, there would be no re-optimization at all. Then the average relative performance at each decision point is calculated over the 60 instances. The result is in Figure 5.2 and Table 5.2, we see that when the look-ahead horizon is fixed at 50, a more frequent re-optimization can resulting a 25% overall better gap than nearly no re-optimization. The relative performance when $dp = dp_i$ is calculated as

$$relative\ \ performance = \frac{cost\ \ at\ \ dp_i}{cost\ \ at\ \ dp = 5} - 1$$

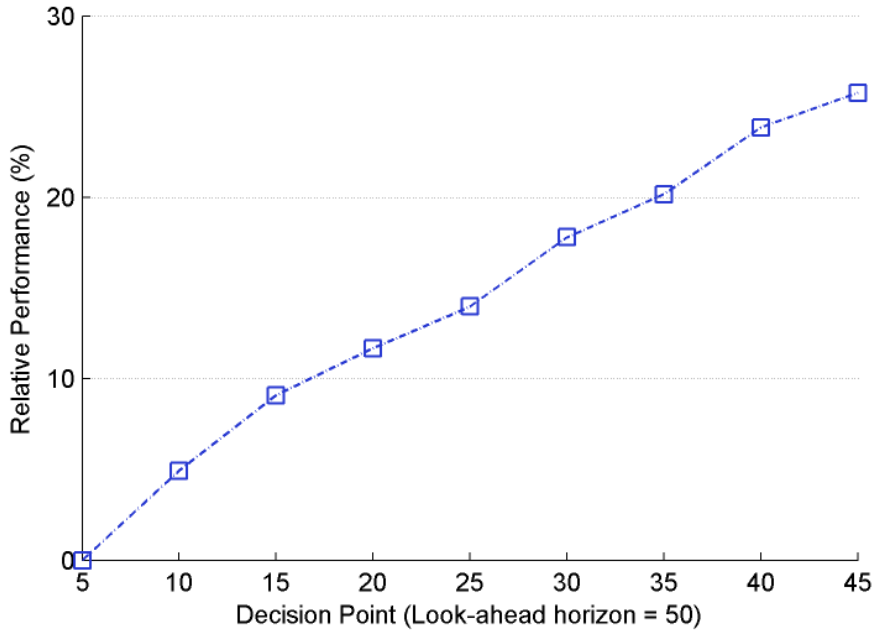

Figure 5.2: On average, smaller decision points provide smaller gaps

Which is the more influential factor among these two? To answer this question, we must have a better view over the impact of these two parameters combined.

	Relative Performance (%)								
dp	5	10	15	20	25	30	35	40	45
Max.	0	16.1	16.2	26.4	23.5	30.1	34.5	37.4	42.7
Ave.	**0**	**4.9**	**9.1**	**11.7**	**14.0**	**17.8**	**20.2**	**23.9**	**25.7**
Min.	0	-5.4	-2.3	1.6	3.5	5.9	6.7	12.0	11.5

Table 5.2: The performance of Dynamic GRID with different values of decision point (%)

The look-ahead horizon h and decision point dp are in fact inter-related: decision point is obviously bounded by the look-ahead horizon. In this simulation, we randomly generate 60 instances with 300 uniformly distributed requests. Each instance is input into the dynamic algorithm under 10 different settings of look-ahead horizon, ranging from 10 to 100. At each value of h, 9 different settings, 1/10 to 9/10 of h, of decision point are chosen to run each instance. For example, for any instance, we run simulation on $dp = \{2, 4, 6, 8, 10, 12, 14, 16, 18\}$ when h is fixed at 20; and on $dp = \{5, 10, 15, 20, 25, 30, 35, 40, 45\}$ when h is fixed at 50. The average algorithm performance is observed over the 60 random instances. The relative performance at h_i, dp_j is calculated as

$$relative\ \ performance = \frac{cost\ \ at\ \ h_i,\ \ dp_j}{cost\ \ at\ \ h = 100,\ \ dp = 10} - 1$$

Figure 5.3 indicates that, indeed, both h and dp have a clear effect on the performance of the dynamic algorithm. Larger h implies better relative performance. On the other hand, within each h, a smaller dp has a positive impact on the performance on average– $dp = \frac{h}{10}$ generally has a huge gap between that of $dp = \frac{9h}{10}$. We further observe that, when the look-ahead horizon h is small, increasing the h by 10 has a larger impact on the performance: the performance of the smallest decision point dp at $h = 10$ is still much worse than that of the largest dp at $h = 20$. This almost still holds when h is increased from 20 to 30. However, this fact does not hold anymore if we increase h from 30 to 40: the smallest dp at $h = 30$ outperforms a medium decision point (around 25) at $h = 40$. Then the impact of increasing h by 10 continues to dwindle when h becomes larger. From

Figure 5.3, we seem to have the rule of thumb that: when $h \geq 30$, the performance of smallest $dp = \frac{h}{10}$ is almost as good as that of $dp = \frac{h+10}{4}$.

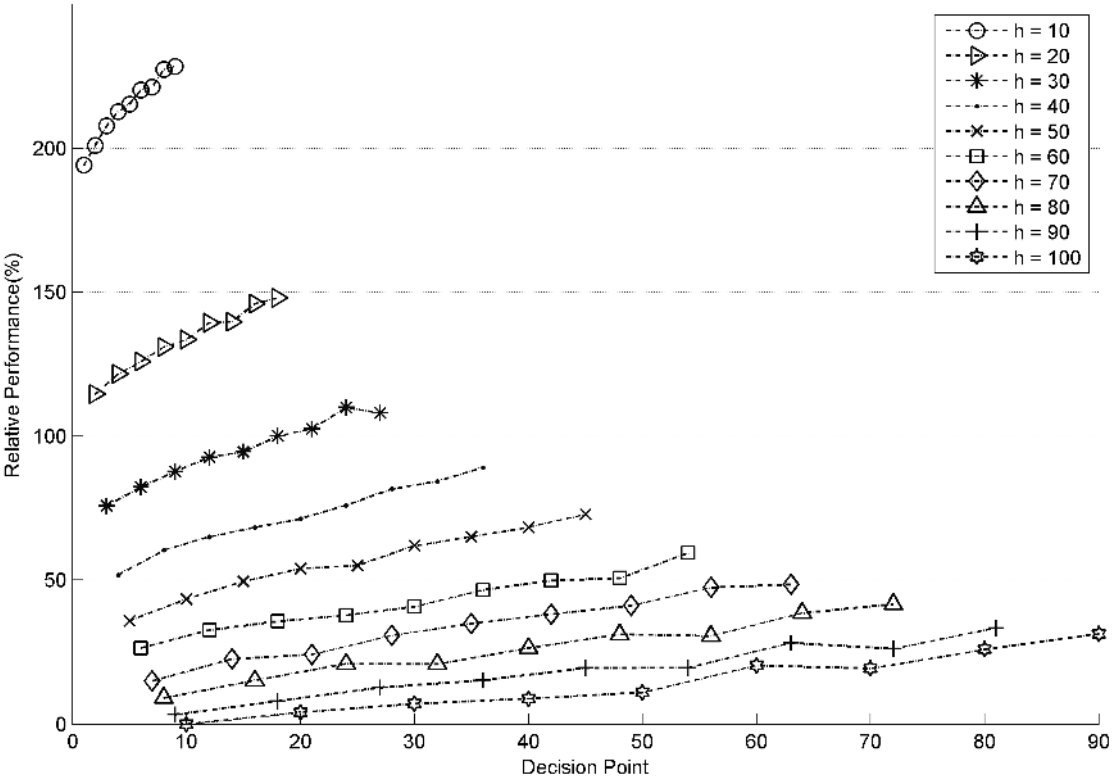

Figure 5.3: The combined effect of look-ahead horizon and decision point

To summarize, the implication of the simulation result is simple: the dynamic algorithm should try to optimize as many as requests as allowed by the time constraints, and as frequent as possible when new requests appear. In practice, the look-ahead horizon is generally determined by how urgent the time constraints are. The more controllable parameter of these two is apparently the frequency of re-optimization. However, also note that $dp = \frac{h}{10}$ means 9 times the computation effort over that of $dp = \frac{9h}{10}$.

Finally, Dynamic GRID is compared against another commonly used heuristic in dynamic problems– Nearest-Neighbour. In Nearest-Neighbour, once a delivery is made, it will immediately go to the next pickup location that is the nearest to its current location. In this simulation, 30 instances with number of requests ranging from 100 to 300 are sovled by these two algorithms. For the dynamic

GRID algorithm, the look-ahead horizon and decision point are fixed at 50 and 1 respectively. The Nearest-Neighbour runs in a similar fashion: the vehicle decides its next request by picking the nearest request to itself among the available 50 requests. In this simulation, the relative performance is defined as

$$relative\ \ performance = \frac{cost\ \ by\ \ Nearest\ Neighbour}{cost\ \ by\ \ Dynamic\ \ GRID} - 1$$

As demonstrated by Figure 5.4, when the number of requests is relatively small, the performance of Nearest-Neighbour is nearly 25% worse than that of Dynamic GRID. The performance gap narrows as the number of requests gets larger, to as small as 10% when $M = 300$. The relative performance between these two algorithms has a high variance, see the box graph in Figure 5.5: though being worse overall, the Nearest-Neighbour can outperform Dynamic GRID, to the extent of $(-)17\%$ when $M = 250$.

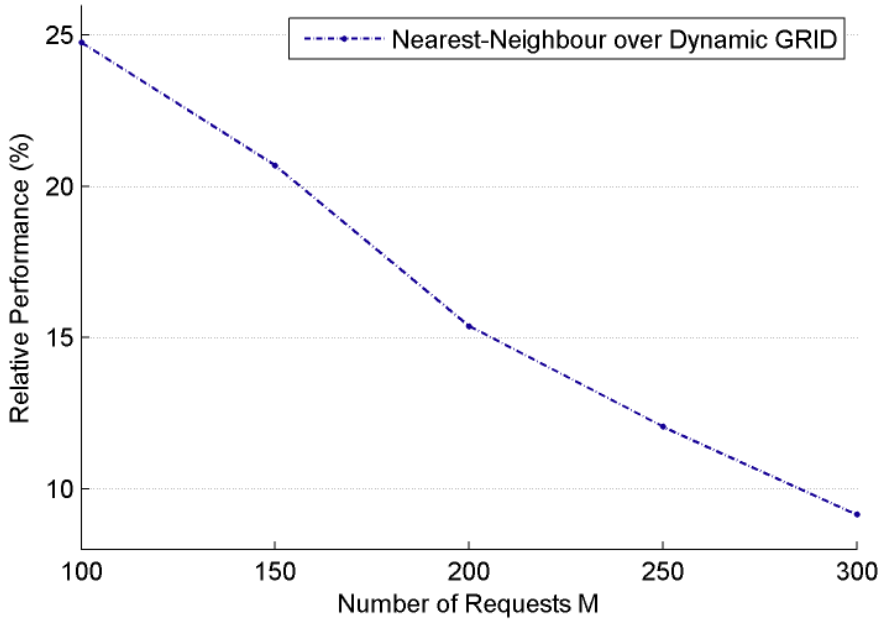

Figure 5.4: Relative performance of Nearest-Neighbour against Dynamic GRID

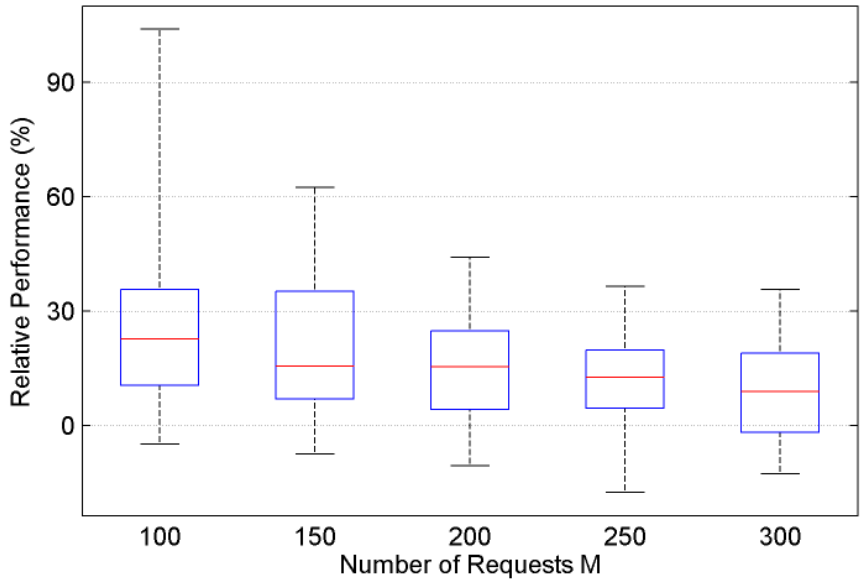

Figure 5.5: The relative performance between Nearest-Neighbour and Dynamic GRID has a high variance

5.2 Multi-Capacity Problems

Multi-capacity problem is referred to as the problem in which the capacity of the vehicle is greater than 1. Compared with unit-capacity problem, it is much harder to solve for four reasons: First, to construct a feasible sequence, each request will have to be broken down to two action nodes, pickup and drop-off, hence twice the sequence size than that of the unit-capacity problem. Secondly, for each request, the pickup and drop-off nodes are related in such a way that the pickup must come before drop-off (preceding constraint). The third reason is that, the problem now has a capacity constraint– the number of items on the vehicle cannot exceed its capacity at all time. Finally, in multi-capacity problems, the energy cost incurred by any movement of the vehicle is now a function of its current load.

In this section, we first introduce an exact formulation for multi-capacity problem. Since the formulation is bilinear, we discuss the solution approach to transform it into linear form. Then we propose three heuristic algorithms for multi-

capacity EESCPs. Their pseudo codes are presented for when capacity equals two. And we show these heuristics can be extended to cases with arbitrary (integer) capacity.

5.2.1 Multi-Capacity Exact Formulation

In multi-capacity problems, we must translate an input of M requests into an input of $2M$ action nodes. Let $P = \{P_1, P_2, P_3...P_M\}$ be the set of all the pickup action nodes, $D = \{D_1, D_2, D_3...D_M\}$ the set of all the drop-off nodes. Then we combine P and D and index all the action nodes from 1 to $2M$. Let this set be $AN = \{a_1, a_2, ..., a_M, a_{M+1}, ..., a_{2M}\}$, a_1, a_2 to a_M represent action nodes from P, and a_{M+1}, a_{M+2} to a_{2M} represent action nodes from D. To simplify the problem, we assume that the load in each request is of the same mass m_l, the vehicle itself has a mass of m_0. Now we define the decision variables:

Decision Variables:

$X = \{x_{ij} \in \{0, 1\}\}$, $x_{ij} = 1$ if action a_i is followed by action a_j, $\forall i, j = 1, 2, ..., 2M, i \neq j + M$;

$U = \{u_k \in [1, 2M]\}$, u_k is a continuous variable; $u_k = s$ if the action a_k is the s^{th} in the solution sequence;

$W = \{w_l \in \{0, 1, ..., CAP\}\}$, w_l is the load of the vehicle after the l^{th} action in the solution sequence;

Parameters:

CAP is a positive integer, the capacity of the vehicle;

$c_{ij}(w_i)$ is the cost incurred by the vehicle going from action a_i directly to a_j, with load w_i. It is a function of w_i;

m_0 is the mass of the empty vehicle. m_l is the mass of the load in each request;

h_{ij}, v_{ij} are the horizontal and vertical displacements between action node i and j respectively

$q_j \in \{-1, 1\}$ is the effect on the load of the vehicle of action a_j. $q_j = 1$ if $j = 1, ..., M$; $q_j = -1$ if $j = M + 1, ..., 2M$;

Objective Function:

$$\min . \sum_{i=1,...,2M} \sum_{j=1,...,2M} c_{ij}(w_i) x_{ij}$$

Constraints:

$$\sum_{i=1,...,2M} x_{ij} = 1 \qquad \forall j = 1..., 2M$$

$$\sum_{j=1,...,2M} x_{ij} = 1 \qquad \forall i = 1..., 2M \quad (5.1)$$

$$if \quad x_{ij} = 1 \Rightarrow u_i \leq u_j - 1 \qquad \forall i, j = 1..., 2M (5.2)$$

$$j \neq 1$$

$$u_i \leq u_{i+M} - 1 \qquad \forall i = 1..., M \quad (5.3)$$

$$1 \leq u_k \leq 2M \qquad \forall k = 1..., 2M$$

$$u_1 = 1 \quad (5.4)$$

$$if \quad x_{ij} = 1 \Rightarrow w_j = w_i + q_j \qquad \forall i, j = 1..., 2M (5.5)$$

$$0 \leq w_l \leq CAP \qquad \forall l = 1..., 2M \quad (5.6)$$

$$c_{ij} = (m_0 + w_i \cdot m_l)(k_H \cdot g\mu \cdot h_{ij} + k_V \cdot g \cdot v_{ij}) \qquad \forall i, j = 1..., 2M (5.7)$$

This formulation is bilinear programming, obtained by adding the load constraints to a sequential formulation for ATSP problems [89]. The objective function is to find the minimum-cost tour to go through all the action nodes. Constraint (5.2) is about the order of the sequence, indicating if action a_j follows action a_i, then the order of a_j should be at least that of a_i plus one. Constraint (5.3) enforces that the pickup action must happen before the drop-off action of the same request. And constraint (5.4) fixes the pickup action of the first request as its first action. In fact, this choice can be made arbitrarily to any pickup action according to the problem setting. Constraint (5.5) and (5.6) are the load constraints– the

load of the vehicle must not exceed the capacity at all time. Constraint (5.7) is the functional relationship between c_{ij} and w_i. Though this formulation appears simple, it is in fact difficult to be solved.

5.2.2 Solution Approach

Let $k_H \cdot g\mu \cdot h_{ij} + k_V \cdot g \cdot v_{ij} = \alpha_{ij}$. Note that the objective function can be rewritten as:

$$\sum_{i,j=1,...,2M} c_{ij}(w_i)x_{ij} = \sum_{i,j=1,...,2M} (m_0 + w_i \cdot m_l) \cdot \alpha_{ij}x_{ij}$$

which contains bilinear terms of $w_i x_{ij}$. This makes it harder to solve this problem. Fortunately, the objective function can be linearized by introducing new decision variables and additional linear constraints.

First, we replace the term $c_{ij}(w_i)x_{ij}$ with $z_{ij} \geq 0$, then the objective function is equivalent to:

$$\sum_{i,j=1,...,2M} c_{ij}(w_i)x_{ij} = \sum_{i,j=1,...,2M} z_{ij}$$

subject to,

$$x_{ij} = 1 \Rightarrow z_{ij} = (m_0 + m_l \cdot w_i)\alpha_{ij} \quad \forall i, j = 1, 2, ..., 2M$$

Further note that it is unnecessary to introduce z_{ij} for every x_{ij}, since by constraint (5.1), there exists one and only one immediate successor j to i such that $x_{ij} = 1$. Thus the above can be finally written as:

$$min. \sum_{i,j=1,...,2M} z_i$$

s.t.

$$z_i \geq (m_0 + m_l \cdot w_i)\alpha_{ij} - \beta(1 - x_{ij}) \quad \forall i, j = 1, 2, ..., 2M$$

β is a large positive real number, it can be chosen as the upper bound of $(m_0 + m_l \cdot w_i)\alpha_{ij}$. Now the formulation is reformulated in linear form, we can use solver like CPLEX to solve this exact formulation, to serve as benchmark for our proposed three heuristics later.

5.2.3 *2AS1*

This heuristic comes from the insight that, for a 2-capacity vehicle, once it has 1 item onboard, it becomes a single-capacity vehicle. Initially the vehicle is in state *Load=0*, it then decides on one item to pickup with single-capacity algorithm GRID, and enter state *Load=1*. In state *Load=1*, it proceeds to decide to pick up another item to enter state *Load=2* or to drop off the item on board to enter state *Load=0*. In state *Load=2*, the vehicle decides on which of the two items onboard to be dropped first, and enter state *Load=1* again. The algorithm runs until all the requests are fulfilled. Table 5.3 is the pseudo code for this algorithm, this algorithm can be extended to when capacity equals $C \geq 3$ by introducing another $C - 2$ intermediate state like STATE:1.

5.2.4 *PPDD*

This heuristic takes the output of a single-capacity algorithm, and adjust the sequence of the single-capacity solution into a 2-capacity fashion. Given a single-capacity solution sequence, say, $[3, 1, 4, 2]$, since the unit-capacity vehicle cannot perform another pickup action before dropping off the item onboard, the real operation sequence for the vehicle is actually $AS = [P_3, D_3, P_1, D_1, P_4, D_4, P_2, D_2]$. The heuristic *PPDD* is simple: adjust AS to a new sequence in the form of two pick-ups followed by two drop-off actions without violating any preceding constraints ($[PPDDPPDD]$), hence the name. The pseudo code for this algorithm is given in Table 5.4.

Algorithm: 2AS1

Input: R– all pickup and delivery locations; R'=R

Initialize: the vehicle in STATE:0 $ONBOARD$:empty;

All requests in R as STATUS:P (waiting to be picked up);

While R' is not empty

If STATE:0

Input R' to algorithm GRID, in the output of GRID, choose the pickup point a_p of the first request a as the next point to visit;

Register the delivery point a_d of a as onboard, $ONBOARD+ = a_d$; Mark request a as STATUS:D (waiting to be dropped off);

Modify R' by changing (a_p, a_d) in R' to (a_d, a_d). As a_d is yet to be visited;

Enter STATE:1;

If STATE:1

Input R' to GRID, in the output of GRID, check the first request b:

If STATUS(b)=P

Register the delivery point b_d of b as onboard, $ONBOARD+ = b_d$; Mark request b in STATUS:D (waiting to be dropped off)

Modify R' by changing (b_p, b_d) in R' to (b_d, b_d);

Enter STATE:2;

If STATUS(b)=D

Delete the delivery point b_d from onboard, $ONBOARD- = b_d$;

Modify R' by deleting (b_d, b_d) in R';

Enter STATE:0;

If STATE:2

Among the two items onboard, choose the nearest one to drop off according to the vehicle's current location;

Modify R' accordingly;

Enter STATE:1;

End While

Table 5.3: Algorithm: 2AS1

Algorithm: PPDD

Input: R– all pickup and delivery locations;

Initialize: The vehicle's initial location;

Input R to algorithm GRID, from the output sequence, generate the unit-capacity action sequence S_1;

The capacity is 2, calculate the number of $[PPDD]$ segments n_s in S_1;

For segment from 1 to n_s:

Rearrange each segment $[P_x, D_x, P_y, D_y]$ into:

$[P_x, P_y, D_x, D_y]/[P_x, P_y, D_y, D_x]/[P_y, P_x, D_y, D_x]/[P_y, P_x, D_x, D_y]$

Choose the one arrangement with the lowest cost given the vehicle's current location;

Add the rearranged segment to the final sequence; Update the vehicle's location;

End For

Table 5.4: Algorithm: *PPDD*

5.2.5 *BUBBLE*

This heuristic starts with any solution sequence, and tries to improve it, by repeatedly swapping each pickup/ delivery action in the original solution with its neighboring actions to achieve less overall cost. Note that the swapping can sometimes lead to infeasible solutions for violating preceding constraints as well as capacity constraints: A pickup action cannot be sequenced after its corresponding delivery action, likewise, a delivery action cannot be moved before its pickup action; The number of items on the vehicle cannot exceed its capacity at all time. The heuristic is designed in such a way that each swapping is only valid if it passes the feasibility check.

Note that since this is a heuristic based on any input sequence, better outcome can be achieved by a change in the initial input of S_1. In our implementation of *BUBBLE*, we used the output of a Multi-capacity Nearest-Neighbour heuristic as the initial input sequence (Table 5.5). By changing the capacity constraint feasibility check function, this algorithm can be easily extend to problem with any

capacity size. This heuristic takes the name from the well-known sorting algorithm, and its pseudo code is shown in Table 5.6.

Algorithm: Multi-capacity Nearest-Neighbour

Input: R– all pickup and delivery locations;

Initialize: The vehicle's initial location; LOAD=0;

While there is still pickup or drop-off nodes left:

    If LOAD:0

        Among all the remaining pickup nodes, choose the nearest one as its next pickup action;

        Enter LOAD=1;

    If LOAD:1

        Among all the remaining pickup nodes, and the drop-off node onboard, choose the nearest one as its next action;

        If a pickup node is chosen:

        Enter STATE:2;

        If the drop-off node is chosen:

        Enter STATE:0;

    If LOAD:2

        Among the two drop-off nodes of the items onboard, choose the nearest one as its next drop-off action;

        Enter LOAD=1;

End While

Table 5.5: Algorithm: Multi-capacity Nearest-Neighbour

5.2.6 Simulation Results

First we compare our three heuristics against the 2-capacity exact formulation to see their optimality gaps. Being NP-complete, the 2-capacity problems are in fact very hard to solve to optimal in a reasonable amount of time. So the largest instances in this simulation only have 10 requests. The average optimality gaps of the three heuristics are calculated over 30 randomly generated instances whose request number ranges from 4 to 10.

Algorithm: BUBBLE

Input: R– all pickup and delivery locations;

Initialize: The vehicle's initial location; CHANGE=TRUE;

Input R to Nearest-Neighbour Heuristic, from the output request sequence, generate the initial action sequence S_1; $S' = S_1$

While there is change to S':

For each node $s \in S'$(iterate from left to right):

If s is a pickup node: $p = s$;

Locate the location of corresponding delivery node of p, $d(p)$, in S';

In the range of $[1, d(p)]$, put p into each new position, check if the capacity constraint is violated anywhere in the new formed sequence; Mark these new locations as "feasible" or "infeasible" accordingly;

Relocate p to the "feasible" location with the minimum cost; Modify S' accordingly;

If s is a drop-off node: $d = s$;

Locate the location of corresponding delivery node of d, $p(d)$, in S';

In the range of $[p(d), end]$, put d into each new position, check if the capacity constraint is violated anywhere in the new formed sequence; Mark these new locations as "feasible" or "infeasible" accordingly;

Relocate d to the "feasible" location with the minimum cost; Modify S' accordingly;

End While

Table 5.6: Algorithm: *BUBBLE*

Shown in Figure 5.6, *BUBBLE* is the closest to the optimal among the three, *PPDD* has a mediocre performance and *2AS1* has generally a very large gap from optimality (more than 200%, thus not a very satisfactory algorithm). Within the range of 10 requests, the optimality gaps of *PPDD* and *BUBBLE* are between 20-60%. Considering the *BUBBLE* algorithm has a larger search effort than *PPDD*, its better performance is expected.

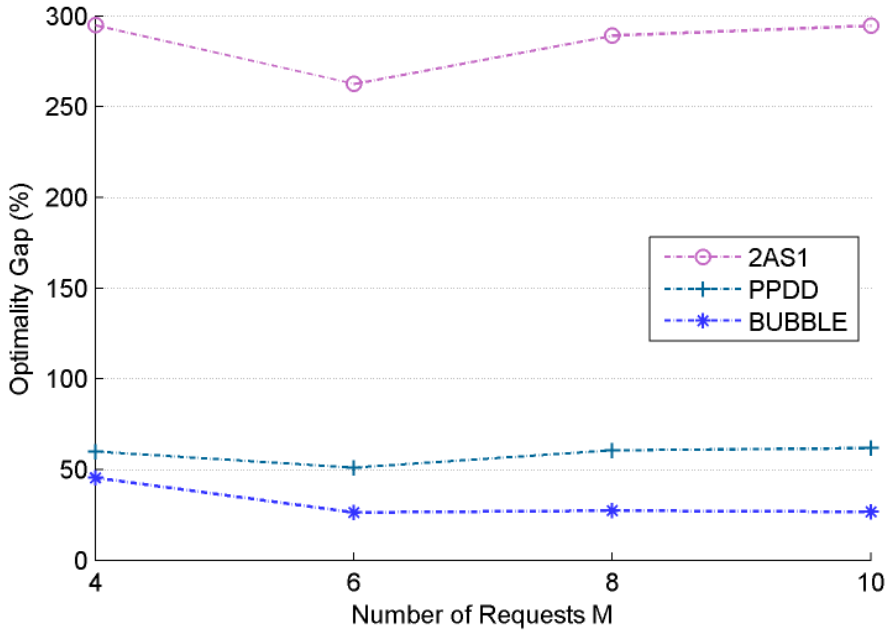

Figure 5.6: The optimality gaps of the three heuristics

We further use the heuristics to solve larger instances. Since *BUBBLE* seems to be the best of the three, we use the cost output of *BUBBLE* as benchmark to examine the other two. Figure 5.7 and Table 5.7shows that *BUBBLE* remains the best of the three in terms of objective value. *2AS1* is still more than 150% worse than *BUBBLE*, and the performance gap between *PPDD* and *BUBBLE* grows larger as the size of instances increases.

Finally we examine the computation time of *PPDD* and *BUBBLE* in Figure 5.8. The computation time of *PPDD* is minimal for its simplicity; *BUBBLE*, on the other hand, takes longer to converge to local optimums.

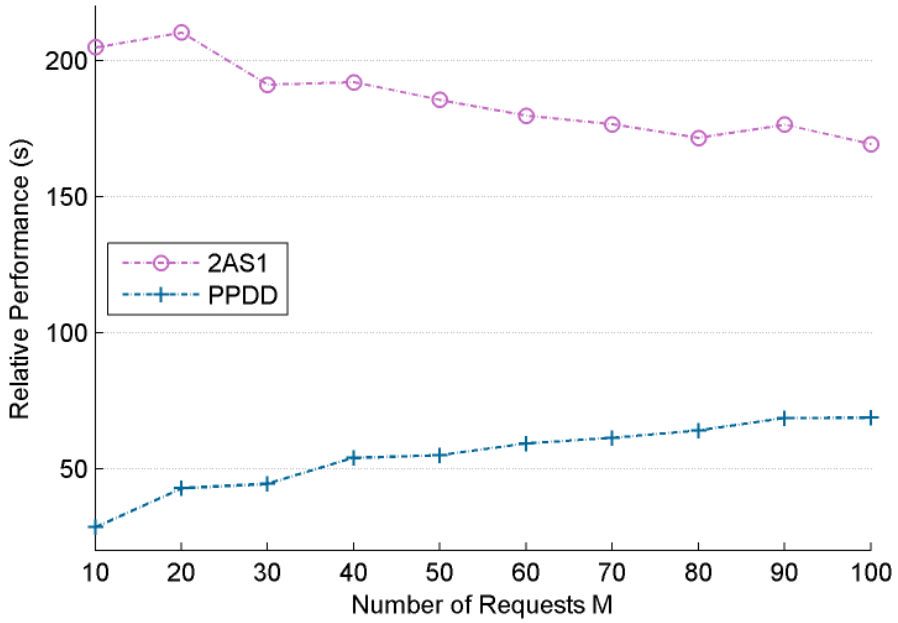

Figure 5.7: The relative performances of the three heuristics

	Relative Performance against BUBBLE (%)					
	PPDD			2AS1		
M	Max.	**Ave.**	Min.	Max.	**Ave.**	Min.
10	75.0	**28.4**	3.1	380.0	**204.8**	136.8
20	66.4	**42.8**	17.5	287.3	**210.2**	135.7
30	64.6	**44.4**	20.0	247.6	**191.0**	140.0
40	74.4	**53.9**	33.8	262.3	**192.0**	142.2
50	74.8	**55.0**	37.3	245.8	**185.5**	141.0
60	78.1	**59.3**	39.9	257.0	**179.7**	140.7
70	78.7	**61.3**	48.7	219.6	**176.5**	140.6
80	79.2	**64.0**	49.2	201.9	**171.5**	139.4
90	84.5	**68.6**	52.3	223.0	**176.4**	133.4
100	85.5	**68.7**	53.1	204.5	**169.2**	136.4

Table 5.7: The relative performances of the *2AS1* and *PPDD* against*BUBBLE* (%)

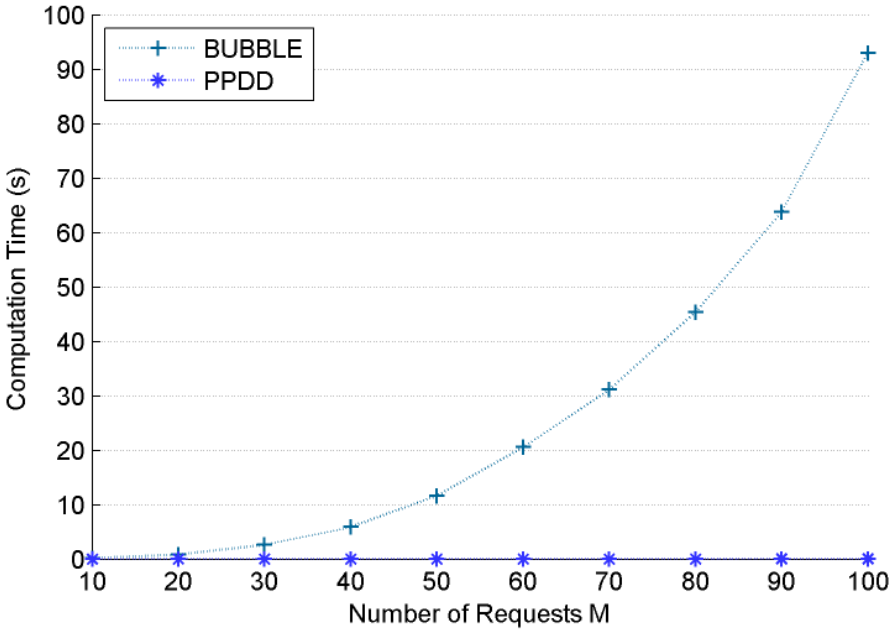

Figure 5.8: The computation time of PPDD and BUBBLE

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Modern Automated Material Handling Systems are an essential part to modern businesses, productions and logistics. Many optimization problems that arise in AMHSs still impose meaningful and valuable research topics. In this thesis, starting from a practical problem, we defined our studied problem: Energy-Efficient Stacker Crane Problem (EESCP) as a stacker crane routing problem on a grid network with cost function in Manhattan norm. The structural properties of this specific problem bring about many insightful discoveries.

First, we showed that EESCP, being the SCP on grid network in Manhattan norm, is still NP-Complete. This proof is based on from C.H. Papadimitriou's work [88].

Secondly, due to the Manhattan norm, some 2D problems can be easily solved to optimal in polynomial time with 1D problem algorithm, if their arc patterns satisfy a set of conditions. The arc pattern is such that, when we divide a 2D instance into its two 1D sub-problems (horizontally and vertically), at least one of the sub-problem satisfies the "free-permutation" property, enabling the other sub-problem to decide the optimal sequence for the original 2D problem alone. Under certain assumptions, we have thoroughly identified all the possible arc patterns for the "free-permutation" property. This can be very useful when applied to some

problems that arise in practical operations similar to Ball and Magazine's circuit printing problem and Atallah and Kosaraju's robotic arm routing problem.

Then we looked into general EESCP instances without the "free-permutation" property. With the introduction of the underlying grid network in our formulation, we are able to improve a very classic algorithm *CRANE* proposed by Frederickson, Hecht et al. [2]. The improvements are four-fold: GRID-L is asymptotically optimal, and converges to optimal very fast in practical problems; GRID-L also has a linear time complexity with respect to the number of requests in the problem instances; GRID-S is designed in such a way that when the arc lengths are small, it can indeed outperform GRID-L, unlike *SMALLARCS* in *CRANE*, whose performance is in fact worse than *LARGEARCS* even if the arc lengths are small; GRID-L and GRID-S together provide a 5/3 theoretical bound, improving from 9/5.

We tentatively try to adopt our static algorithm GRID to dynamic environments by adding two dynamic parameters: Look-ahead Horizon and Decision Point. The effects on the performance when changing these two parameters are examined by simulations. The simulation result shows that the look-ahead horizon has a significant impact on the dynamic algorithm performance: the larger the look-ahead horizon, the smaller optimality gap. A smaller decision point generally means a higher frequency of re-optimization. The simulations have also shown that frequent re-optimizations statistically improve the performance of the dynamic algorithm.

Finally, we investigate the multi-capacity problem. The formulation for this problem is bilinear due to the problem being innately more difficult. We propose a technique to reformulate it into linear form. Then we develop three heuristics in total. Two of them are based on rearranging other feasible sequence into a new sequence of pickup and drop-off actions that better utilize the extra capacity, without violating any preceding constraints or load constraints. These two heuristics turn out to have quite small gaps from optimality, especially *BUBBLE*.

6.2 Future Work

In this section, we will discuss how the work reported in this thesis can be further extended.

6.2.1 Integrate Time Information

In our specific problem, we have assumed no time window for each request. Though there are practical cases where this assumption holds, for our work to extend to more general routing problems in various kinds of AMHSs, we have to address the energy efficiency issues as well as time issues such as service rate, total makespan, or longest waiting time. For example, in a problem where energy efficiency is the only concern, a request that comes first has a chance to get served last, making its waiting time unreasonably long. To take on time information such as "estimated time of arrival" or "latest time of delivery" is to add time windows into the studied problem and make the model to fit in more general scenarios, while making the problem harder to solve. Still, "slacking" as much as time constraints allow is the ultimate goal of an energy-efficient algorithm. To find the optimal balance between the two objectives of time and energy in a multi-objective optimization problem can be a valuable issue to further look into.

6.2.2 Collaboration Between Multiple Vehicles

As a means to increase system throughput, many AHMSs adopt multiple vehicles. To determine how to distribute the workload between all the available vehicles, as well as how to coordinate the vehicles to avoid potential collisions, adds a new level of complexity to this problem. As we have discussed in Chapter 2, there have been studies on multi-vehicle SCPs. Previous work was either in industries and problems where collision avoidance is not an issue, or the result was too system-design specific. So it is still meaningful to devise schemes or algorithms to help collaborate multiple vehicles to achieve better overall energy and time efficiency.

6.2.3 Further into the Multi-capacity Problem

Multi-capacity operations are essentially a trade-off between throughput and the energy consumption– picking up another item while having items onboard often means unnecessary detour for the items onboard and hence extra energy consumption for the system. In this sense, the philosophy of energy-efficient operations somehow discourages full-load operations, unless necessary. This is an interesting balance to further look into. Also, the solution method used in this thesis for multi-capacity is in a general form, more effort could be put into finding ways to integrate more system structure into the modeling.

6.2.4 Stochastic Framework

Most of our work in this thesis is about solving static problems, and assumes no advance knowledge about the future. However, the real-world problems are never truly static– they are always dynamic on a infinite time horizon. Forecasts such as the seasonal, daily or hourly expected arrival rate, experience on the frequently seen request patterns issued in the system, are valuable information to be integrated into the dynamic model. These information can shape the decision making and help increase the expected overall energy-efficiency. It would be useful and also viable to extend this work to use stochastic programming framework to tackle meaningful dynamic problems, given some information about the future.

The first application is the Docking Problem. For instance, after the stacker crane vehicle having completed all the given requests, knowing that the next request would come sometime later, the vehicle wants to choose a location to dock, to minimize the response time once the next request comes. Assuming that there are 3 very likely locations for the next pickup point to be, at the immediate stage, the vehicle needs to choose the expected location to minimize the expected traveling time to any one of these three locations. At stage two, once there is an update of the information of the incoming request, the vehicle then adjust its docking locations accordingly.

The second is to use the deterministic equivalent of stochastic incoming requests to incorporate uncertainties into the deterministic frame work. To do this, for each uncertain incoming request, given the distributions of its pickup node and delivery node, we can use the expectation (weighted mean) of the distributions to generate a deterministic equivalent request. See Figure 6.1 for a demonstration.

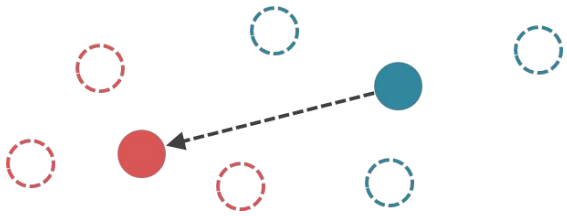

Figure 6.1: The deterministic equivalent of an uncertain request

However, this is only an estimate of future requests, once valid information about the future requests is obtained, the routing must be actively re-optimized.

Author's Publications

Papers under Preparation

(i) Fang Zhou, Jianfeng Mao,"Energy-Efficient Stacker Crane Routing on a Grid". *Under Preparation.*

References

[1] J. Huston. Save energy in your warehouse to meet green initiatives, 2013. Available at: http://www.supplychaindigital.com/warehousing/1905/Save-Energy-in-Your-Warehouse-to-Meet-Green-Initiatives.

[2] G.N. Frederickson, M.S. Hecht, and C.E. Kim. Approximation algorithms for some routing problems. In *Foundations of Computer Science, 1976., 17th Annual Symposium on*, pages 216–227, Oct 1976.

[3] K.J. Roodbergen and I.F.A. Vis. A survey of literature on automated storage and retrieval systems. *European Journal of Operational Research*, 194(2):343 – 362, 2009.

[4] W.A. Günthner, C. Tilke, and S. Rakitsch. Energy efficiency in bulk materials handling. *Bulk Solids Handling*, 30(3):138 – 142, 2010.

[5] German Federal Environment Agency (UBA). Stromsparen: weniger kosten, weniger kraftwerke, weniger co_2 fakten und argumente für das handeln auf der verbraucherseite, Dec 2009. Available at: http://www2.fml.mw.tum.de/fml/images/Publikationen/Gunthner_Tilke_Rakitsch_BulkSolidsHandling_03-10_EnergyEfficiency.pdf.

[6] Y.A. Bozer J.A. Tompkins, J.A. White and J.M.A. Tanchoco. *Facilities planning*. Wiley, 4th edition, 2010.

[7] R. Siddhartha. *Introduction to Materials Handling*. New Age International, 1st edition, Oct 2007.

[8] S. Hur, Y.H. Lee, S.Y. Lim, and M.H. Lee. A performance estimation model for as/rs by m/g/1 queuing system. *Computers & Industrial Engineering*, 46(2):233 – 241, 2004. Special Issue on Selected Papers from the 27th. International Conference on Computers and Industrial Engineering, Part 1.

[9] R. Manzini, M. Gamberi, and A. Regattieri. Design and control of an as/rs. *The International Journal of Advanced Manufacturing Technology*, 28(7-8):766–774, 2006.

[10] R.J. Linn and R.A. Wysk. An expert system framework for automated storage and retrieval system control. *Computers & Industrial Engineering*, 18(1):37 – 48, 1990.

[11] T. Balluff. Energy Efficiency-Intralogistics focused on energy efficiency, 2009. Available at: `http://www.daifukueurope.com/news/press/146/2009-28/Energy-Efficiency`.

[12] J. Huston. Save energy in your warehouse to meet green initiatives, 2013.

[13] C. Prasse, A. Kamagaew, S. Gruber, K. Kalischewski, S. Soter, and M. Ten Hompel. Survey on energy efficiency measurements in heterogenous facility logistics systems. In *Industrial Engineering and Engineering Management (IEEM), 2011 IEEE International Conference on*, pages 1140–1144. IEEE, 2011.

[14] P.A. Makris, A.P. Makri, and C.G. Provatidis. Energy-saving methodology for material handling applications. *Applied energy*, 83(10):1116–1124, 2006.

[15] G. Frederickson and D. Guan. Preemptive ensemble motion planning on a tree. *SIAM Journal on Computing*, 21(6):1130–1152, 1992.

[16] M. Atallah and S. Kosaraju. Efficient solutions to some transportation problems with applications to minimizing robot arm travel. *SIAM Journal on Computing*, 17(5):849–869, 1988.

[17] M.O. Ball and M.J. Magazine. Sequencing of insertions in printed circuit board assembly. *Operations Research*, 36(2):192–201, 1988.

[18] G.N. Frederickson and D.J. Guan. Nonpreemptive ensemble motion planning on a tree. *Journal of Algorithms*, 15(1):29 – 60, 1993.

[19] A. Coja-Oghlan, S.O. Krumke, and T. Nierhoff. A heuristic for the stacker crane problem on trees which is almost surely exact. *Journal of Algorithms*, 61(1):1 – 19, 2006.

[20] K. Treleaven, M. Pavone, and E. Frazzoli. Asymptotically optimal algorithms for one-to-one pickup and delivery problems with applications to transportation systems. *IEEE Transactions on Automatic Control*, 58(9):2261–2276, Sept 2013.

[21] K. Treleaven, M. Pavone, and E. Frazzoli. Models and efficient algorithms for pickup and delivery problems on roadmaps. In *2012 IEEE 51st Annual Conference on Decision and Control (CDC)*, pages 5691–5698, Dec 2012.

[22] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. *Management Sciencies Research Report No.388*, 1976.

[23] T. Ávila, Á. Corberán, I. Plana, and J.M. Sanchis. The stacker crane problem and the directed general routing problem. *Networks*, 65(1):43–55, 2015.

[24] G. Laporte. Modeling and solving several classes of arc routing problems as traveling salesman problems. *Computers & Operations Research*, 24(11):1057 – 1061, 1997.

[25] J. Cirasella, D. Johnson, L. McGeoch, and W. Zhang. The asymmetric traveling salesman problem: Algorithms, instance generators, and tests. In *Algorithm Engineering and Experimentation*, volume 2153 of *Lecture Notes in Computer Science*, pages 32–59. Springer Berlin Heidelberg, 2001.

[26] L. Zheng and W. Zheng. Genetic coding for solving both the stacker crane problem and its k-variant. In *IEEE International Conference on Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century.*, volume 2, pages 1061–1066 vol.2, Oct 1995.

[27] F.J. Srour and S. Velde. Are stacker crane problems easy? A statistical study. *Computers & Operations Research*, 40(3):674 – 690, 2013.

[28] D.J. Guan. Routing a vehicle of capacity greater than one. *Discrete Applied Mathematics*, 81(13):41 – 57, 1998.

[29] M. Ozden. A simulation study of multiple-load-carrying automated guided vehicles in a flexible manufacturing system. *International Journal of Production Research*, 26(8):1353–1366, 1988.

[30] D.M. Stein. An asymptotic, probabilistic analysis of a routing problem. *Mathematics of Operations Research*, 3(2):89–101, 1978.

[31] H.N. Psaraftis. Analysis of an $o(N^2)$ heuristic for the single vehicle many-to-many euclidean dial-a-ride problem. *Transportation Research Part B: Methodological*, 17(2):133 – 145, 1983.

[32] H.N. Psaraftis. An exact algorithm for the single vehicle many-to-many dial-a-ride problem with time windows. *Transportation Science*, 17(3):351–357, 1983.

[33] M. Kubo and H. Kasugai. Heuristic algorithms for the single vehicle dial-a-ride problem. *Journal of the Operations Research Society of Japan*, 33(4):354–365, 1990.

[34] L.J.J. Bruggen, J.K. Lenstra, and P.C. Schuur. Variable-depth search for the single-vehicle pickup and delivery problem with time windows. *Transportation Science*, 27(3):298–311, 1993.

[35] J. Desrosiers, Y. Dumas, and F. Soumis. A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows. *American Journal of Mathematical and Management Sciences*, 6(3-4):301–325, 1986.

[36] K.S. Ruland and E.Y. Rodin. The pickup and delivery problem: Faces and branch-and-cut algorithm. *Computers & Mathematics with Applications*, 33(12):1 – 13, 1997.

[37] M. Grunow, H. Günther, and M. Lehmann. Dispatching multi-load agvs in highly automated seaport container terminals. *OR Spectrum*, 26(2):211–235, 2004.

[38] D. Sinriech and L. Palni. Scheduling pickup and deliveries in a multiple-load discrete carrier environment. *IIE Transactions*, 30(11):1035–1047, 1998.

[39] D. Sinriech and J. Kotlarski. A dynamic scheduling algorithm for a multiple-load multiple-carrier system. *International Journal of Production Research*, 40(5):1065–1080, 2002.

[40] Y. Dumas, J. Desrosiers, and F. Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54(1):7 – 22, 1991.

[41] H. Xu, Z. Chen, S. Rajagopal, and S. Arunapuram. Solving a practical pickup and delivery problem. *Transportation science*, 37(3):347–364, 2003.

[42] S. Røpke, J. Cordeau, and G. Laporte. Models and a branch-and-cut algorithm for pickup and delivery problems with time windows. *Networks*, 49(4):258–272, 2007.

[43] H.C. Lau and Z. Liang. Pickup and delivery with time windows: Algorithms and test case generation. *International Journal on Artificial Intelligence Tools*, 11(03):455–472, 2002.

[44] R. Bent and P. Van Hentenryck. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & Operations Research*, 33(4):875–893, 2006.

[45] B. Mahadevan and T.T. Narendran. Design of an automated guided vehicle-based material handling system for a flexible manufacturing system. *International Journal of Production Research*, 28(9):1611–1622, 1990.

[46] M.E. Johnson and M.L. Brandeau. An analytic model for design of a multi-vehicle automated guided vehicle system. *Management Science*, 39(12):1477–1489, 1993.

[47] T. Ganesharajah, N. Hall, and C. Sriskandarajah. Design and operational issues in agv-served manufacturing systems. *Annals of Operations Research*, 76(0):109–154, 1998.

[48] J. Lee. Composite dispatching rules for multiple-vehicle agv systems. *SIMULATION*, 66(2):121–130, 1996.

[49] T. Schouwenaars, B. De Moor, E. Feron, and J. How. Mixed integer programming for multi-vehicle path planning. In *European control conference*, volume 1, pages 2603–2608, 2001.

[50] W.B. Powell, B. Bouzaene-Ayari, and H.P. Simo. Chapter 5 dynamic models for freight transportation. In C. Barnhart and G. Laporte, editors, *Transportation*, volume 14 of *Handbooks in Operations Research and Management Science*, pages 285 – 365. Elsevier, 2007.

[51] W.B. Powell. An operational planning model for the dynamic vehicle allocation problem with uncertain demands. *Transportation Research Part B: Methodological*, 21(3):217–232, 1987.

[52] W.B. Powell, Y. Sheffi, K.S. Nickerson, K. Butterbaugh, and S. Atherton. Maximizing profits for north american van lines' truckload division: A new framework for pricing and operations. *Interfaces*, 18(1):21–41, 1988.

[53] M.R. Swihart and J.D. Papastavrou. A stochastic and dynamic model for the single-vehicle pick-up and delivery problem. *European Journal of Operational Research*, 114(3):447 – 464, 1999.

[54] J. Yang, P. Jaillet, and H. Mahmassani. On-line algorithms for truck fleet assignment and scheduling under real-time information. *Transportation Research Record: Journal of the Transportation Research Board*, 1667:107–113, 1999.

[55] J. Yang, P. Jaillet, and H. Mahmassani. Real-time multivehicle truckload pickup and delivery problems. *Transportation Science*, 38(2):135–148, 2004.

[56] D. Tjokroamidjojo, E. Kutanoglu, and G.D. Taylor. Quantifying the value of advance load information in truckload trucking. *Transportation Research Part E: Logistics and Transportation Review*, 42(4):340 – 357, 2006.

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

[58] M. Mes, M. Heijden, and A. Harten. Comparison of agent-based scheduling to look-ahead heuristics for real-time transportation problems. *European Journal of Operational Research*, 181(1):59 – 75, 2007.

[59] G. Berbeglia, J. Cordeau, I. Gribkovskaia, and G. Laporte. Static pickup and delivery problems: aclassification scheme and survey. *TOP*, 15(1):1–31, 2007.

[60] M.W.P. Savelsbergh and M. Sol. The general pickup and delivery problem. *Transportation Science*, 29(1):17–29, 1995.

[61] S. Parragh, K. Doerner, and R. Hartl. A survey on pickup and delivery problems. *Journal fr Betriebswirtschaft*, 58(1):21–51, 2008.

[62] B.L. Golden, S. Raghavan, and E.A. Wasil. *The Vehicle Routing Problem: Latest Advances and New Challenges: latest advances and new challenges*, volume 43. Springer Science & Business Media, 2008.

[63] J. Cordeau, G. Laporte, J. Potvin, and M. Savelsbergh. Transportation on demand. *Handbooks in operations research and management science*, 14:429–466, 2007.

[64] O.G. Madsen, H. Ravn, and J. Rygaard. A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. *Annals of Operations Research*, 60(1):193–208, 1995.

[65] L. Suen, A. Ebrahim, and M. Oksenhendler. Computerised dispatching for sharedride taxi operations in canada. *Transportation Planning and Technology*, 7(1):33–48, 1981.

[66] D.O. Santos and E.C. Xavier. Taxi and ride sharing: A dynamic dial-a-ride problem with money as an incentive. *Expert Systems with Applications*, 42(19):6728 – 6737, 2015.

[67] P. Toth and D. Vigo. Fast local search algorithms for the handicapped persons transportation problem. In IbrahimH. Osman and JamesP. Kelly, editors, *Meta-Heuristics*, pages 677–690. Springer US, 1996.

[68] B. Rekiek, A. Delchambre, and H.A. Saleh. Handicapped person transportation: An application of the grouping genetic algorithm. *Engineering Applications of Artificial Intelligence*, 19(5):511 – 520, 2006.

[69] R. Borndörfer, M. Grötschel, F. Klostermeier, and C. Küttner. Telebus berlin: Vehicle scheduling in a dial-a-ride system. In NigelH.M. Wilson, editor, *Computer-Aided Transit Scheduling*, volume 471 of *Lecture Notes in Economics and Mathematical Systems*, pages 391–422. Springer Berlin Heidelberg, 1999.

[70] J. Mageean and J.D. Nelson. The evaluation of demand responsive transport services in europe. *Journal of Transport Geography*, 11(4):255 – 270, 2003.

[71] J. Cordeau and G. Laporte. The dial-a-ride problem (darp): Variants, modeling issues and algorithms. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1(2):89–101, 2003.

[72] J. 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.

[73] J. Cordeau, M. Iori, G. Laporte, and J.J. Salazar Gonzlez. A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with lifo loading. *Networks*, 55(1):46–59, 2010.

[74] G.B. Dantzig and J.H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

[75] P. Toth and D. Vigo, editors. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[76] G. Laporte. Fifty years of vehicle routing. *Transportation Science*, 43(4):408–416, November 2009.

[77] R.M. Karp. A patching algorithm for the nonsymmetric traveling-salesman problem. *SIAM Journal on Computing*, 8(4):561–573, 1979.

[78] A. Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

[79] G. Gutin and A. Yeo. Assignment problem based algorithms are impractical for the generalized tsp. *Australasian Journal of Combinatorics*, 27:149–154, 2003.

[80] J. Marecek. *The traveling salesman problem: A computational study*. INFORMS 7240 PARKWAY DR, STE 310, HANOVER, MD 21076-1344 USA, 2008.

[81] R. Matai, M.L. Mittal, and S. Singh. *Traveling salesman problem: An overview of applications, formulations, and solution approaches*. INTECH Open Access Publisher, 2010.

[82] G. Gutin and A.P. Punnen. *The traveling salesman problem and its variations*, volume 12. Springer Science & Business Media, 2002.

[83] G. Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247, 1992.

[84] H.A. Eiselt, M. Gendreau, and G. Laporte. Arc routing problems, Part II: The rural postman problem. *Operations Research*, 43(3):399–414, 1995.

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

[86] P. Zajac. Evaluation method of energy consumption in logistic warehouse systems, 2015.

[87] C. Umans and W. Lenhart. Hamiltonian cycles in solid grid graphs. In *38th Annual Symposium on Foundations of Computer Science, Proceedings*, pages 496–505, Oct 1997.

[88] C.H. Papadimitriou. The euclidean travelling salesman problem is np-complete. *Theoretical Computer Science*, 4(3):237–244, 1977.

[89] A.J. Orman and H.P. Williams. A survey of different integer programming formulations of the travelling salesman problem. In *Optimisation, Econometric and Financial Analysis*, volume 9 of *Advances in Computational Management Science*, pages 91–104. Springer Berlin Heidelberg, 2007.

[90] A.W. Tucker H.W. Kuhn. An extension of a theorem of dantzig's. *Linear Inequalities and Related Systems, Annals of Mathematics Studies (AM-38)*, pages 247 – 254, 1956.

[91] T. Zaslavsky. Signed graphs. *Discrete Applied Mathematics*, 4(1):47 – 74, 1982.

[92] M. Hanan. On steiner's problem with rectilinear distance. *SIAM Journal on Applied Mathematics*, 14(2):255–265, 1966.

[93] M. Zachariasen. A catalog of hanan grid problems. *Networks*, 38(2):76–83, 2001.