THE ORDER-PICKING PROBLEM IN PARALLEL-AISLE WAREHOUSES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MELİH ÇELİK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
INDUSTRIAL ENGINEERING

JUNE 2009

Approval of the thesis:

THE ORDER-PICKING PROBLEM IN PARALLEL-AISLE WAREHOUSES

submitted by **MELİH ÇELİK** in partial fulfillment of the requirements for the degree of **Master of Science in Industrial Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Nur Evin Özdemirel _____
Head of Department, **Industrial Engineering**

Assoc. Prof. Dr. Haldun Süral _____
Supervisor, **Industrial Engineering Department, METU**

Examining Committee Members:

Prof. Dr. Nur Evin Özdemirel
Industrial Engineering, METU _____

Assoc. Prof. Dr. Haldun Süral
Industrial Engineering, METU _____

Prof. Dr. M. Selim Aktürk
Industrial Engineering, Bilkent University _____

Asst. Prof. Dr. Serhan Duran
Industrial Engineering, METU _____

Prof. Dr. Ömer Kırca
Industrial Engineering, METU _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: MELİH ÇELİK

Signature :

ABSTRACT

THE ORDER-PICKING PROBLEM IN PARALLEL-AISLE WAREHOUSES

Çelik, Melih
M.S., Department of Industrial Engineering
Supervisor : Assoc. Prof. Dr. Haldun Süral

June 2009, 183 pages

Order-picking operations constitute the costliest activities in a warehouse. The order-picking problem (OPP) aims to determine the route of the picker(s) in such a way that the total order-picking time, hence the order-picking costs are minimized. In this study, a warehouse that consists of parallel pick aisles is assumed, and various versions of the OPP are considered. Although the single-picker version of the problem has been well studied in the literature, the multiple-picker version has not received much attention in terms of algorithmic approaches. The literature also does not take into account the time taken by the number of turns during the picking route. In this thesis, a detailed discussion is made regarding the computational complexity of the OPP with a single picker. A heuristic procedure, which makes use of the exact algorithm for the OPP with no middle aisles, is proposed for the single-picker OPP with middle aisles, and computational results on randomly generated problems are given. Additionally, an evolutionary algorithm that makes use of the cluster-first, route-second and route-first, cluster-second heuristics for the VRP is provided. The parameters of the algorithm are determined based on preliminary runs and the algorithm is also tested on randomly generated problems, with different weights given to the cluster-first, route-second and route-first, cluster-second approaches. Lastly, a polynomial time algorithm is proposed for the problem

of minimizing the number of turns in a parallel-aisle warehouse.

Keywords: Order-Picking Problem, Heuristics, Evolutionary Algorithms, Turn Penalties, Computational Complexity

ÖZ

KORİDORLARI BİRBİRİNE PARALEL DİZİLMİŞ DEPOLARDA SİPARİŞ TOPLAMA PROBLEMİ

Çelik, Melih
Yüksek Lisans, Endüstri Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Haldun Süral

Haziran 2009, 183 sayfa

Sipariş toplama işlemi, bir depodaki en yüksek maliyetli faaliyettir. Sipariş Toplama Problemi'nin (STP) amacı toplayıcıların rotasını toplam sipariş toplama zamanını, dolayısıyla da sipariş toplama maliyetini minimize edecek şekilde belirlemektir. Bu çalışmada birbirine paralel sipariş toplama koridorlarının bulunduğu bir depo düşünülerek STP'nin değişik versiyonları ele alınmıştır. Literatürde bu problemin tek toplayıcılı versiyonu yaygın bir şekilde çalışılmış olmasına rağmen, çok toplayıcılı versiyonu üzerinde algoritmik yaklaşımlar bakımından yeterince durulmamıştır. Literatürde ayrıca, toplayıcının toplam dönüş sayısının sipariş toplama zamanına etkisi de ihmal edilmektedir.

Bu tezde, tek toplayıcının bulunduğu STP'nin hesaplama karmaşıklığı üzerine detaylı bir çalışma ortaya konmuştur. STP'nin ara koridor içermeyen durumları için öne sürülen algoritmayı kullanarak tek toplayıcılı ve ara koridorlu STP'yi çözmek için bir sezgisel prosedür öne sürülmüş ve bu prosedür rasgele üretilmiş problemler üzerinde denenerek sonuçları verilmiştir. Buna ek olarak, rotalama problemi için kullanılan "önce kümele, sonra rotala" ve "önce rotala, sonra kümele" sezgisellerinden faydalanan bir evrimsel algoritma geliştirilmiştir. Algoritma

“önce kümele, sonra rotala” ve “önce rotala, sonra kümele” yaklaşımlarına farklı ağırlıkların verilmesiyle rasgele üretilmiş problem setleri üzerinde test edilmiştir. Son olarak, paralel koridorlu depolarda dönüş sayısını minimize etme problemi için bir polinom zamanlı algoritma önerilmiştir.

Anahtar Kelimeler: Sipariş Toplama Problemi, Sezgiseller, Evrimsel Algoritmalar, Dönüş Cezaları, Hesaplama Karmaşıklığı

To Mom, Dad and Simi

ACKNOWLEDGMENTS

This thesis work has been completed with the help and support of a considerable number of people. It is a pleasure for me to have the opportunity to acknowledge their help and support here. I apologize and express my gratitude to the ones whose names have not been mentioned in this limited space.

My vocabulary of adjectives is not sufficient to describe the impact that Dr. Haldun Süral, my supervisor, has had in my personality and my way of looking at academic life. It has been indescribable to have him as a supervisor, as a colleague and as a friend ever since my sophomore year. I can only say that his supervision in this thesis is a very small part of what he has given to me both as an academician and as a person. It is unfortunate for me not to have him as my supervisor for my doctoral study, but I will be led by his guidance all throughout the rest of my life.

My last two years as a research assistant at the Industrial Engineering Department of the Middle East Technical University have shaped up my future career plans, and also my personality. During these two years I have had the opportunity to work in a wonderful environment, for which I am greatly indebted to all the academic staff of the department. Above all, I have most felt the encouragement and support of my two office mates: Güvenç Değirmenci, who has witnessed the beginning of my thesis work and has been very helpful with his experience and Büşra Atamer, whose understanding, sympathy, encouragement and help I can never pay back no matter what. I keep the hope of working together with them in my future career, and wish them the best with theirs. I am also thankful to İbrahim and Nihan Görmez Karahan for their help and advice during the first steps of this thesis, Aslı Gül Buğdacı for sharing the evenings that followed long and tiring work hours, and Aras and Şirin Barutçuoğlu for their lovely company, among all my other workmates.

This thesis would never be complete without the reviews and comments of the examining committee, Dr. Nur Evin Özdemirel, Dr. Selim Aktürk, Dr. Serhan Duran and Dr. Ömer Kırca. I appreciate the valuable feedback I have received from them. I feel privileged to have

known and assisted Dr. Meral Azizoğlu. I will be following her valuable advices in my both professional and personal life. I am also honored to have been enlightened by the great vision of Dr. Sencer Yeralan, and to have shared his friendship.

Above all the experience I have gained throughout the last two years, it has been the students I have assisted that led me to pursue an academic career. I owe a lot to the graduates of 2009 and prospective graduates of 2010 of the Industrial Engineering Department for the great memories. I wish them the best of luck in their future careers and hope to keep in touch along the way.

Having known him for more than thirteen years, it will be hard for me not to have Kıvanç Tos around for some time. His great friendship is highly acknowledged. I am also indebted to Esra Sarıarslan for her efforts in keeping my mind off the hard work during my spare time, and for her valuable motivation. Without the help of Kerem Demirtaş, computer implementation of the work in this thesis would never be that easy.

Last and the most, I would like to express my gratitude to the beloved members of my family. My mother, Meral Çelik, and my father, Salih Çelik, have always given me the freedom to choose the way I want to follow throughout my life and given me a hand when I need help. Having Semih Çelik as my brother is probably the best thing about my life. I am ensured that their love and support will be more than enough to put up with the distance we will be separated with during the next few years. Every single achievement I gain throughout is completely devoted to them.

TABLE OF CONTENTS

ABSTRACT iv

ÖZ vi

DEDICATION viii

ACKNOWLEDGMENTS ix

TABLE OF CONTENTS xi

LIST OF TABLES xv

LIST OF FIGURES xvii

CHAPTERS

1 INTRODUCTION 1

2 AN OVERVIEW OF WAREHOUSE DESIGN PROBLEMS 5

2.1 Strategic Level Problems 5

2.1.1 Warehouse Functions and Flows 6

2.1.2 Process Flow Design and Systems Selection Problems . . 8

2.1.3 Storage Capacity Planning Problems 10

2.2 Tactical Level Problems 11

2.2.1 Palletization and Block Stacking Problems 12

2.2.2 Layout Design Problems 13

2.2.3 Storage Assignment Problems 15

2.2.4 Zoning Problems 18

2.3 Operational Level Problems 20

2.3.1 Order Batching Problems 20

2.3.2 Picker Routing Problems 22

2.3.3 Order Accumulation and Sorting Problems 23

2.4 Conclusion on Warehouse Design Problems 24

3 REVIEW ON THE COMPLEXITY OF THE ORDER-PICKING PROBLEM 26

3.1 Basic Definitions and Notation 26

3.2 Polynomially Solvable Cases of the OPP and Heuristic Approaches . 30

3.2.1 The 1-OPP 31

3.2.2 The 2-OPP 31

3.2.3 The 3-OPP 38

3.2.4 Heuristic Procedures for the k-OPP with $k \geq 3$ 44

3.3 Complexity of the k-OPP with $k \geq 4$ 48

3.3.1 The Traveling Salesman Problem 48

3.3.2 The Steiner Traveling Salesman Problem (StSP) 55

3.3.3 Conclusion on Complexity 58

4 THE ORDER-PICKING PROBLEM IN A PARALLEL-AISLE WAREHOUSE: SINGLE-PICKER CASE 62

4.1 Subroutines of the Algorithm 63

4.1.1 Procedure *merge* 64

4.1.2 Procedure *reach* 69

4.1.3 Procedure *invert* 70

4.1.4 Procedure *improve* 70

4.2 The Merge-and-Reach Heuristic 73

4.3 Two Example Problems 73

4.4 Improvement: The Merge-and-Reach$^+$ Heuristic 75

4.5 Computational Experiments 81

4.5.1 Test Instances 81

4.5.2 Computational Results of the Suggested Heuristics 82

5 THE ORDER-PICKING PROBLEM IN A PARALLEL-AISLE WAREHOUSE: MULTIPLE-PICKER CASE 89

5.1 Literature Survey 90

5.1.1 Literature on k-OPP(s) 90

5.1.2 Literature on the Evolutionary Algorithms for VRP 92

5.1.3 Literature on the Evolutionary Algorithms for the Steiner Tree Problem 96

5.2 An Evolutionary Algorithm for k-OPP(s) with Load Capacities and Limits on Time 98

5.2.1 A Cluster-First, Route-Second Approach for the Case with Load Capacities 99

5.2.2 A Route-First, Cluster-Second Approach for the Case with Limits on Travel Time 103

5.2.3 General Parameter Settings 108

5.2.4 The Algorithm 111

5.2.5 Computational Experiments 113

6 THE ORDER-PICKING PROBLEM IN A PARALLEL-AISLE WAREHOUSE WITH TURN PENALTIES 127

6.1 Literature Survey on Turn Minimization in General Graphs 127

6.2 A Turn Minimizing Algorithm for 2-OPP 130

6.2.1 Conversion to the Rural Postman Problem 131

6.2.2 The Algorithm 137

6.2.3 Extensions 138

6.3 A Turn Minimizing Algorithm for k-OPP with $k \geq 3$ 145

6.3.1 Conversion to the Rural Postman Problem 145

6.3.2 The Algorithm 147

6.3.3 Extensions 151

7 CONCLUSION AND FURTHER RESEARCH DIRECTIONS 159

REFERENCES 163

APPENDICES

A ALTERNATIVE HEURISTIC SOLUTIONS TO THE PROBLEM IN FIGURE 3.1 169

A.1 Optimal Solution 169

A.2 S-shape Heuristic Solution 170

A.3 Largest Gap Heuristic Solution 171

A.4 Aisle-by-Aisle Heuristic Solution 172

A.5 Combined Heuristic Solution 173

B NP-COMPLETENESS OF THE TRAVELING SALESMAN PROBLEM . . 174

C COMPARISON OF HEURISTIC PROCEDURES FOR k-OPP 179

D APPLICATION OF THE 2-OPP ALGORITHM TO FIND THE TURN MINIMIZING TOUR 182

LIST OF TABLES

TABLES

Table 3.1 Resulting equivalence classes after adding the connection types in Figure 3.6 to each L_j^+ equivalence class, dashed lines indicating suboptimality or infeasibility 34

Table 3.2 Resulting equivalence classes after adding the connection types in Figure 3.5 to each L_j^- equivalence class, dashed lines indicating suboptimality 34

Table 3.3 Solution to the example problem in Figure 3.8, bold entries indicating part of the optimal solution, dashed lines indicating infeasiblity or suboptimality . . . 36

Table 3.4 Resulting equivalence classes L_j^{+y} after adding the connection types in Figure 3.5 to each L_j^- equivalence class, dashed lines indicating suboptimality 41

Table 3.5 Resulting equivalence classes L_j^{+x} after adding the connection types in Figure 3.5 to each L_j^{+y} equivalence class, dashed lines indicating suboptimality 42

Table 3.6 Resulting equivalence classes L_{j+1}^- after adding the connection types in Figure 3.6 to each L_j^{+x} equivalence class, dashed lines indicating infeasibility or suboptimality 43

Table 4.1 Resulting average travel times (in seconds) by the merge-and-reach and merge-and-reach$^+$ heuristic procedures as well as the average optimal times for our problem set, with the best heuristic results indicated in bold (2,000 instances for each setting) 83

Table 4.2 Percent deviations for the combined$^+$ heuristic by Roodbergen and De Koster [73], the merge-and-reach and merge-and-reach$^+$ heuristics, with the best heuristic performance indicated in bold (2,000 instances for each setting) 84

Table 4.3 The number of times each cut procedure gives the best solution for our problem set, bold entries indicating statistical significance of that cut value on the performance of the heuristic, $\alpha = 0.99$ (2,000 instances for each setting) 85

Table 4.4 Percent deviations of the 2-opt improvement procedure from the optimal travel times, and the percent improvement of the 2-opt scheme over the merge-and-reach solutions for the preliminary runs (200 instances for each setting) . . . 88

Table 5.1 Percent deviations from single picker optima and percent improvements over random search solutions and initial population incumbents for the cases α = 0% and α = 100% (averages of 30 instances), bold entries indicating superiority over the other method 121

Table 5.2 Percent deviations from single picker optima and percent improvements over random search solutions and initial population incumbents for the cases α = 25% and α = 75% (averages of 30 instances), bold entries indicating superiority over the other method 124

Table C.1 Resulting average travel times (in seconds) by the largest gap, S-shape and aisle-by-aisle heuristic procedures for the problem set of Roodbergen and De Koster [73], with the best heuristic results indicated in bold (2,000 instances for each setting) 180

Table C.2 Resulting average travel times (in seconds) by the combined, and combined$^+$ heuristic procedures and average optimal times for the problem set of Roodbergen and De Koster [73], with the best heuristic results indicated in bold (2,000 instances for each setting) 181

Table D.1 Resulting equivalence classes after adding the possible connection types in Figure 3.6 to each L_j^+ equivalence class, dashed lines indicating suboptimality or infeasibility 183

Table D.2 Resulting equivalence classes after adding the possible connection types in Figure 3.5 to each L_j^- equivalence class, dashed lines indicating suboptimality . . 183

LIST OF FIGURES

FIGURES

Figure 2.1 The functions and flows between them in a typical warehouse, where the arrows represent the following: (1) direct putaway to reserve, (2.1) replenishment to case picking, (2.2) replenishment to broken case picking, (3) direct putaway to primary storage, (4) accumulation and sortation from primary storage, (5) shipment and (6) cross-docking 7

Figure 2.2 A summary of the warehouse design problems discussed in this chapter, arrows indicating hierarchical effects 25

Figure 3.1 A parallel-aisle warehouse with 2 middle aisles and 25 pick items 27

Figure 3.2 Graph representation of the warehouse in Figure 3.1 28

Figure 3.3 A general grid graph (a) and a solid grid graph (b) 30

Figure 3.4 Recursive definition of an S-P graph: If G in (a) is S-P, then by deleting (uv) and adding either series (b) or parallel edges (c), we obtain G' as an S-P graph 31

Figure 3.5 The six possible connection types within the aisles 33

Figure 3.6 The five possible connection types between the aisles 33

Figure 3.7 Procedure *Ratliff_Rosenthal* 35

Figure 3.8 An example 2-OPP with 6 aisles and 8 items, where v_0 corresponds to the depot 36

Figure 3.9 The optimal route for the problem given in Figure 3.8 37

Figure 3.10 Formation of an S-P graph for a 2-OPP with 4 aisles and 5 items: (a) The initial graph, (b) Formation of the second aisle in Step 2 and the duplicated edge in step 3, (c) End of Step 3, (d) Addition of items in Step 4 38

Figure 3.11 The fourteen possible connection types between the aisles 40

Figure 3.12 Summary of the procedure of Roodbergen and De Koster [72] for the 3-OPP 44

Figure 3.13 An example problem with a single middle aisle, 7 pick aisles and 15 items 45

Figure 3.14 Optimal solution of the problem in Figure 3.13 45

Figure 3.15 Odd (a) and even (b) alternating strips 52

Figure 3.16 (a) $R(10,7)$ (b) $\eta(4,3)$ (c) $\omega(4,3)$ 53

Figure 3.17 Summary of the complexity of the OPP-related problems 61

Figure 4.1 Two overlapping partial solutions with no single vertical edge 65

Figure 4.2 The resulting merged solution for the subproblem in Figure 4.1 65

Figure 4.3 Two overlapping solutions with six single vertical edges 66

Figure 4.4 (a) The partial solution to the example subproblem in Figure 4.3 when edge deletion is complete, (b) the resulting solution after matching 67

Figure 4.5 Procedure *merge* 68

Figure 4.6 Procedure *reach* 70

Figure 4.7 A pair of non-overlapping solutions 71

Figure 4.8 Intermediate solution to the subproblem in Figure 4.7 after the reach move 71

Figure 4.9 The resulting reach solution to the subproblem in Figure 4.7 72

Figure 4.10 Procedure *invert* 72

Figure 4.11 Procedure *improve* 73

Figure 4.12 Heuristic *merge_and_reach* 74

Figure 4.13 The set of 2-OPP solutions to each block of the problem in Figure 3.1 . . . 76

Figure 4.14 The merge-and-reach heuristic solution for the warehouse in Figure 3.1 . . 77

Figure 4.15 An example problem with 9 blocks, 7 aisles and 10 items 78

Figure 4.16 Intermediate solution to the problem in Figure 4.15 79

Figure 4.17 The merge-and-reach route for the problem in Figure 4.15, with a total length of 207 units 80

Figure 5.1 Procedure *hill − climbing* 92

Figure 5.2 An example of order crossover (a) the parents and (b) the offspring 95

Figure 5.3 An example chromosome for the case with 10 items 100

Figure 5.4 Procedure *cfrs_fitness* 100

Figure 5.5 Procedure *repair* 101

Figure 5.6 2-point crossover between parents including 10 items 102

Figure 5.7 An example swap operation with 10 items 103

Figure 5.8 An example of the encoding scheme with 10 items: (a) the tour, (b) the corresponding individual 103

Figure 5.9 (a) An example tour with 10 items and the depot, (b) The corresponding order is represented as a graph whose shortest path problem solution gives the assignment of the items to the pickers and their picking orders 104

Figure 5.10 An example nearest neighbor crossover: (a) the distance matrix with the first row and column indicating the node index numbers (b) the two parents, (c) the union graph G, (d) the resulting tour, (e) the offspring 106

Figure 5.11 An example for the displacement operator: (a) the individual, (b) the mutated individual 107

Figure 5.12 The steps of the combined evolutionary algorithm 112

Figure 5.13 The depot points for each of the block settings: (a) single block, (b) 2 blocks, (c) 4 blocks, (d) 8 blocks 115

Figure 5.14 Taguchi analysis results for $\alpha = 100\%$ 117

Figure 5.15 Taguchi analysis results for $\alpha = 0\%$ 119

Figure 6.1 An illustration of Move Type 1 - Case 1, where no turn is incurred 132

Figure 6.2 An illustration of move Type 1 - Case 2: (i) with partial traversal, (ii) with complete traversal. Both moves require 2 turns in total 133

Figure 6.3 An illustration of Move Type 2: (i) with partial traversal, incurring 4 turns, (ii) with complete traversal, incurring 2 turns 133

Figure 6.4 The only case when partial traversal can possibly perform better than complete traversal: (a) partial traversal of aisle k followed by reaching the depot incurs 5 turns, (b) complete traversal of aisle k followed by reaching the depot also incurs 5 turns 134

Figure 6.5 An illustration of Move Type 3 - Case 1: (i) with partial traversal, (ii) with complete traversal, both cases incurring 3 turns 135

Figure 6.6 An illustration of Move Type 3 - Case 2, incurring 1 turn 136

Figure 6.7 Algorithm *turn_2-OPP* 138

Figure 6.8 When there are even non-empty aisles, complete traversal of aisle k is possible and performs better than partial traversal 140

Figure 6.9 When there are odd non-empty aisles and items on both sides of v_0, a U-turn at aisle k is an alternative optimal, (i) the U-turn with partial traversal, (ii) the U-turn with complete traversal. Both incur 2 extra turns. 140

Figure 6.10 There are odd non-empty aisles and an item on only one side of v_0, (i) complete traversal results in 2 extra turns, (ii) partial traversal of k 141

Figure 6.11 Algorithm *turn_2-OPP_modified* 142

Figure 6.12 An example problem with 6 pick aisles, 5 non-empty aisles and 12 items . 143

Figure 6.13 Step-by-step solution to the problem in Figure 6.12, (i) required edges, (ii) the subtours, (iii) the optimal solution with 10 turns 144

Figure 6.14 Algorithm *turn_k-OPP* 148

Figure 6.15 An example problem with 2 blocks, 5 non-empty pick aisles and 14 items, with the required edges shown in bold and the unified edge shown in dashed lines 149

Figure 6.16 The optimal solution to the turn minimization problem given in Figure 6.15 150

Figure 6.17 Algorithm $turn_k - OPP_modified$ 154

Figure 6.18 An example problem with 2 blocks, 5 non-empty pick aisles and 14 items, with required edges shown in bold 155

Figure 6.19 Optimal solution to the example problem in Figure 6.18, which incurs 10 turns 156

Figure 6.20 An example problem with 2 blocks, 5 non-empty pick aisles and 14 items, the required edges shown in bold 157

Figure 6.21 Optimal solution to the problem in figure 6.20 with 12 turns 158

Figure A.1 Optimal solution of the example problem in Figure 3.1, with a total travel distance of 164 units 169

Figure A.2 S-shape heuristic solution to the example problem in Figure 3.1, with a total travel distance of 192 units 170

Figure A.3 Largest gap heuristic solution to the example problem in Figure 3.1, with a total travel distance of 190 units 171

Figure A.4 Aisle-by-aisle heuristic solution for the problem in Figure 3.1, with a total travel distance of 190 units 172

Figure A.5 Combined heuristic solution for the example problem in Figure 3.1, with a total travel distance of 182 units 173

Figure B.1 Transformation from Hamiltonian Cycle for Grid Graphs to the General TSP [52] 177

CHAPTER 1

INTRODUCTION

Today's competitive business environment puts great emphasis on the efficient and effective management of supply chains. An important aspect regarding management of supply chains is the management of warehouses, which, according to Bartholdi and Hackman [7], serve three main purposes. First of all, they provide better matching of supply with demand by allowing the space to build up inventories in order to buffer the supply chain against surges in demand. Secondly, they provide an opportunity for consolidating products so that the fixed cost of transporting the products to the customers is decreased. Lastly, an increasing number of warehouses serves the purpose of value-added processing, which allows for product differentiation to serve the customers in a faster and less costly manner. Therefore better management of warehouses not only contributes to faster response times to the demands of the customers, but it also decreases overall costs in the supply chain.

A warehouse basically serves four main functions: receiving, storage, order-picking and shipping. Receiving activities include orderly receipt of all the materials coming into the warehouse, assuring the quality and quantity of the materials are as ordered, and transferring materials to storage. Storage activities consist of physical containment of merchandise awaiting demand. Order-picking refers to the activity of collecting the items corresponding to an order or a set of orders. Shipping activities may include control for completeness, packaging, accumulation and loading of the merchandise. Order-picking operations constitute the costliest activities in a warehouse. When warehousing costs are broken down into components, Tompkins et al. [79] have found out that 55% of all warehousing costs in the United States can be attributed to the activities of order-picking. There are various decisions that have to be made in order to perform efficient and effective order-picking. These include tactical decisions like assignment of products to the storage areas and zoning of picking areas; as well as operational

decisions like batching of orders, routing of order-pickers, accumulation and sorting of orders. Tompkins et al. [79] have analyzed the time spent by a picker and have found out that half of the picking time is occupied by travelling. Therefore, providing efficient routing methods can significantly improve the response time to an order, thereby decreasing the warehousing costs.

The order-picking problem (OPP), which is the main problem discussed in this thesis, is the problem of picking and preparation of the items corresponding to an order in the shortest amount possible. The objective is to determine the route that minimizes the time required for this operation. The problem is applicable for both conventional warehouses that employ off-the-shelf manual order-picking, and automated warehouses that employ automatic storage/retrieval systems (AS/RS). Several approaches to tackle this problem exist in the literature, and almost all of these try to minimize the travel distance as the objective function. In this study, we tackle the OPP in parallel-aisle warehouses, in which the items are located on pick aisles that are parallel to each other. The warehouse might also include middle aisles, which are orthogonal to the pick aisles, to better aid the crossing of the picker between the pick aisles.

The OPP has received considerable amount of attention in the literature: There exists a polynomial time algorithm for routing of the pickers in a warehouse with no middle aisle, due to Ratliff and Rosenthal [71]; and another polynomial time algorithm for the case in which a single middle aisle exists, due to Roodbergen and De Koster [72]. For the remaining cases, various heuristic procedures have been proposed in the literature. Although some of these heuristics are quite simple to implement, they obtain results which are not close to the optimal. Those which provide good results do so for a certain set of problems. This motivates the development of a more robust heuristic. Consequently, we propose a robust heuristic procedure for the OPP in warehouses with middle aisles, and perform computational experiments on randomly generated problems in order to test its performance and compare the results with the alternative solution procedures in the literature.

The OPP with the objective of distance minimization is polynomially solvable for the cases with no middle aisle, or a single middle aisle (due to [71] and [72]); but the complexity of the general problem has not been discussed. In this thesis, we provide an extensive review of the literature on the problems that are relevant to the order-picking problem in terms of the

complexity argument. These include the Traveling Salesman Problem, its special cases such as the Steiner Traveling Salesman Problem, and the Graphical Steiner Traveling Salesman Problem; as well as the Steiner Tree Problem.

There are cases in which employing a single picker to collect the items corresponding to an order is not plausible. The main assumption in the single-picker OPP is that the picker is uncapacitated. There might be cases in which the order size is too large to be handled by a single picker. Another motivation that might trigger the need to employ multiple pickers is that there might be limits on the lead time or shipment time of the products so that employing a single picker cannot meet this limit. Lastly, the warehouse might be divided into zones, each of which is assigned to a picker. Due to these reasons, it is worthwhile to consider the order-picking problem involving multiple pickers. Unlike the single-picker case, this version of the problem has not received much attention in the literature. The only algorithmic study, to the best of our knowledge, is due to Geng et al. [37], who propose a very-large scale neighborhood approach for the problem. For this problem, we propose an evolutionary algorithm that makes use of the well-known cluster-first, route-second and route-first, cluster-second approaches to the vehicle routing problem. Since the problem set of [37] is not available, we generate a random set of problems and test the performance of various settings of the evolutionary algorithm on these problems.

The picking time of an order depends on five main factors: the travel time of the picker, the acceleration/deceleration of the picking vehicle, entering/leaving the aisles, picking the items and turning the corners or making U-turns. The first of these factors is handled by the procedures in the literature and in this study that try to minimize the travel time of the picker. The acceleration/deceleration times, time required to enter/exit the aisles and to pick the items is constant and independent of the route of the picker. However, the time required to turn the corners and make U-turns takes up considerable amount of time, especially when picking is performed using automated vehicles. For this end, one also needs to consider the turns incurred by the order-picker. Despite the fact that distance minimization problems that include turn penalties as well as turn minimization problems have been extensively studied for general graphs in the literature, the OPP has never been studied considering the effects of turns. In this study, we propose an algorithm for the problem of minimizing the number of turns in a parallel-aisle warehouse. Although the turn minimization problems for general graphs are generally NP-hard, our algorithm runs in polynomial time for the special case in

OPP graphs. The algorithm covers the cases where the middle aisles might or might not exist, or where the depot might be at any point in the warehouse.

The outline of this thesis is as follows. We first discuss the warehouse design problems in general in Chapter 2, in order to discover where the order-picking problem lies among these and its relative importance. In Chapter 3, we give the basic definitions regarding the OPP and the notation used throughout this study, followed by a review of the issues on the computational complexity of the problem. Chapter 4 proposes a robust heuristic procedure for the OPP with a single picker. The multiple-picker case is discussed and an evolutionary approach is proposed in Chapter 5, which is followed by the formulation of a polynomial time turn-minimizing algorithm for the OPP in Chapter 6. We conclude the study in Chapter 7.

CHAPTER 2

AN OVERVIEW OF WAREHOUSE DESIGN PROBLEMS

The order-picking problem (OPP) lies among many other interdependent warehouse design problems, and the aim of this chapter is to provide a summary of these warehouse design problems, to discuss their connections with the OPP, and to provide results from the literature. In discussing these problems, there are two different approaches. The first one, as employed by Gu et al. [43], classifies the problems according to the warehouse functions they serve, such as receiving, shipping, order-picking, etc. The second approach, as in Rouwenhorst et al. [74], considers the problems according to the decision making levels they address: strategic, tactical or operational. In this chapter, we take a similar approach to the one taken by Rouwenhorst et al. [74], and classify the problems with regard to the decision levels they correspond to: under either strategic, tactical or operational level problems.

The organization of the chapter is as follows. Section 2.1 deals with the strategic level warehouse design problems, which mainly consist of equipment and systems selection among with process flow design. Section 2.2 considers the tactical level problems, which mainly include layout determination, forward and reserve storage area determination, and selection of the storage concept (random, dedicated, class-based, etc.). In Section 2.3 we discuss the operational level problems, consisting of order batching, picker routing, order accumulation, and sorting. Lastly, we derive conclusions on the literature overview in Section 2.4

2.1 Strategic Level Problems

Th strategic level problems address decisions that include high investments and that have impact on the long term. Regarding strategic level decisions, Rouwenhorst et al. [74] discuss

two main groups of problems. These two problem types both address the selection of process flow design and types of systems (storage system, sorting system, etc.), distinguished by their main considerations. The first type of problem considers technical capabilities of the systems and equipments while making the selection. The second one is concerned with economic considerations, trying to minimize the investment and operational costs. Naturally, the two criteria (technical capabilities and economic issues) are to be considered together when designing a warehouse process flow and equipment system. Owing to this, we take these two groups of problems together here, and try to take an overall view. In addition to these problems, we also discuss the storage capacity planning problem, which is also considered among the strategic level problems by Cormier [19].

Before discussing the warehouse process flow and systems design problems, an introduction to the warehouse processes is given. Then, the approaches to these problems are discussed. Lastly, storage capacity planning problems are considered.

2.1.1 Warehouse Functions and Flows

As mentioned by Bartholdi and Hackman [7], warehouse processes can be classified into two main headings: *inbound processes* and *outbound processes*. The inbound processes, which refer to the activities that take place between the receipt of the product until an order arrives for it, can further be broken down into the activities of *receiving* and *put-away*. The outbound processes consist of the activities carried out from the time an order is received for the product until its shipment out of the warehouse. These activities can also be broken down into *order-picking* and *checking, packing and shipping*.

Figure 2.1, which has been adapted from Tompkins et al. [79], summarizes the functions of a warehouse and illustrates the possible flows among these functions. In this setting, pallet picking, case picking and broken case picking refer to the order-picking activities with respect to the sizes of the items to be picked. In this sense, pallet picking involves a larger size and amount of pick-items, and requires less labor, as material handling is reduced compared to case and broken case picking. Due to this property of pallet picking, there is a greater opportunity for automation. Broken case picking requires handling the smallest units of measure in the warehouse and because of the size and variety of the stock keeping units to be handled, automation is almost impossible. Therefore this type of order-picking is performed manually

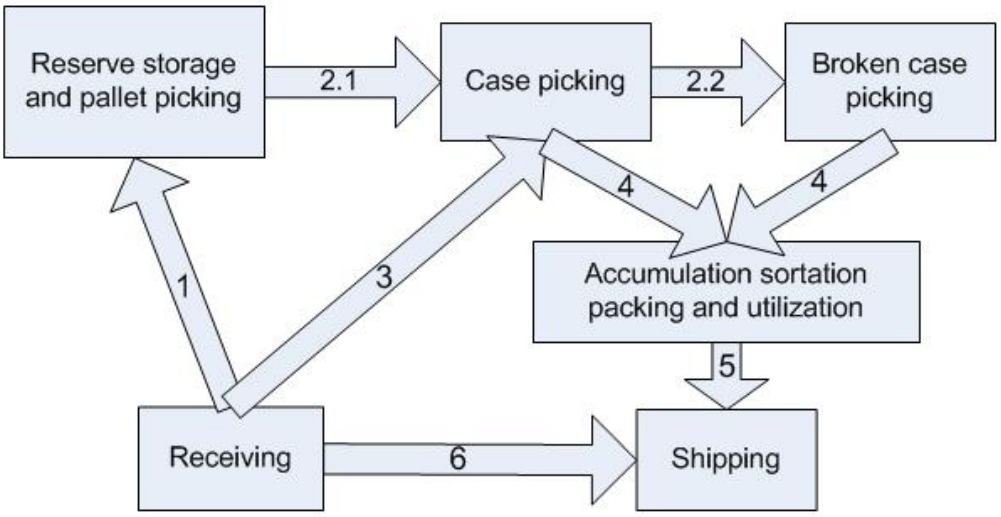

Figure 2.1: The functions and flows between them in a typical warehouse, where the arrows represent the following: (1) direct putaway to reserve, (2.1) replenishment to case picking, (2.2) replenishment to broken case picking, (3) direct putaway to primary storage, (4) accumulation and sortation from primary storage, (5) shipment and (6) cross-docking

and requires a high amount of labor.

Another distinction that can be observed from Figure 2.1 is the one between *reserve storage* and *forward*, i.e. *primary storage*. In the reserve storage, which is also called the bulk storage, the products are stored in larger amounts, such as in pallet racks. Material handling is less compared to the forward storage. The main concern here is storing the products in the most economical way. On the other hand, the forward storage, also called the primary storage, stores the products in smaller amounts, such as in shelves. The need for handling the products in smaller amounts causes the requirement for more material handling. The main concern in a forward storage area is to provide the order-pickers with easier access to the products for retrieval.

The functions in a warehouse, adapted from [79] and summarized in Figure 2.1, can be listed as follows:

1. *Receiving* includes the orderly receipt of materials into the warehouse, the assurance of their quality and quantity, and disbursement of these to the functions that need them.

2. *Putaway* operations include the placement of the materials or products in the storage.

Material handling and placement are involved in this process. The putaway of an incoming item can be done in two ways. In *putaway to reserve*, the item is placed in the reserve storage area, supposedly in larger amounts. In *putaway to primary*, the item is placed in the forward storage area in smaller amounts.

3. *Storage* is the process that the product goes through between its putaway and arrival of its demand. How an item is stored depends on the size and quantity of that item in inventory, and the material handling characteristics of that item and its container.

4. *Order-picking*, as defined in the introduction, refers to the activity of collecting the items corresponding to an order or a set of orders. This is basically the main service given by the warehouse to its customer and the main motivation in how a warehouse is designed.

5. *Replenishment* is the process of feeding the primary storage area from the reserve storage. As the item undergoes replenishment from the first putaway until shipping, the size of replenishment gets smaller, and the material handling effort increases. In Figure 2.1, the arrows labeled 2.1 and 2.2 indicate such a case, where an item is stored in pallets in the reserve area, replenished into a storage area where it is stored in cases, and further replenished into a storage area where it is stored in broken cases.

6. When an order includes more than one item, the picked items must go through *sorting* and the order must be *accumulated*. This process can be accomplished during or after order-picking.

7. The *shipping* process may include checking the order for completeness, packaging, accumulating orders and loading.

8. *Cross-docking* is a special operation where an item goes directly to shipping upon its receipt, where it may possibly go through an accumulation process.

2.1.2 Process Flow Design and Systems Selection Problems

The process flow design and selection of the types of warehousing systems form the first class of strategic level warehouse design problems. The overall problem consists of selecting and sequencing a subset of the processes discussed in the preceding section so that the functions

that the warehouse is aimed to serve are accomplished. Several design problems can be classified under this type. These include whether or not to have a separate reserve storage area, whether or not to allow different types of storage systems, whether or not to batch orders, the type(s) of storage system(s) to employ, how many storage units to locate, the type(s) of sorting system(s), if applied, and so on. Which processes of the warehouse to automate and which to perform manually can also be considered as another problem.

In the literature, these problems have not received as much attention as the tactical and operational level warehouse design problems have. Due to the complexity of the overall problem, most of the studies employ a hierarchical solution procedure. For instance, in Gray et al. [41], a set of problems including the warehouse layout, equipment and technology selection, item location, zoning, picker routing, pick list generation, and order batching is considered. A hierarchical multi-stage solution approach is proposed employing a series of mathematical models in order to reduce the solution space to a number of alternatives that dominate the others. Then, simulation is applied to validate the solution process and to further fine-tune the resulting policies. The procedure is employed on a real-life case in a spare-parts warehouse and is observed to come up with significant savings and increased warehouse efficiency.

McGinnis et al. [62] aim to formalize the process flow design and appropriate warehousing systems selection process by proposing a conceptual framework for warehouse design. The conceptual design of a warehouse is represented using the notion of a *function flow network*, which is basically a network consisting of nodes referring to warehouse functions such as receiving, storage functions, pick-line, sorting, accumulation, etc. and the arrows representing the aggregate flow of material between the functions. It is stated that the first series of decisions in warehouse design, being the architectural ones, consist of determining the arcs and nodes of the functional flow network. The authors then go on to define the *expert warehouse design process*, which starts by profiling, determining the requirements that the warehouse must satisfy based on historical data. Then, functional requirements and costs are determined, providing a basis for the architectural decisions. These decisions result in updating the profiling process, and the design cycle goes on until a final decision on the design process is reached by the procedure.

Bodner et al. [12] extend the solution approach by McGinnis et al. [62] into two phases. In the first one, the design process in practice is studied using an ethnographic methodology

in order to formalize the design processes, information usage, decision making criteria and evaluation procedures. The second phase tries to formalize the research results for formulating a scientific warehouse design methodology and to implement computational tools to aid designers in practice. Generally, the design process is initiated by gathering of the data from the clients, which usually contains information on products (or SKUs), vendor shipment history, inventory history and customer order history. The next phase is profiling, in which the data are analyzed for certain patterns that can affect the design. The output of the process is the architectural design. Finally, the focus shifts to the specification and optimization of the systems that implement the functions. The general idea to formalize this process is using computational decision support tools, which will replace ad-hoc spreadsheet and database tools. This is done using a web-based platform hosted on a central server, allowing higher accessibility to expert designers. The user enters input data through queries, after which the data are analyzed by the design decision support system through various design modules, MIP solver and layout generators. Finally, possible alternatives and visualizations are returned to the user. The approach is applied on a case study, in which design decisions were needed in several areas, among which four are mentioned. The first decision is whether to redesign all processes in the warehouse, or a subset of them. The second decision involves whether to discontinue or make more efficient the use of trailers for storing inventories. Thirdly, a decision of whether or not to use automated shipping/receiving and storage systems is to be made. The last decision is on segmentation of the customers or eliminating broken case orders from small customers. In the end, the following four decisions were made: Redesigning and reconfiguring a subset of processes, redesigning the operation of trailers, limited automation in the receiving process, and limited customer segmentation.

2.1.3 Storage Capacity Planning Problems

The last group of strategic level warehouse design problems to be discussed are the storage capacity planning problems. These problems are strategic in the sense that determination of a capacity has a long term impact. The objective of these problems is to determine the optimal capacity of the warehouse in each period of the planning horizon so that the total relevant costs are minimized while satisfying the demand for products. The literature usually considers developing mathematical models either for the storage capacity planning problem alone or its combination with other strategic level problems.

Cormier and Gunn [20] consider the determination of the optimal warehouse capacity expansion schedule and the underlying multi-item inventory policy in an environment where the demands vary arbitrarily during a finite planning horizon, assuming dedicated storage policy, uniform stock withdrawal within a period and instantaneous replenishment. The problem is converted to the shortest path problem on a network and a dynamic programming approach is developed, where the state variables correspond to the total capacity of the warehouse and the stages correspond to time periods. For each period, the inventory amounts based on the capacity and demand in that period are determined. The demand grows linearly or exponentially. On the two sets of problems, the effects of annual demand growth and fixed and variable costs of facility expansion on the optimal policies are discussed.

Aghezzaf [2] discusses the joint problem of capacity planning and warehouse location for a supply chain, where the only cause of uncertainty is in demand. First, an integer programming formulation for the case when demand is known with certainty, is proposed. The model handles warehouse location and capacity expansion in an integrated manner, and assumes that transfer of commodities between warehouses is possible for better capacity utilization. It is shown that this deterministic problem is NP-hard by reduction to the single-source capacitated lot-sizing problem. By taking into account the uncertainty in demand, a robust optimization model is proposed. The model aims to come up with warehouse location and capacity expansion policies that are robust under every scenario applied. In order to obtain robust solutions, Lagrangian decomposition is applied.

2.2 Tactical Level Problems

The tactical level problems aim to come up with decisions that have a medium term effect and require moderate investments. These problems take strategic level decisions as constraints and impose further constraints on the operational level problems. Although some of the studies we consider here may include partially strategic level or operational level problems as well, we distinguish the tactical level problems based on the framework provided by Cormier [19] and include palletization, block stacking, layout design, storage assignment and zoning problems in the class of tactical level problems.

2.2.1 Palletization and Block Stacking Problems

Palletization can be defined to be the formation of a pallet load through a palletizer by stacking the incoming boxes together. Various conventional methods can be used for the process. Tompkins et al. [79] list these as the block pattern, row pattern, pinwheel pattern, honeycomb pattern, split-row pattern, split-pinwheel pattern and brick pattern. Solutions to the palletization problems try to achieve the objectives of maximum pallet volume utilization, pallet loading time reduction, demand satisfaction, loading stability maximization and WIP reduction. Block stacking is the procedure of storing large quantities of palletized or boxed products on top of each other in stacks, without using racks. Usually, the main objective in block stacking is space utilization.

Palletization problems have not received much attention in the literature. Yet the study by Abdou and El-Masry [1] is worth citing. In this study, a three-dimensional palletization problem is considered where one pallet is loaded at a time, boxes arrive in random sequences, the box types are predefined and random stacking is used to load the boxes on the pallet. The objectives are listed as maximization of the volumetric pallet utilization, maximization of the load stability, minimization of the work-in-process inventory, and minimization of the robotic palletization time. The main contribution of the study is the inclusion of load stability as a performance measure. Based on these, a heuristic procedure consisting of two parts is proposed. The first part aims to build stable blocks out of the boxes in the WIP and those on the conveyor, whereas the second part stacks the boxes on the pallet considering the demand constraints. The heuristic procedure is applied on three case studies and the validation of the procedure has been made using simulation. As a result, the heuristic procedure is observed to outperform the existing ones in the literature in terms of all the performance measures.

For the block stacking problems, efficient algorithms are only available to compute the optimal lane depth for a single product, assuming that all lanes have equal depths. Goetschalckx and Ratliff [40] propose an algorithm to compute the optimal number of lanes and the optimal lane depth for a single product, when the lanes are allowed to have different depths. It is shown that the optimal lane depth follows a triangular pattern. The optimal lane depth pattern is compared experimentally with several heuristic patterns. Several near-optimal and efficient heuristics are identified. In most warehouses, the lane depth on each side of a single aisle is kept constant for layout and material flow purposes. An optimal algorithm to assign a single

product to such limited number of lane depths is also derived. Based upon this algorithm, a procedure for determining the lane depths and the number of lanes in a warehouse for storing multiple products is developed. If the warehouse is perfectly balanced, then the procedure minimizes the required warehouse area.

2.2.2 Layout Design Problems

As De Koster et al. [24] put it, the layout design problem can be classified into two parts with respect to the order-picking problem. The first group of problems deal with the layout of the facility containing the order-picking system. These types of problems are called as the *facility layout problems*. The main decisions are where to locate the departments such as receiving, picking, storage, sorting, shipping, etc. The second class considers the layout within the order-picking system. These types of problems are referred to as the *internal layout design* or *aisle configuration problems*. The main decision consists of determining the number of blocks and the number, length and width of aisles in each block of the picking area.

A second classification of the layout design problems with respect to the order-picking operations can be made in terms of the type of order-picking operations carried out within the warehouse. Again, two classes can be mentioned. The first class is the layout design problem for the manual order-picking systems, where human operators are employed for picking the items. The second type is the layout design for automated storage/retrieval systems (AS/RS).

The studies regarding the layout design problem concerning the manual order-picking systems are not many. Among these, the study by Queirolo et al. [70] can be mentioned, where the layout design of a warehouse is considered for a soft drink manufacturer, which includes 11 blocks with a total number of 4,408 storage cells. The total length of each block puts a constraint on the number of pallets that can be used. The authors show that the corresponding layout design problem is NP-hard. For the solution of this problem, a genetic algorithm that makes use of a simulation procedure is proposed. Four parents are used to generate a single offspring. The crossover operator, called the *basic gene scanning operator*, selects the mostly occurring gene in the parents for each gene of the offspring and breaks ties randomly. Five different heuristic procedures are applied as mutation operators. The fitness function evaluation, which takes up considerable time if evaluated by the genetic algorithm, is approximated using simulation. A manual fine tuning procedure is applied in the end.

Considering layout design problems for automated storage/retrieval systems (AS/RS), the literature is more abundant. An example that can be given is the study by Larson et al. [56], in which the layout in a warehouse employing an AS/RS with class-based storage assignment policy (the storage assignment policies will be discussed in the next section) is considered. The overall objective is to come up with a layout design that most effectively allows the usage of floor space and equipment. It is assumed that the material enters and exits the facility from single input and output points (these input and output points are not necessarily identical), each storage region consists of only one storage medium, and all primary aisles are parallel and of the same length. A three-phase heuristic algorithm is proposed for the design of the layout and assignment of the items to the storage points. In the first phase, the aisle layout and dimensions are determined by first determining the path from the input point to the output point and establishing an initial primary aisle, then determining the slot dimensions based on the dimension requirements of the storage mediums and the storage zone length accordingly. The second phase determines the storage medium using the necessary data for the items, coming up with the number of storage locations needed for each item. The last phase determines the allocation of storage space to the items by using a heuristic classification algorithm. The three-phase algorithm is applied on a case study conducted for a manufacturer of personal hygiene products and a decrease of 17% in the storage area requirement is observed, compared to the existing system.

A more recent layout design study concerning an AS/RS is by De Koster et al [25]. A 3-dimensional compact AS/RS is considered, consisting of an automated crane guiding the horizontal and vertical movements of the pallets and a gravity or powered conveying mechanism to control the depth movement of the rack. The advantage of such a system is the ability to conduct full automation, enabling storage and retrieval of items on a small area. The main contribution of the study is in the sense that it provides both travel time estimation and system dimensioning for a 3-dimensional storage system. In developing a closed form expression for the expected travel time, first, a single-command cycle (where the crane can perform either pick-up or drop-off of an item in one cycle) and random storage assignment policy (to be discussed in the next section) is assumed. Based on the expected travel time, a mathematical model for the dimensioning of the racks is formulated. It is then shown that when the racks are square-in time (the length -in time- of the rack is equal to the height -in time- of the rack), the expected travel time is minimized. Afterwards, this result is extended to the case of dual-

command cycles (where the crane can perform pick-up and drop-off of an item in the same cycle), and it is shown that the square-in-time racks minimize the expected travel time for the case of dual-command cycles as well.

2.2.3 Storage Assignment Problems

The storage operation takes place between the putting-away of an item to the stock point until a demand arrives for it. The objectives in designing a storage system for a warehouse include the maximization of space utilization, equipment utilization, labor utilization, material accessibility and material protection while satisfying the customer demand at the same time. We will discuss the storage assignment problems in two headings. First, we consider the forward-reserve allocation problem and then, we investigate the storage assignment policies.

It is common for warehouses to have separate areas for the bulk stock (in the reserve area) and the pick stock (in the forward area), as this distinction speeds up the order-picking process. Yet such a separation brings about several issues to be considered. The first problem is to determine how much of each item will be stored in the forward area and how much of it will be stored in the reserve area. The demand frequencies and demand quantities of the items can provide useful for the solution of this problem. The second problem is to determine the overall sizes of the forward and reserve areas. The trade-off between the replenishment efforts and order-picking time must be considered here. The smaller the forward area, the less time it will take the order-pickers to pick the items on their lists, but the more will be the need to replenish the stocks in the forward area more frequently. Basically, these two problems constitute the forward-reserve allocation problem.

Van Den Berg et al. [81] consider the forward-reserve allocation problem in a warehouse where the orders are picked in a certain period of time. The objective is to determine which replenishments minimize the expected amount of time to replenish the forward area. It is assumed that the storage capacities of the forward and reserve areas are known, picking of an item from the reserve area is also possible, concurrent replenishment is possible with known replenishment times, and the demand of each item follows normal distribution. A 0-1 integer programming model is formulated for the case where there is no limit on the amount of concurrent replenishment. The problem is shown to be NP-hard by reduction to the knapsack problem. A greedy knapsack heuristic, which provides near-optimal solutions, is proposed for

the problem. A second 0-1 integer programming model is formulated for the case where there is a limit on the amount of the concurrent replenishments, possibly due to a limit on personnel, or to avoid congestion within the warehouse. A relaxation improvement heuristic, which makes use of the rounding-down of the continuous solution values from the LP relaxation of the problem. The algorithms are tested on a set of generated problems, and it is observed that the algorithm comes up with good results in terms of the expected labor time.

Bartholdi and Hackman [7] devote a chapter of their book "Warehouse & Distribution Science" to the forward-reserve allocation problem. Different from Van Den Berg et al. [81], it is assumed that the entire replenishment quantity of a stock-keeping unit can be carried in one trip. The question of how much of each stock-keeping unit to store in the forward area is considered. It is shown that in order to minimize labor to maintain a forward area, each unit of storage space must be replenished at the same frequency. Then, two widely-used strategies in order to answer the same question are introduced, the first one being to allocate the same amount of space to each stock-keeping unit and the second one being to store an equal time supply of each stock-keeping unit. Contrary to the common belief that the second strategy is superior over the first one, they prove that the two strategies require the same total amount of replenishments. Then, the second question of which stock-keeping units to include in the forward area is tried to be answered, providing expressions for the stock-keeping units to satisfy in order to be included in the forward area if the objective is to minimized labor and equalize space or replenishment frequency respectively. Also lower bounds on the amount of an item to be included in the forward area are provided, if it is to be done so.

The second group of storage assignment problems arise considering the storage assignment policies. Four main policies can be considered. *Random storage* lies at one end of the extreme. In this case, every incoming pallet is stored in a location of the warehouse selected randomly from the available ones. The main advantage of the random storage method is the need for higher space utilization or lower space requirement. On the other hand, order-picking travel distance will increase as a result. Another disadvantage is the fact that order-pickers cannot get familiar with the places of the items within the warehouse, and therefore the need for a computerized environment will increase. At the other extreme is the *dedicated storage*, which requires storing of each item at a fixed location. Dedicated storage allows the order-pickers to become familiar with the item locations at the expense of lower space utilization or higher space requirement. *Class-based storage* combines the two methods mentioned. The items

are grouped in such a way that the fastest moving class contains a small percentage of the products, but contributes to a high percentage of the turnover. The classes are assigned fixed locations within the warehouse, but the storage assignment within each location is random. One last policy to be considered is the concept of *family grouping*, in which the similarities or relationships between the demands of the items are considered. The idea is to cluster the items that are likely to be picked together. A vast amount of literature on storage assignment policies is available, of which we summarize three.

In an earlier study, Malmborg and Krishnakumar [60] consider a storage assignment problem in a warehouse employing palletized AS/RS with multiple-command cycles (multiple storage and retrieval transactions can be performed by a picker in a cycle), with parallel-arranged racks. No correlation between the orders of the items, stochastic and stationary storage transactions, capacitated order-pickers and first come - first served servicing of the item orders are assumed. Three problems are mainly tackled: item to location assignment, item to order-picker assignment and picker routing. The total expected order-picking cost is calculated by modelling the order arrival process as a queue and the expected travel distance for a picker is found accordingly. The optimal storage assignment policy is determined depending on the two statements proposed by the authors. The first statement is that when the order pickers are equally efficient, equal work balancing minimizes the total idle time of the system. However, it does not guarantee maximum throughput by itself, as it may lead to excessive travel times for the order-pickers due to inefficient routing. The second statement is that the cube-per-order index (COI) assignment policy, which consists of assigning items to pickers depending on the ratio of the space occupied by the item to its turnover rate, yields a least cost solution for the total order-picking cost of the system. These two statements provide useful rules-of-thumb for storage assignment and picker routing.

Considering family grouping, Liu [58] discusses a clustering technique for a group of items in the slots of gravity-flow racks and to sequence the picking lists of customers. In this setting, the historical information on the previous orders is available, and based on this it is possible to determine a similarity measure for both the items and the customers. The similarity measure between two items is given as the ratio of their common orders to the maximal order quantity on the order where the two items are listed. The similarity measure between two customers is the ratio of their common item quantity to the maximal item quantity in which the customers have ordered in the same time interval. A mixed integer programming model to

maximize the total similarity value is given, and the proposed solution procedure is by using a primal-dual algorithm. The algorithm terminates when the required number of clusters is obtained. The algorithm is applied for a low-volume, multi-item distribution center and the results were compared with the existing stock location policy, showing a significant amount of improvement.

Petersen and Aase [67] take a more overall view, and consider the storage assignment problem along with order-batching and picker routing problems, with the following features: There are 10 picking aisles with front and back cross aisles, the aisles are wide enough to allow two-way travel, and narrow enough to allow picking from both sides, the demand for the items follows Pareto's principle, and picking is performed manually by a capacitated picker. Simulation is used to compare combinations of various batching, storage allocation and picker routing policies. Different policies are included in each of batching, storage assignment and picker routing; namely, strict order, FCFS and bin-pack for batching; random, class-based, and within-aisle volume-based for storage assignment; traversal, combined, and optimal for routing. Seven different order sizes are considered. It has been shown that changing all three of the existing policies (strict order batching, traversal routing and random storage) yields average savings between 27% and 29%. Sensitivity analyses for each of the factors are made in order to verify the results.

2.2.4 Zoning Problems

The problems we have discussed up to this point may either implicitly or explicitly assume that each picker may traverse any part of the warehouse during the order-picking activities. Especially in larger warehouses or when the items corresponding to an order are widely distributed throughout the warehouse, a need for assigning specific zones of the warehouse to the pickers might be reasonable. The assignment is simply called *zoning*. The advantages of zoning include reduced travel distances for the pickers, reduced traffic congestion and the pickers becoming familiar with the item locations in their zones. However, since the items of a single order are picked by multiple pickers, the orders must be consolidated before shipment to the customer.

There are several strategies for zoning. In the first one, called *progressive assembly* or *pick-and-pass*, an order-picker picks the items in the zone assigned to him and passes the pick list to

the next order-picker, who performs the same operations in his zone. The process continues until the last order-picker picks the items in his zone. The main advantage of this method is that order consolidation is made easier at the expense of higher order fulfillment time. The second strategy is *parallel* or *synchronized* picking, where each picker simultaneously picks the items on the pick list in his zone. In this way, orders are fulfilled in shorter time but more effort for order consolidation is needed as a consequence. A third strategy is the idea of *bucket brigades*, put forward by Bartholdi and Hackman [7]. In this case, no fixed boundaries between the zones exist. A picker travels back to the previous picker or to the starting point when he finishes picking the items assigned to him and restarts picking. This method minimizes the idle times of the pickers but requires more training.

Although the literature on zoning is limited, there has been a significant increase in the studies concerning zoning in the recent years. Petersen [66] considers a warehouse that uses zone picking, and tries to determine the optimal sizing of the zones, along with the relationships of zoning with storage policy and order size in terms of the route lengths. Manual order-picking is assumed in a warehouse where the aisles are wide enough to allow a picker to change direction within an aisle yet narrow enough to pick items from both sides and each picking route starts and ends at a depot. An experimental design is made with four factors: zone configuration (1, 2, 3 or 4 aisles), zone size (small, medium or large), storage policy (random, within-aisle or across-aisle), and pick list size (1, 5, 10, 20 or 30 items). Simulation is run for 50 replications of each factor combination, and the results are statistically analyzed. It is observed that the size of the picking zone, the storage policy and the order size affect the route length significantly. Large zones and small orders for zones of 3 and 4 aisles result in significantly less travel length than the others. Additionally, within-aisle storage policy dominates the other two.

Le-Duc and De Koster [57] provide a heuristic procedure to estimate the travel distance and determine the zones in a 2-block warehouse operating manual order-picking. The warehouse includes rectangular racks which can store more than one product type, items in the same class are assumed to have the same order frequency, and no order batching is applied. First, the average travel distance is estimated for a single aisle using a probabilistic model. The error of probabilistic model is tested using simulation, assuming class-based or random storage assignments. It is shown that the maximum error is around 2.5% (4.4%) for class-based assignment (random assignment) for the largest order sizes, showing that the travel time ap-

proximation is valid. For the case with multiple aisles, simulation reveals approximate errors of 7.3% and 7.9% for the two cases respectively. In order to determine the zoning, a mathematical model with linear constraints and a nonlinear objective function is formulated. Due to nonlinearity, the solution of the model needs considerable computational time. As a consequence, a heuristic procedure is developed for zoning. Computational experiments show that the procedure produces near-optimal results.

2.3 Operational Level Problems

The operational level problems are related to decisions that affect day-to-day transactions and do not require substantial amounts of investments. These problems take strategic and tactical level decisions as constraints and usually consider the assignment and control problems of people and equipment. Taking a similar approach to that of De Koster et al. [24], we classify the operational level problems under three main headings. The first of these are the order batching problems, the second group consists of picker routing problems, and the last group includes problems related to order accumulation and sorting.

2.3.1 Order Batching Problems

Single order picking, in which an order-picker picks items corresponding to a single order in a single trip, is especially useful when the sizes of the orders are large, or the time between two orders is large enough so that waiting for the orders to be accumulated increases the order fulfillment time substantially, or when the capacities of the order-pickers do not allow picking multiple orders in a single trip. When the sizes of the orders are small, and when the capacities of the order-pickers allow picking multiple orders in a single trip, *order batching* can be applied. There are two main methods for order batching. *Proximity batching*, in which assignment of orders to the batches are made based on the proximities of their storage locations, is the first of these. The main issue in this problem is to determine a proximity measure for the orders. The second method is *time window batching*, which forms batches from the orders that arrive in the same time interval, which is called a time window. The orders are then picked in the same trip in the following stages.

Gademann and Ven De Velde [32] discuss proximity batching in a parallel aisle warehouse

that contains no middle aisle. A single input/output station exists, and travel time is assumed to be linear in terms of the travel distance. There are n orders to be split into c batches, where n is a multiple of c. The objective is to minimize the total travel time. The problem is shown to be NP-complete in the strong sense by polynomial reduction from the partition into triangles problem, and is shown to be polynomially solvable when there are 2 batches. Due to the NP-completeness of the problem, a branch-and-price approximation algorithm is proposed. The five-step algorithm initializes by the formulation of an integer linear program which has exponential number of variables. A pricing algorithm follows the formulation procedure that is used to verify the global optimality of the solution to the LP relaxation of the model. The pricing algorithm uses a branch-and-bound procedure to find columns with minimum reduced costs. If the pricing algorithm no longer returns a column with negative reduced cost, the LP solution provides a lower bound on the optimal solution. A Lagrangian relaxation procedure improves the lower bound found by the LP relaxation. A column generation algorithm iteratively uses the first three steps to solve the linear relaxation and a branch-and-price algorithm uses the result by the column generation procedure to come up with a solution to the overall problem. The algorithm is tested on several generated problems in a warehouse where class-based storage is applied and observed to perform well both in terms of solution quality and computation time.

Won and Olafsson [85] consider the order batching and order-picking problems together. A warehouse is assumed where end-of-aisle order-picking is applied, the distance metric is Tchebysheff and orders that cannot be split into multiple batches. First, an integer programming formulation for the joint batching and picking problem is given. Ideas are borrowed from the formulations of the bin packing problem for order batching and the travelling salesman problem for the order-picking parts. Due to the NP-completeness of the problem, the required number of constraints for the proposed model is exponential. Therefore the authors propose two heuristic procedures. The first procedure solves the order batching and order-picking problems sequentially, that is, the solution to the batching problem is an input for the order-picking problem. In developing the heuristic, the orders are treated on a first-come-first-served basis, and the solution procedure is based on solving the traveling salesman problem using a 2-opt heuristic procedure. When computational experiments are performed, it is observed that the algorithm finds good quality solutions for small to medium sized problems, but deviation from optimal is high for the large-sized problems. Therefore a second heuris-

tic is developed solving the batching and picking problems jointly. The algorithm provides considerably more benefits than the sequential approach, as shown by computational results.

2.3.2 Picker Routing Problems

According to Tompkins et al. [79], around 55% of the warehouse operating expenses is caused by the order-picking operations. Furthermore, when the distribution of the time spent by an order picker is analyzed, it is observed that about 50% of the time is spent by travelling for picking the items. Therefore picker routing aims to aid the costliest problem in warehouse operations. The routing problem in a warehouse is a special case of the traveling salesman problem, but due to the fact that not all the nodes in the corresponding graph of a warehouse have to be visited, the problem takes the form of a Steiner traveling salesman problem. Since the detailed survey on the literature of this problem is provided in the next two chapters, we briefly summarize the related literature here.

For the warehouses containing no middle aisles, Ratliff and Rosenthal [71] propose an algorithm linear in terms of the number of aisles in the warehouse. They make use of seven *equivalence classes* and enumerate their combinations for each aisle starting with the leftmost one and obtain a dynamic programming-based algorithm. The enumerative procedure of this algorithm makes it harder to implement an use in practice. Various heuristic procedures have been defined to overcome this difficulty. Hall [45] summarizes the most widely used ones such as the S-shape heuristic, the largest gap heuristic and the combined heuristic. Makris and Giakoumakis [59] also propose a k-opt type heuristic procedure. De Koster and Van Der Poort [26] extend the algorithm of Ratliff and Rosenthal [71] to the case where the start and end nodes of the picker are different, i.e., an extension from the picker's route to the picker's path. They call this case *decentralized depositing*. The algorithm simply consists of assuming a dummy sink node that has zero distance to the corner points and solving the tour problem.

For the case with middle aisles, the literature mostly consists of heuristic procedures. Roodbergen and De Koster [72] extend the work of Ratliff and Rosenthal [71] to the case with a single middle aisle. In this case, the number of possible equivalence classes increases from 7 to 25. Vaughan and Petersen [82] propose an aisle-by-aisle heuristic, which is a DP-based approach that seeks the best entrance and exit points of the aisles. The proposed heuristic

can handle the case where the widths of the cross aisles are non-negligible. It allows the pick aisles to be visited only once, where each visit has to pick all the items in the aisle. However, this approach may result in solutions of high optimality gap, especially when almost all the items are located near the front and back cross aisles. Roodbergen and De Koster [73] summarize the heuristic procedures developed for the case with middle aisles such as S-shape, largest gap, aisle-by-aisle, and propose a DP-based combined heuristic, which combines the procedures of S-shape and largest gap heuristics.

2.3.3 Order Accumulation and Sorting Problems

When zoning and/or batching is applied, that is, the items corresponding to a single order are picked by multiple pickers and/or a single picker picks items corresponding to multiple orders, these orders need to be accumulated and sorted before shipment to the customers. In a typical accumulation/sorting (A/S) system, upon completion of the order-picking process, a picker places all the items (which may correspond to different orders) to a transport-conveyor located adjacent to the front or back cross aisle. The items then move on to a circulation conveyor of the sorter and enter the assigned shipping lane as soon as the items of the preceding order assigned to the same shipping lane have entered. Otherwise, the items keep on circulating around the conveyor. On entering the shipping lane, the order is loaded onto the waiting truck and the lane is made available for the next order assigned to it.

The literature on the A/S systems is limited. Meller [63] studies the A/S problem in a warehouse with an A/S system as described previously. It is furthermore assumed that the loading sequence of orders to the trucks is fixed, the items corresponding to an order can be loaded to a truck in any sequence, each item occupies one unit of space on the lane, and multiple orders from the same truck can be assigned to a lane. A mathematical model is developed for the problem, and the solution procedure makes use of the minimum cardinality formulation to determine the optimal number of lanes, providing a constraint for the lane assignment problem, which determines the assignment of lanes to the orders. The solution procedure is applied to ten small-sized and ten large-sized problems (the size of the problems is defined by the maximum number of items that can be loaded onto a truck) and optimal solutions are found for each. Although the large-sized problems differ significantly from the small-sized problems in terms of solution time, all of the problems can be solved to optimality within

reasonable amount of time.

Johnson [51] discusses the impact of sorting strategies on the performance of A/S systems in an A/S system described in this section. It is assumed that no blocking occurs between the shipping lanes and the circulation conveyor (in other words, either processing at the shipping lane is shorter than that of the sortation process or there is enough buffer space in the shipping lane so that blocking is avoided). Two families of sorting strategies are considered for comparison. The first family consists of *fixed priority rules*, where the priority rule does not change during sorting. *Smallest order first* and *Largest order first* are two examples of fixed priority rules. The second family consists of *next available rules*. Each time an order is sorted, the next item to pass the bar code scanner at the shipping lane determines the next order to be sorted. After coming up with the expected sorting times for each family of strategies, it is shown that the next available rule dominates the fixed priority rules. The robustness of the findings is shown using simulation.

2.4 Conclusion on Warehouse Design Problems

As can be observed throughout the chapter, the warehouse design problems discussed here are interdependent on each other. Figure 2.2 summarizes the relationships between each type of problem in each decision level. Each arrow indicates that the second problem takes the results of the first as given. In other words, when these problems are considered together in a warehouse environment, they are solved in sequence. As expected, most of the hierarchical relations occur from a higher decision making level to the lower. However, there are also cases where the problems on the same decision making level have such relations. For instance, layout design problems, being one of the tactical level problems, take the results of process flow design and systems selection problems and storage capacity planning problems, which are strategic level problems, as inputs. The layout design of the warehouse is an input to the storage assignment and zoning problems (tactical level) as well as order batching and picker routing problems (operational level).

When the literature on the warehouse design problems is observed in general, there are a number of remarks to make about the uniformity of the studies on the decision levels they aid and the approaches they take. First of all, despite the fact that there is vast amount of litera-

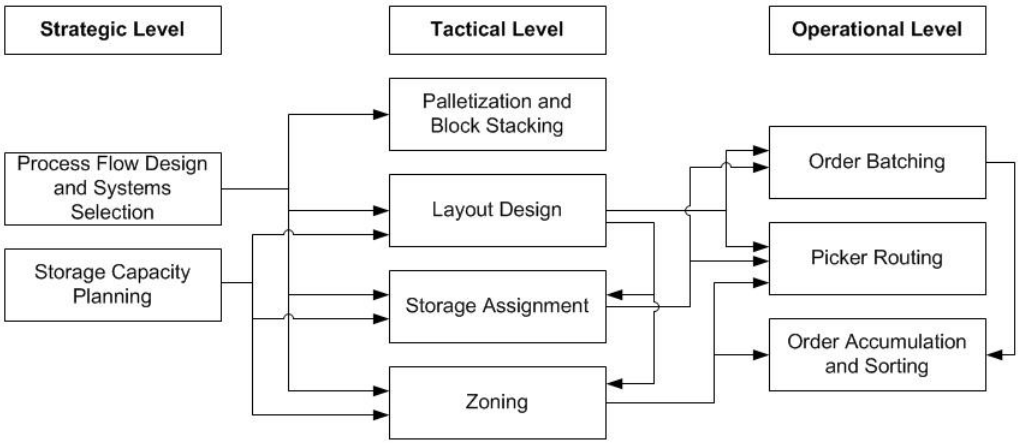

Figure 2.2: A summary of the warehouse design problems discussed in this chapter, arrows indicating hierarchical effects

ture on the tactical and operational level problems, strategic level problems have not received the amount of attention they deserve, as they provide inputs to the problems in the tactical and operational levels. This is due to the reason that most of the studies in the literature discuss problems that are well-defined in the sense that the performance measures are clearly expressed and well-quantified, stochasticity is limited, and the interactions with the environment are almost always ignored. This drawback prevents the results from being applicable in practice.

Another important issue that lacks the attention of the literature is the fact that the problems are also isolated from each other. When a number of problems are considered together, the solution approach is most of the time hierarchical. This causes suboptimalities in the overall problem. Even though tackling the problems in an integrated manner is a difficult challenge, ignoring this integrity widens the gap between the theoretical studies and the implementations in practice.

CHAPTER 3

REVIEW ON THE COMPLEXITY OF THE ORDER-PICKING PROBLEM

Having provided an overview of the warehouse design problems along with their connection to the order-picking problem (OPP) in Chapter 2, we can now define the specifics of the OPP. Note that throughout this thesis, the k-OPP refers to the OPP with k cross aisles. First, we give the basic definitions and notation related to the OPP in Section 3.1, which will also be used throughout this thesis. Then, we will discuss the complexity of the problem, which is "open". Even though a polynomial time algorithm exists for the 2-OPP due to Ratliff and Rosenthal [71], and for the 3-OPP due to Roodbergen and De Koster [72], the complexity of the OPP in terms of the number of cross aisles has not yet been determined. The complexity discussion is divided mainly into two headings. Section 3.2 gives the polynomially solvable special cases of the OPP and their algorithms. In Section 3.3, we discuss the complexity of the problem with more than one middle aisle, with reference to the findings on the complexity of related problems in the literature.

3.1 Basic Definitions and Notation

Whenever the OPP is mentioned, we are assuming a parallel-aisle warehouse, an example of which is shown in Figure 3.1, consisting of *pick aisles* that contain the items to be picked (shown as black squares in the figure). The warehouse also contains *cross aisles*, which are orthogonal to the pick aisles. It is assumed that cross aisles do not contain any pick items. The two cross aisles at the ends of the warehouse are named *front* and *back cross aisles*, whereas the remaining cross aisles are named *middle aisles*. In the existence of middle aisles,

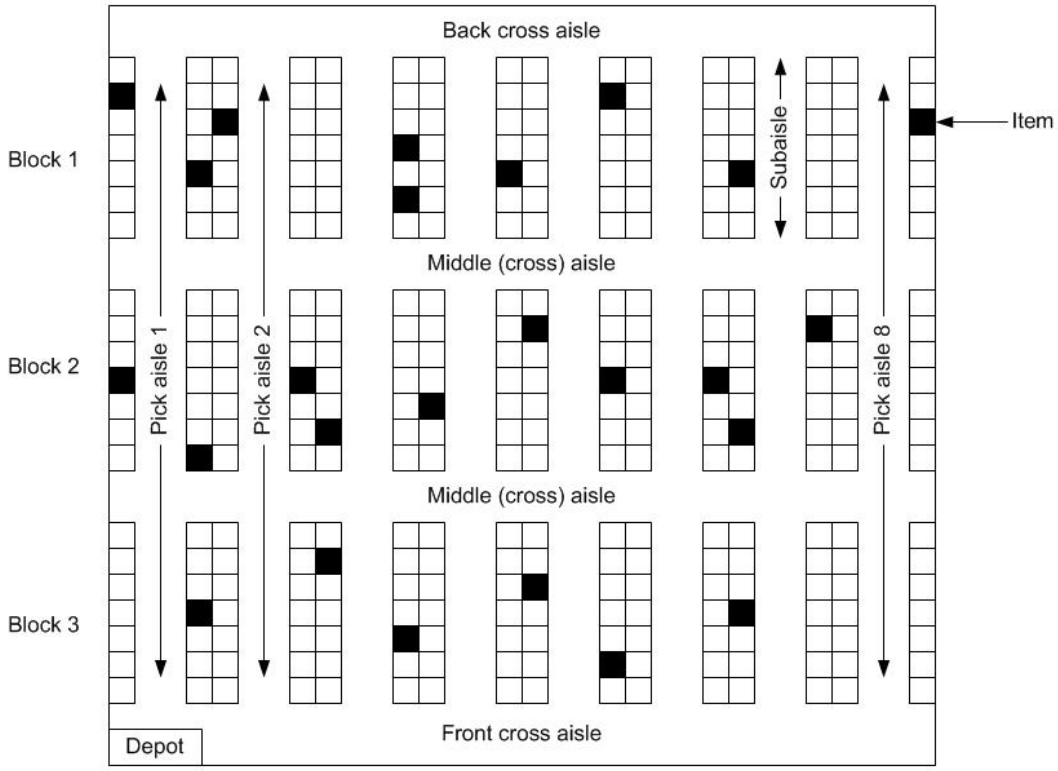

Figure 3.1: A parallel-aisle warehouse with 2 middle aisles and 25 pick items

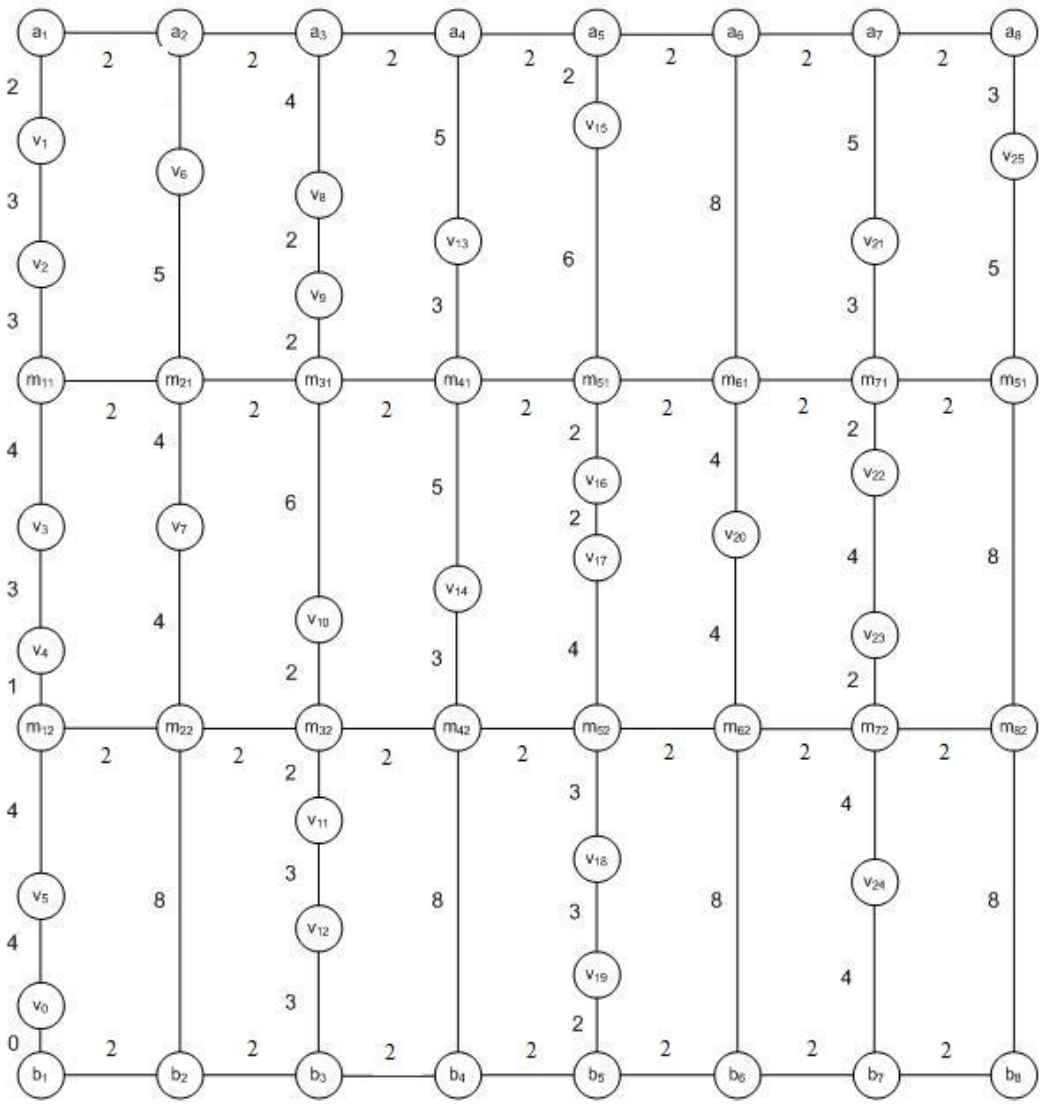

Figure 3.2: Graph representation of the warehouse in Figure 3.1

pick aisles are divided into *subaisles* and the warehouse is divided into *blocks*. In a feasible solution to the problem, the picker starts at the *depot*, picks all items and returns to the depot. Unless otherwise stated, the depot is assumed to be at the left corner of the front cross aisle. It is also assumed that the cross aisles have zero width.

We will denote the k-OPP as the order-picking problem in a warehouse with k cross aisles. As a special case, the 1-OPP refers to the case with only the front cross aisle. It is obvious that the solution to the 1-OPP is trivial; the optimal route starts at the depot, traverses all nonempty pick aisles beginning with the leftmost one, picks all items and returns to the depot. Therefore throughout the rest of the report it is assumed that $k \geq 2$.

A graph $G = (V, E)$ consists of a *vertex* or *node* set V and an *edge* set E. An edge is denoted by its two end nodes u and v as (uv). Each edge $e \in E$ has a *distance* or *cost* w_e associated with it. The k-OPP can be defined as a graph problem as follows: Each item i is denoted as vertex v_i, where v_0 denotes the depot. Corners of the back cross aisle with pick aisle i are denoted by vertex a_i, whereas corners of the front cross aisle with pick aisle i are denoted by vertex b_i. Corners of pick aisle i with middle aisle j (with $j = 1$ being the closest to the back cross aisle and so on) are denoted by vertex m_{ij} The edges represent the accessibility between items, from a corner to an item and vice versa. Figure 3.2 shows the graph representation of the warehouse shown in Figure 3.1.

Given a node subset $W \subseteq V$, $G(W)$ denotes the subgraph of G induced by W and $E(W)$ denotes its edges. The *degree* of a vertex is the number of edges incident to it.

A *path* between two nodes u and v consists of a series of consecutive edges $e_1, e_2, \ldots, e_n$ with e_1 having u as a start node, e_n having v as an end node, and there is no repetition of edges. Two nodes u and v are *connected* if there exists a path between them. A *cycle* is a path with $u = v$. A *tour* $\langle v_1, v_2, \ldots, v_n \rangle$ is a cycle that contains all the nodes of G. A *tree* is a connected subgraph of G that does not contain a cycle.

A graph *G=(V, E)* is said to be *Eulerian* if there exists a path that goes through all edges of the graph. A *Hamiltonian* graph, on the other hand, has a cycle spanning all of its nodes. The *Chinese Postman Problem* (CPP) consists of determining the least cost (or distance) traversal of a Eulerian graph. On the other hand, the *Rural Postman Problem* (RPP) aims to find a least cost (or distance) traversal of a subset R of the edges of a Eulerian graph.

A *grid graph* is a finite, vertex-induced subgraph of the *infinite grid*, which is the infinite graph with set of vertices $Z \times Z$ and set of edges $\{\{(xy), (x'y')\} : |x - x'| + |y - y'| = 1\}$. A *solid grid graph*, on the other hand, is a grid graph with no "holes" in it, that is, all bounded faces of the graph have area one. Figure 3.3 illustrates a general grid graph and a solid grid graph. A *rectangular solid grid graph* is simply a solid grid graph with equal number of grids for each row and each column.

Two edges e_1 and e_2 are *in series* if they are incident to a vertex of degree 2 and are *parallel* if they join the same pair of distinct vertices. A *series-parallel* (S-P) graph is recursively defined as follows:

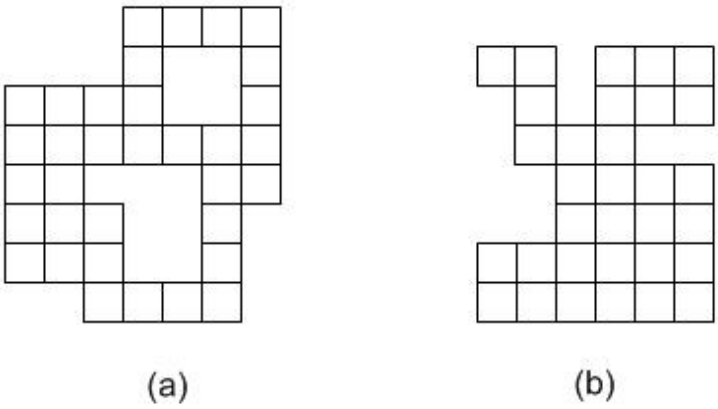

Figure 3.3: A general grid graph (a) and a solid grid graph (b)

- A graph that simply consists of vertices v_1 and v_2 joined by two parallel edges e_1 and e_2, each of which has end nodes as v_1 and v_2 is series-parallel.

- If G is a series-parallel graph, then a graph obtained from G by replacing any edge of G by series or parallel edges is series-parallel.

Figure 3.4 illustrates the second part of the definition. If G containing *(uv)* is an S-P graph, then by adding a series of edges *(uu'),(u'v)* after deleting *(uv)* or adding a duplicate of edge *(uv)* creates another S-P graph G'.

3.2 Polynomially Solvable Cases of the OPP and Heuristic Approaches

In this section, we discuss the polynomially solvable cases of the order-picking problem, and provide the polynomial-time algorithms and heuristic procedures. First, we discuss the case of 1-OPP, in which only the front cross aisle exists. Then, we review the 2-OPP, which is the smallest non-trivial case of the OPP. We give the polynomial time algorithm by Ratliff and Rosenthal [71], as well as the heuristic procedures. Following this, we provide the polynomial algorithm for 3-OPP, which is due to Roodbergen and De Koster [72]. Lastly, we discuss the heuristic procedures for the case where middle aisles exist (k-OPP with $k \geq 3$).

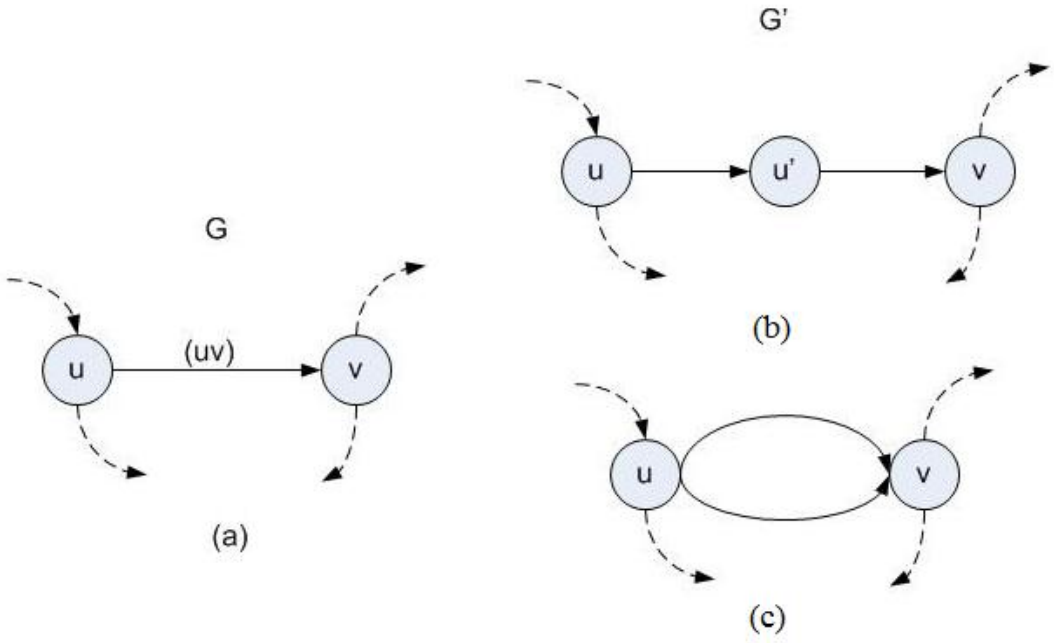

Figure 3.4: Recursive definition of an S-P graph: If G in (a) is S-P, then by deleting (uv) and adding either series (b) or parallel edges (c), we obtain G' as an S-P graph

3.2.1 The 1-OPP

The 1-OPP is the simplest of the order-picking problems. As discussed before, it consists of the front cross aisle and the pick aisles connected to it. The solution to the 1-OPP is trivial. The optimal route starts at the depot, which is on the intersection of the cross aisle and the left-most pick aisle, then proceeds by collecting the items starting with the left-most non-empty aisle. When the picker picks the last item in the right-mos aisle, he returns to the depot using the front cross aisle.

3.2.2 The 2-OPP

The 2-OPP is the basic case of non-trivial order-picking problems. Ratliff and Rosenthal [71] provide an efficient solution algorithm for the problem, solvable in linear time in terms of the number of aisles.

The algorithm for the 2-OPP is a DP-based algorithm, which provides an aisle-by-aisle solution procedure [71]. Here, the aisles constitute the stages and the states are the distances

needed to traverse all the aisles to the left of, and including that stage. The following definitions are due to [71]:

Definition 3.1 *A partial tour subgraph is a subgraph that spans a subset of the nodes of the graph.*

Definition 3.2 *Let the aisles be numbered in increasing order starting from the left-most one up to the right-most one ($j = 1$ for the left-most one, $j = 2$ for the one on its right, etc). A partial tour subgraph L_j^- consists of the vertices a_j and b_j as well as all the vertices in aisles up to (but not including) aisle j. Partial tour subgraph L_j^+, on the other hand, consists of all the vertices in aisles up to and including aisle j.*

Definition 3.3 *The equivalence class of a partial tour subgraph is defined by three parameters: the degrees of a_i and b_i and the number of components in the partial tour subgraph. In the case of the 2-OPP, the first two parameters can be E (even), U (uneven) or 0 (zero), and the third can be 0C (no component), 1C (a single connected component), or 2C (two connected components).*

Ratliff and Rosenthal [71] show that there are only seven equivalence classes for any partial tour subgraph, namely (U,U,1C), (0,E,1C), (E,0,1C), (E,E,1C), (E,E,2C), (0,0,0C) and (0,0,1C), six possible connection types as shown in Figure 3.5, and five different ways of connecting the aisles as shown in Figure 3.6.

The algorithm starts by forming the L_1^+ configurations for each of the seven equivalence classes, using the six possible connection types described in Figure 3.5. Then, it goes on to form the L_2^- configurations by adding the between-aisle connections shown in Figure 3.6. Table 3.1 shows the resulting equivalence classes when the between-aisle connections in Figure 3.6 are added to each equivalence class.

It can be observed from Table 3.1 that a number of solutions are obtained for each equivalence class. For the next step, each equivalence class will have an associated unique solution, which is the shortest of the obtained solutions in that equivalence class. Having formed the L_2^- solutions, the next step is to find the L_2^+ solutions, which will be obtained by adding the connection types shown in Figure 3.5 to each equivalence class. Table 3.2 shows the result-

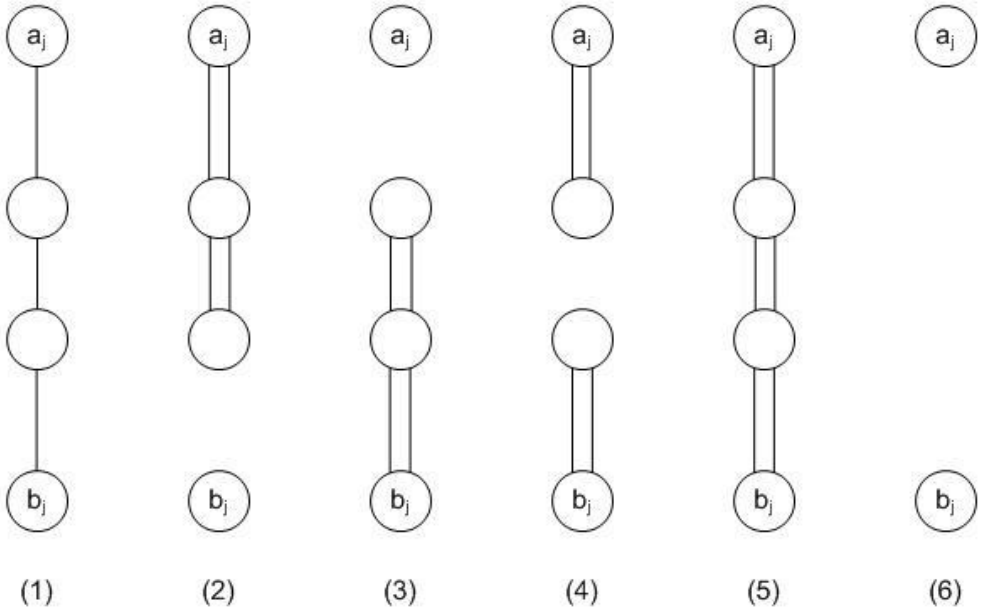

Figure 3.5: The six possible connection types within the aisles

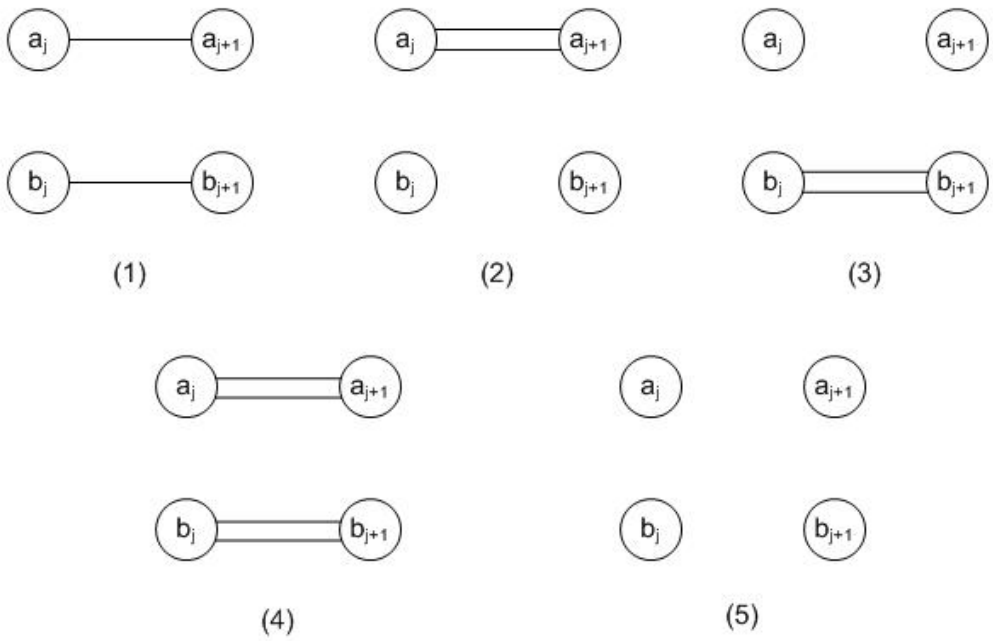

Figure 3.6: The five possible connection types between the aisles

Table 3.1: Resulting equivalence classes after adding the connection types in Figure 3.6 to each L_j^+ equivalence class, dashed lines indicating suboptimality or infeasibility

L_j^+ equivalence classes	Connection types in Figure 3.6				
	(1)	(2)	(3)	(4)	(5)
(U,U,1C)	(U,U,1C)	-	-	-	-
(E,0,1C)	-	(E,0,1C)	-	(E,E,2C)	(0,0,1C)
(0,E,1C)	-	-	(0,E,1C)	(E,E,2C)	(0,0,1C)
(E,E,1C)	-	(E,0,1C)	(0,E,1C)	(E,E,1C)	(0,0,1C)
(E,E,2C)	-	-	-	(E,E,2C)	-
(0,0,0C)	-	-	-	-	(0,0,0C)
(0,0,1C)	-	-	-	-	(0,0,1C)

Table 3.2: Resulting equivalence classes after adding the connection types in Figure 3.5 to each L_j^- equivalence class, dashed lines indicating suboptimality

L_j^- equivalence classes	Connection types in Figure 3.5					
	(1)	(2)	(3)	(4)	(5)	(6)
(U,U,1C)	(E,E,1C)	(U,U,1C)	(U,U,1C)	(U,U,1C)	(U,U,1C)	(U,U,1C)
(E,0,1C)	(U,U,1C)	(E,0,1C)	(E,E,2C)	(E,E,2C)	(E,E,1C)	(E,0,1C)
(0,E,1C)	(U,U,1C)	(E,E,2C)	(0,E,1C)	(E,E,2C)	(E,E,1C)	(0,E,1C)
(E,E,1C)	(U,U,1C)	(E,E,1C)	(E,E,1C)	(E,E,1C)	(E,E,1C)	(E,E,1C)
(E,E,2C)	(U,U,1C)	(E,E,2C)	(E,E,2C)	(E,E,2C)	(E,E,1C)	(E,E,2C)
(0,0,0C)	(U,U,1C)	(E,0,1C)	(0,E,1C)	(E,E,2C)	(E,E,1C)	(0,0,0C)
(0,0,1C)	-	-	-	-	-	(0,0,1C)

ing equivalence classes when the within-aisle connections in Figure 3.5 are added to each equivalence class.

As in the previous case, each equivalence class in L_{j+1}^- will be the shortest of the solutions obtained in that class.

After finding the solutions for each equivalence class for L_2^+, the algorithm then goes on to find out each equivalence class solution for L_3^-, L_3^+ and so on. The optimal solution is the shortest of the solutions associated with the equivalence classes (0,E,1C), (E,0,1C), (E,E,1C) and (0,0,1C) for L_n^+. Figure 3.7 summarizes Procedure *Ratliff_Rosenthal*.

As an example, consider a 2-OPP with 8 items and 6 aisles illustrated in Figure 3.8. Table 3.3 summarizes the solution procedure for this example problem, whose optimal solution is 62, and the optimal route is depicted in Figure 3.9.

The enumerative procedure of the algorithm by Ratliff and Rosenthal [71] makes it harder to

```
begin

\\initialize
-   i = 1; \\equivalence classes
-   j = 1; \\aisles
-   l = 1; \\connection types in Figure 3.5;
-   m = 1; \\connection types in Figure 3.6;
-   z(L_1^-(i)) = 0 for all i;
-   z(L_j^-(i)) = ∞ for all 2 ≤ j ≤ n;
-   z(L_j^+(i)) = ∞ for all j;

while j ≤ n do
        while i ≤ 7 do
                while l ≤ 6 do
-                       Apply connection type l to equivalence class i. Let the equivalence class of the
                        solution be p and its length be z;
                        if z ≤ z(L_j^+(p))
-                               z(L_j^+(p)) = z;
                        endif
-               l ← l+1;
                endwhile
-       i ← i+1;
        endwhile

-       i = l = 1;
        while i ≤ 7 do
                while m ≤ 5 do
-                       Apply connection type m to equivalence class i. Let the equivalence class of the
                        solution be p and its length be z;
                        if z ≤ z(L_{j+1}^-(p))
-                               z(L_{j+1}^-(p)) = z;
                        endif
-               m ← m+1;
                endwhile
-       i ← i+1;
        endwhile

-       i = m = 1;
-   j ← j+1;

endwhile

-   Optimal solution: min {z(L_n^+(E, E,1C)), z(L_n^+(E,0,1C)), z(L_n^+(0, E,1C)), z(L_n^+(0,0,1C))};

end
```

Figure 3.7: Procedure *Ratliff_Rosenthal*

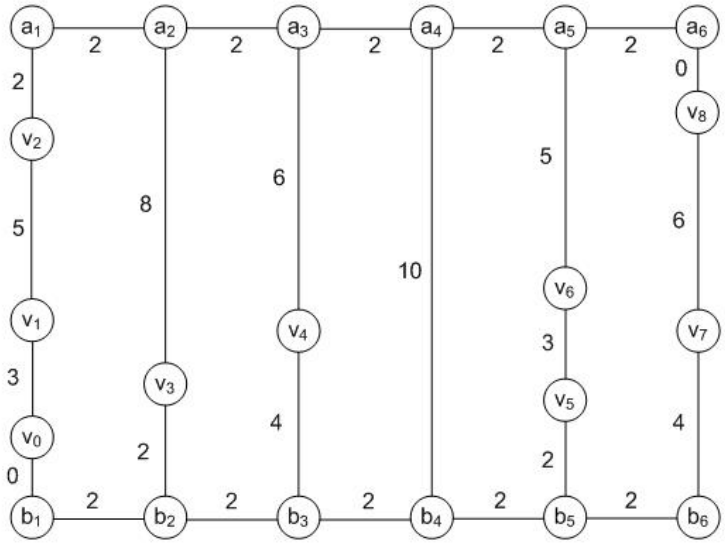

Figure 3.8: An example 2-OPP with 6 aisles and 8 items, where v_0 corresponds to the depot

Table 3.3: Solution to the example problem in Figure 3.8, bold entries indicating part of the optimal solution, dashed lines indicating infeasiblity or suboptimality

	L_1^-	L_1^+	L_2^-	L_2^+	L_3^-	L_3^+
(1. U,U,1C)	—	**10, -, i**	**14, 1, i**	**18, 1, iii**	**22, 1, i**	**30, 1, iii**
(2. E,0,1C)	—	20, -, ii	24, 2, ii	40, 2, ii	28, 4, ii	40, 2, ii
(3. 0,E,1C)	—	16, -, iii	20, 3, iii	24, 3, iii	28, 3, iii	36, 3, iii
(4. E,E,1C)	—	20, -, v	28, 4, iv	24, 1, i	32, 4, iv	32, 1, i
(5. E,E,2C)	—	10, -, iv	18, 5, iv	22, 5, iii	30, 5, iv	36, 2, iii
(6. 0,0,0C)	—	—	—	—	—	—
(7. 0,0,1C)	—	—	16, 3, v	—	24, 3, v	—

	L_4^-	L_4^+	L_5^-	L_5^+	L_6^-	L_6^+
(1. U,U,1C)	**34, 1, i**	**34, 1, vi**	**38, 1, i**	**48, 1, iii**	**52, 1, i**	60, 1, iv
(2. E,0,1C)	36, 4, ii	36, 2, vi	40, 2, ii	56, 2, ii	52, 4, ii	64, 2, ii
(3. 0,E,1C)	36, 4, iii	36, 3, vi	40, 3, iii	50, 3, iii	52, 4, iii	72, 3, iii
(4. E,E,1C)	40, 4, iv	40, 4, vi	48, 4, iv	48, 1, i	56, 4, iv	**62, 1, i**
(5. E,E,2C)	44, 3, iv	44, 5, vi	44, 2, iv	50, 2, iii	58, 3, iv	60, 2, iv
(6. 0,0,0C)	—	—	—	—	—	—
(7. 0,0,1C)	32, 4, v	32, 7, vi	32, 7, v	—	48, 4, v	—

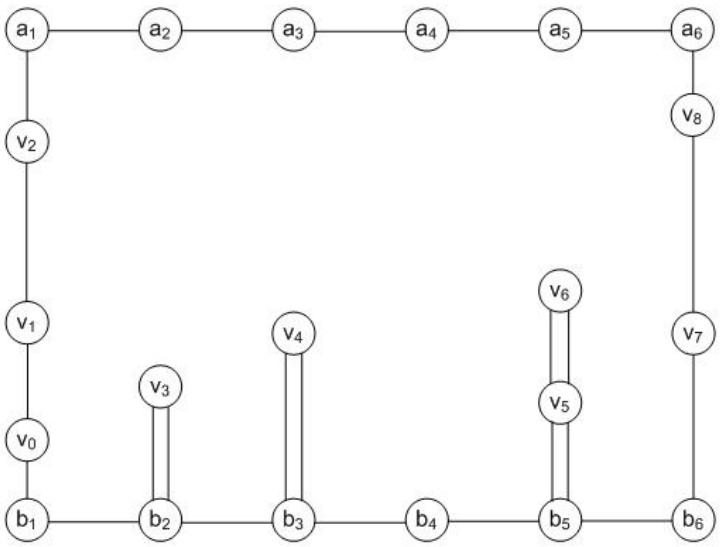

Figure 3.9: The optimal route for the problem given in Figure 3.8

implement and use in practice. Various heuristic procedures have been defined to overcome this difficulty. Hall [45] summarizes the most widely used ones such as the S-shaped heuristic, the largest gap heuristic and the combined heuristic. Makris and Giakoumakis [59] also propose a k-opt type heuristic procedure. Nevertheless, the study by Rosenthal and Ratliff [71] proves that the 2-OPP is solvable in linear time, i.e. in $O(n)$, where n is the number of aisles. De Koster and Van Der Poort [26] extend the algorithm of [71] to the case where the start and end nodes of the picker are different, i.e., an extension from the picker's route to the picker's path. They call this case *decentralized depositing*. The algorithm simply consists of assuming a dummy sink node that has zero distance to the corner points and solving the tour problem.

A nice property of the 2-OPP is that its graphical representation is an S-P graph (see Figure 3.10). The S-P graph formation of the 2-OPP with n aisles is as follows:

1. Start with two vertices connected by two parallel edges.
2. Replace one of the two edges by a series of two edges, creating two more nodes, ending up with an empty warehouse with 2 aisles.
3. Duplicate the edge corresponding to the right-most pick-aisle. Replace the duplicated edge by a series of two edges, creating two more nodes. Repeat this step *n-2* times to

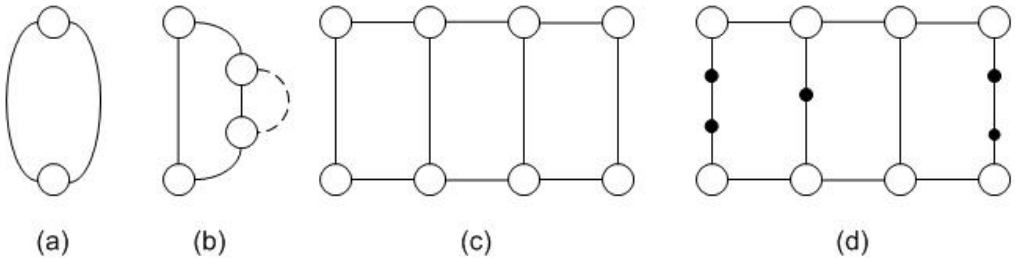

Figure 3.10: Formation of an S-P graph for a 2-OPP with 4 aisles and 5 items: (a) The initial graph, (b) Formation of the second aisle in Step 2 and the duplicated edge in step 3, (c) End of Step 3, (d) Addition of items in Step 4

obtain an empty warehouse with n aisles.

4. Add each item by replacing edges corresponding to aisles by a series of two edges. Repeat until all items are placed.

The property of the 2-OPP that it is actually an S-P graph has a nice implication on its tractability. We will see later that the more general Graphical Steiner Traveling Salesman Problem (GStSP) is polynomially solvable in S-P graphs.

3.2.3 The 3-OPP

The main difficulty in tackling the k-OPP with $k \geq 3$ is that it becomes exponentially hard to implement an extension of the algorithm by Ratliff and Rosenthal [71] as the number of cross aisles increases. Roodbergen and De Koster [72] have provided such an extension for a single middle aisle ($k = 3$). The procedure splits the problem into its two blocks. For each aisle, it first enumerates the possible solutions for the lower block, and combines these solutions with the upper solutions after enumeration. The following additional definitions are due to [72]. All other definitions are identical with those in Section 3.2.2.

Definition 3.4 *Let j denote the aisle index. If Y_j is the subgraph consisting of vertices m_{j1}, b_j and the vertices between them, then $L_j^{+y} = L_j \cup Y_j$. Similarly, if X_j is the subgraph consisting of the vertices a_j, m_{j1} and the vertices between them, then $L_j^{+x} = L_j^{+y} \cup X_j$.*

Definition 3.5 *The equivalence class of a partial tour subgraph for the 3-OPP is defined by*

a quintuplet: the degrees of a_i, m_{j1} and b_i, and the number of components in the partial tour subgraph and which two vertices are in the same component (only if there are two components). The first three elements can be E (even), U (uneven) or 0 (zero), the fourth can be 0C (no component), 1C (a single connected component), 2C (two connected components), or 3C (three connected components) and the last can be a-mc, m-ac or c-am only if the fourth is 2C. Otherwise it is empty.

It is shown by Roodbergen and De Koster [72] that the 25 possible equivalence classes are (0,0,0,0C), (0,0,0,1C), (E,E,E,1C), (E,E,E,3C), (E,0,0,1C), (0,E,0,1C), (0,0,E,1C), (E,E,0,1C), (E,0,E,1C), (0,E,E,1C), (U,U,0,1C), (U,0,U,1C), (0,U,U,1C), (E,U,U,1C), (U,E,U,1C), (U,U,E,1C), (E,E,0,2C), (E,0,E,2C), (0,E,E,2C), (E,U,U,2C), (U,E,U,2C), (U,U,E,2C), (E,E,E,2C,a-mc), (E,E,E,2C,m-ac), and (E,E,E,2C,c-am). Six different connection types within each subaisle are as in the case of 2-OPP, but the number of different ways to connect the aisles increases from 5 to 14. These are shown in Figure 3.11.

After initializing all the L_1^- solutions to zero, the algorithm proceeds by enumerating the L_1^{+y} solutions using the within-aisle connections in Figure 3.5. Then, it enumerates the L_1^{+x} solutions based on the ones in L_1^{+y}, again using the within-aisle connections in Figure 3.5. Following these, L_2^- solutions are enumerated using the between-aisle connections in Figure 3.11. Then L_2^{+y} solutions are enumerated, and so on. The procedure continues until L_n^{+x} solutions are obtained, and the optimal solution is the one having the shortest distance among the equivalence classes (0,0,0,1C), (E,0,0,1C), (0,E,0,1C), (0,0,E,1C), (E,E,0,1C), (E,0,E,1C), (0,E,E,1C), and (E,E,E,1C). Tables 3.4 and 3.5 show the resulting equivalence classes when the within-aisle connections in Figure 3.5 are added to each equivalence class. Table 3.6 shows the resulting equivalence classes when the between-aisle connections in Figure 3.11 are added to each equivalence class.

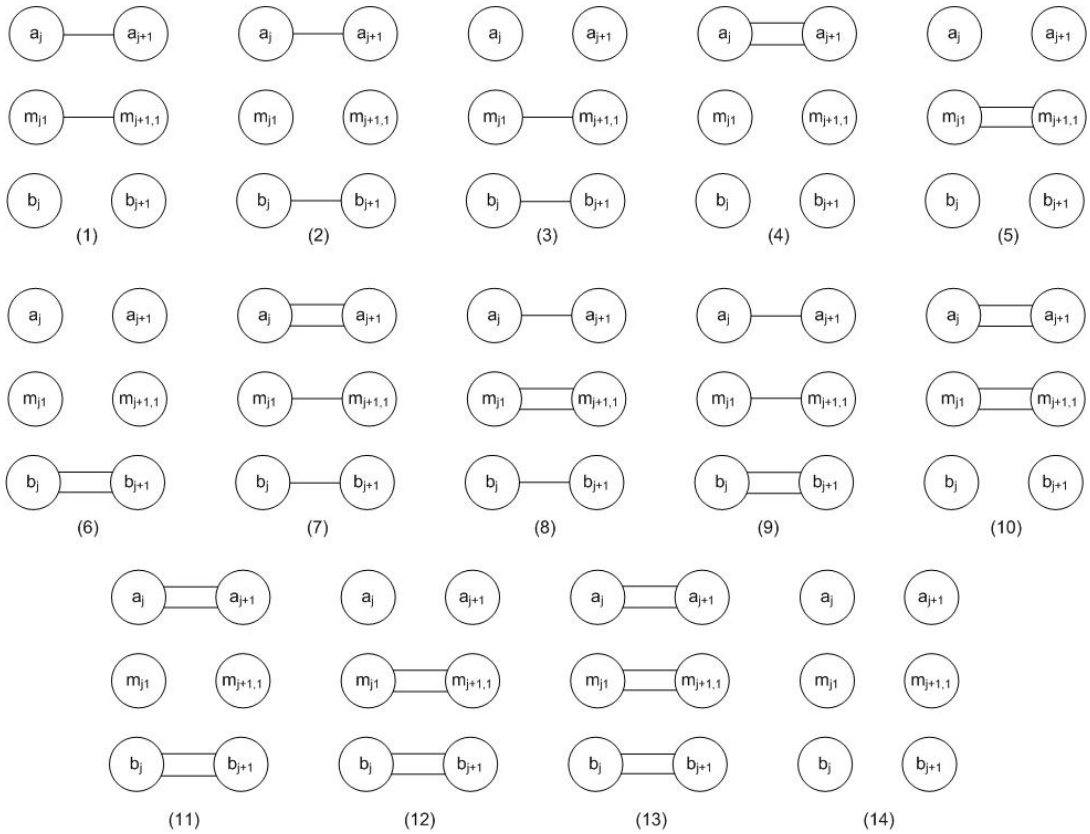

Figure 3.11: The fourteen possible connection types between the aisles

Table 3.4: Resulting equivalence classes L_j^{+y} after adding the connection types in Figure 3.5 to each L_j^{-} equivalence class, dashed lines indicating suboptimality

L_j^{-} equivalence classes	Connection types in Figure 3.5					
	(1)	(2)	(3)	(4)	(5)	(6)
(0,0,0,0C)	(0,U,U,1C)	(0,E,0,1C)	(0,0,E,1C)	(0,E,E,2C)	(0,E,E,1C)	(0,0,0,0C)
(0,0,0,1C)	—	—	—	—	—	(0,0,0,1C)
(E,0,0,1C)	(E,U,U,2C)	(E,E,0,2C)	(E,0,E,2C)	(E,E,E,3C)	(E,E,E,2C,a-mc)	(E,0,0,1C)
(0,E,0,1C)	(0,U,U,1C)	(0,E,0,1C)	(0,E,E,2C)	(0,E,E,2C)	(0,E,E,1C)	(0,E,0,1C)
(0,0,E,1C)	(0,U,U,1C)	(0,E,E,2C)	(0,0,E,1C)	(0,E,E,2C)	(0,E,E,1C)	(0,0,E,1C)
(E,E,0,1C)	(E,U,U,1C)	(E,E,0,1C)	(E,E,E,2C,c-am)	(E,E,E,2C,c-am)	(E,E,E,1C)	(E,E,0,1C)
(E,0,E,1C)	(E,U,U,1C)	(E,E,E,2C,m-ac)	(E,0,E,1C)	(E,E,E,2C,m-ac)	(E,E,E,1C)	(E,0,E,1C)
(0,E,E,1C)	(0,U,U,1C)	(0,E,E,1C)	(0,E,E,1C)	(0,E,E,1C)	—	(0,E,E,1C)
(E,E,E,1C)	(E,U,U,1C)	(E,E,E,1C)	(E,E,E,1C)	(E,E,E,1C)	—	(E,E,E,1C)
(U,U,0,1C)	(U,E,U,1C)	(U,U,0,1C)	(U,U,E,2C)	(U,U,E,2C)	(U,U,E,1C)	(U,U,0,1C)
(U,0,U,1C)	(U,U,E,1C)	(U,E,U,2C)	(U,0,U,1C)	(U,E,U,2C)	(U,E,U,1C)	(U,0,U,1C)
(0,U,U,1C)	(0,E,E,1C)	(0,U,U,1C)	(0,U,U,1C)	(0,U,U,1C)	—	(0,U,U,1C)
(E,U,U,1C)	(E,E,E,1C)	(E,U,U,1C)	(E,U,U,1C)	(E,U,U,1C)	—	(E,U,U,1C)
(U,E,U,1C)	(U,U,E,1C)	(U,E,U,1C)	(U,E,U,1C)	(U,E,U,1C)	—	(U,E,U,1C)
(U,U,E,1C)	(U,E,U,1C)	(U,U,E,1C)	(U,U,E,1C)	(U,U,E,1C)	—	(U,U,E,1C)
(E,E,0,2C)	(E,U,U,2C)	(E,E,0,2C)	(E,E,E,3C)	(E,E,E,3C)	(E,E,E,2C,a-mc)	(E,E,0,2C)
(E,0,E,2C)	(E,U,U,2C)	(E,E,E,3C)	(E,0,E,2C)	(E,E,E,3C)	(E,E,E,2C,a-mc)	(E,0,E,2C)
(0,E,E,2C)	(0,U,U,1C)	(0,E,E,2C)	(0,E,E,2C)	(0,E,E,2C)	(0,E,E,1C)	(0,E,E,2C)
(E,E,E,2C,a-mc)	(E,U,U,2C)	(E,E,E,2C,a-mc)	(E,E,E,2C,a-mc)	(E,E,E,2C,a-mc)	—	(E,E,E,2C,a-mc)
(E,E,E,2C,m-ac)	(E,U,U,1C)	(E,E,E,2C,m-ac)	(E,E,E,2C,m-ac)	(E,E,E,2C,m-ac)	(E,E,E,1C)	(E,E,E,2C,m-ac)
(E,E,E,2C,c-am)	(E,U,U,1C)	(E,E,E,2C,c-am)	(E,E,E,2C,c-am)	(E,E,E,2C,c-am)	(E,E,E,1C)	(E,E,E,2C,c-am)
(E,U,U,2C)	(E,E,E,1C,a-mc)	(E,U,U,2C)	(E,U,U,2C)	(E,U,U,2C)	—	(E,U,U,2C)
(U,E,U,2C)	(U,U,E,1C)	(U,E,U,2C)	(U,E,U,2C)	(U,E,U,2C)	(U,E,U,1C)	(U,E,U,2C)
(U,U,E,2C)	(U,E,U,1C)	(U,U,E,2C)	(U,U,E,2C)	(U,U,E,2C)	(U,U,E,1C)	(U,U,E,2C)
(E,E,E,3C)	(E,U,U,2C)	(E,E,E,3C)	(E,E,E,3C)	(E,E,E,3C)	(E,E,E,2C,a-mc)	(E,E,E,3C)

Table 3.5: Resulting equivalence classes L_j^{+x} after adding the connection types in Figure 3.5 to each L_j^{+y} equivalence class, dashed lines indicating suboptimality

L_j^- equivalence classes	Connection types in Figure 3.5					
	(1)	(2)	(3)	(4)	(5)	(6)
(0,0,0,0C)	(U,U,0,1C)	(E,0,0,1C)	(0,E,0,1C)	(E,E,0,2C)	(E,E,0,1C)	(0,0,0,0C)
(0,0,0,1C)	—	—	—	—	—	(0,0,0,1C)
(E,0,0,1C)	(U,U,0,1C)	(E,0,0,1C)	(E,E,0,2C)	(E,E,0,2C)	(E,E,0,1C)	(E,0,0,1C)
(0,E,0,1C)	(U,U,0,1C)	(E,E,0,2C)	(0,E,0,1C)	(E,E,0,2C)	(E,E,0,1C)	(0,E,0,1C)
(0,0,E,1C)	(U,U,E,2C)	(E,0,E,2C)	(0,E,E,2C)	(E,E,E,3C)	(E,E,E,2C,c-am)	(0,0,E,1C)
(E,E,0,1C)	(U,U,0,1C)	(E,E,0,1C)	(E,E,0,1C)	(E,E,0,1C)	—	(E,E,0,1C)
(E,0,E,1C)	(U,U,E,1C)	(E,0,E,1C	(E,E,E,2C,m-ac)	(E,E,E,2C,m-ac)	(E,E,E,1C)	(E,0,E,1C)
(0,E,E,1C)	(U,U,E,1C)	(E,E,E,2C,a-mc)	(0,E,E,1C)	(E,E,E,2C,a-mc)	(E,E,E,1C)	(0,E,E,1C)
(E,E,E,1C)	(U,U,E,1C)	(E,E,E,1C)	(E,E,E,1C)	(E,E,E,1C)	—	(E,E,E,1C)
(U,U,0,1C)	(E,E,0,1C)	(U,U,0,1C)	(U,U,0,1C)	(U,U,0,1C)	—	(U,U,0,1C)
(U,0,U,1C)	(E,U,U,1C)	(U,0,U,1C)	(U,E,U,2C)	(U,E,U,2C)	(U,E,U,1C)	(U,0,U,1C)
(0,U,U,1C)	(U,E,U,1C)	(E,U,U,2C)	(0,U,U,1C)	(E,U,U,2C)	(E,U,U,1C)	(0,U,U,1C)
(E,U,U,1C)	(U,E,U,1C)	(E,U,U,1C)	(E,U,U,1C)	(E,U,U,1C)	—	(E,U,U,1C)
(U,E,U,1C)	(E,U,U,1C)	(U,E,U,1C)	(U,E,U,1C)	(U,E,U,1C)	—	(U,E,U,1C)
(U,U,E,1C)	(E,E,E,1C)	(U,U,E,1C)	(U,U,E,1C)	(U,U,E,1C)	—	(U,U,E,1C)
(E,E,0,2C)	(U,U,0,1C)	(E,E,0,2C)	(E,E,0,2C)	(E,E,0,2C)	(E,E,0,1C)	(E,E,0,2C)
(E,0,E,2C)	(U,U,E,2C)	(E,0,E,2C)	(E,E,E,3C)	(E,E,E,3C)	(E,E,E,2C,c-am)	(E,0,E,2C)
(0,E,E,2C)	(U,U,E,2C)	(E,E,E,3C)	(0,E,E,2C)	(E,E,E,3C)	(E,E,E,2C,c-am)	(0,E,E,2C)
(E,E,E,2C,a-mc)	(U,U,E,1C)	(E,E,E,2C,a-mc)	(E,E,E,2C,a-mc)	(E,E,E,2C,a-mc)	(E,E,E,1C)	(E,E,E,2C,a-mc)
(E,E,E,2C,m-ac)	(U,U,E,1C)	(E,E,E,2C,m-ac)	(E,E,E,2C,m-ac)	(E,E,E,2C,m-ac)	(E,E,E,1C)	(E,E,E,2C,m-ac)
(E,E,E,2C,c-am)	(U,U,E,2C)	(E,E,E,2C,c-am)	(E,E,E,2C,c-am)	(E,E,E,2C,c-am)	—	(E,E,E,2C,c-am)
(E,U,U,2C)	(U,E,U,1C)	(E,U,U,2C)	(E,U,U,2C)	(E,U,U,2C)	(E,U,U,1C)	(E,U,U,2C)
(U,E,U,2C)	(E,U,U,1C)	(U,E,U,2C)	(U,E,U,2C)	(U,E,U,2C)	(U,E,U,1C)	(U,E,U,2C)
(U,U,E,2C)	(E,E,E,2C,c-am)	(U,U,E,2C)	(U,U,E,2C)	(U,U,E,2C)	—	(U,U,E,2C)
(E,E,E,3C)	(U,U,E,2C)	(E,E,E,3C)	(E,E,E,3C)	(E,E,E,3C)	(E,E,E,2C,c-am)	(E,E,E,3C)

Table 3.6: Resulting equivalence classes L^{-}_{j+1} after adding the connection types in Figure 3.6 to each L^{+x}_{j} equivalence class, dashed lines indicating infeasibility or suboptimality

	(1)	(2)	(3)	(4)	(7)	(8)	(9)	(14)
(U,U,0,1C)	(U,U,0,1C)	—	—	—	—	—	—	—
(U,0,U,1C)	—	(U,0,U,1C)	—	—	—	—	—	—
(0,U,U,1C)	—	—	(0,U,U,1C)	—	—	—	—	—
(E,0,0,1C)	—	—	—	(E,0,0,1C)	—	—	—	(0,0,0,1C)
(E,U,U,1C)	—	—	(0,U,U,1C)	—	(E,U,U,1C)	—	—	—
(U,E,U,1C)	—	(U,0,U,1C)	—	—	—	(U,E,U,1C)	—	—
(U,U,E,1C)	(U,U,0,1C)	—	—	—	—	—	(U,U,E,1C)	—
(E,U,U,2C)	—	—	—	—	(E,U,U,2C)	—	—	—
(U,E,U,2C)	—	—	—	—	—	(U,E,U,2C)	—	—
(U,U,E,2C)	—	—	—	—	—	—	(U,U,E,2C)	—

	(4)	(5)	(6)	(10)	(11)	(12)	(13)	(14)
(0,0,0,0C)	—	—	—	—	—	—	—	(0,0,0,0C)
(0,0,0,1C)	—	—	—	—	—	—	—	(0,0,0,1C)
(0,E,0,1C)	—	(0,E,0,1C)	—	—	—	—	—	(0,0,0,1C)
(0,0,E,1C)	—	—	(0,0,E,1C)	—	—	—	—	(0,0,0,1C)
(E,E,0,1C)	(E,0,0,1C)	(0,E,0,1C)	—	(E,E,0,1C)	—	—	—	(0,0,0,1C)
(E,0,E,1C)	(E,0,0,1C)	—	(0,0,E,1C)	—	(E,0,E,1C)	—	—	(0,0,0,1C)
(0,E,E,1C)	—	(0,E,0,1C)	(0,0,E,1C)	—	—	(0,E,E,1C)	—	(0,0,0,1C)
(E,E,E,1C)	(E,0,0,1C)	(0,E,0,1C)	(0,0,E,1C)	(E,E,0,1C)	(E,0,E,1C)	(0,E,E,1C)	(E,E,E,1C)	(0,0,0,1C)
(E,E,0,2C)	—	—	—	(E,E,0,2C)	—	—	—	—
(E,0,E,2C)	—	—	—	—	(E,0,E,2C)	—	—	—
(0,E,E,2C)	—	—	—	—	—	(0,E,E,2C)	—	—
(E,E,E,2C,a-mc)	—	—	—	(E,E,0,2C)	(E,0,E,2C)	—	(E,E,E,2C,a-mc)	—
(E,E,E,2C,m-ac)	—	—	—	(E,E,0,2C)	—	(0,E,E,2C)	(E,E,E,2C,m-ac)	—
(E,E,E,2C,c-am)	—	—	—	—	(E,0,E,2C)	(0,E,E,2C)	(E,E,E,2C,c-am)	—
(E,E,E,3C)	—	—	—	—	—	—	(E,E,E,3C)	—

A summary of this procedure is provided in Figure 3.12. It can be observed that the complexity of the algorithm is $O(n)$, where n refers to the number of pick aisles. This is determined by the *while* loop, called for each aisle. An example problem is illustrated in Figure 3.13. For space saving reasons, the detailed solution procedure is not given, but the resulting optimal solution, whose total distance is 106 units, is depicted in Figure 3.14.

```
begin
- j ← 1;
- set optimal of all 25 configurations of L_1^- PTS to zero;
while j ≠ n do
-   add each configuration in Figure 3 to each of the 25 L_j^- PTS to obtain optimal 25 L_j^{+y} PTS;
-   add each configuration in Figure 3 to each of the 25 L_j^{+y} PTS to obtain optimal 25 L_j^{+x} PTS;
-   add each configuration in Figure 10 to each of the 25 L_j^{+x} PTS to obtain optimal 25 L_{j+1}^- PTS;
-   j ← j+1;
endwhile
-   compute the last aisle in the same way as in the first step in the while loop;
-   select the minimum of the eight L_n^+ solutions as the optimal solution;
end
```

Figure 3.12: Summary of the procedure of Roodbergen and De Koster [72] for the 3-OPP

3.2.4 Heuristic Procedures for the k-OPP with $k \geq 3$

Although a polynomial time algorithm is provided for the 3-OPP, it can be observed that even for a single middle aisle, the procedure is too complicated to implement in practice, let alone the case with multiple middle aisles. Consequently, various heuristic procedures have been developed and applied in practice for warehouses with middle aisles. These include conventional heuristics such as S-shaped and largest gap, the DP-based aisle-by-aisle heuristic due to Vaughan and Petersen [82], another DP-based heuristic called the combined heuristic due to Roodbergen and De Koster [73], and the combined$^+$ heuristic, which is an improvement heuristic over combined and also due to Roodbergen and De Koster [73].

S-shape heuristic is the simplest of the heuristic procedures for the k-OPP with $k \geq 3$ and hence is the easiest to implement. The picker first moves to the left end of the middle aisle which is nearest to the back cross aisle. Then, all the items in the upmost block are picked using the S-shape procedure for a single block. When the front end of the last nonempty aisle is reached, the picker checks the distance to the back ends of the leftmost and rightmost

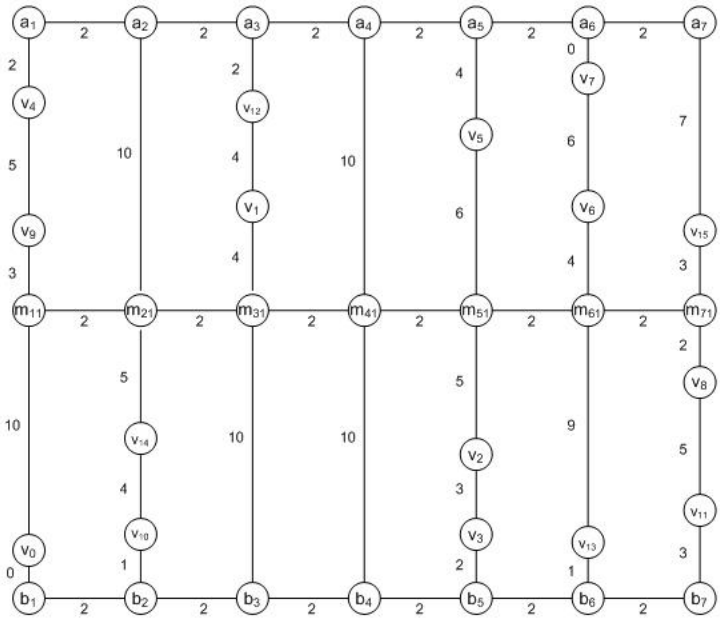

Figure 3.13: An example problem with a single middle aisle, 7 pick aisles and 15 items

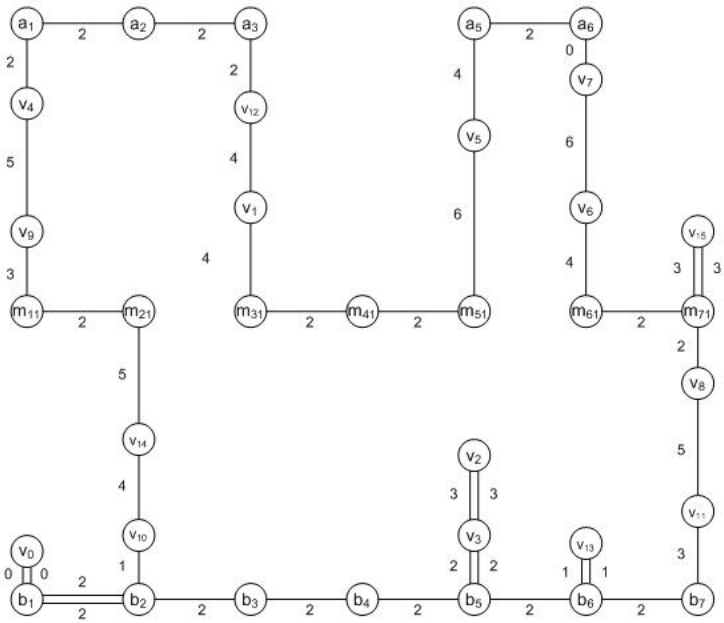

Figure 3.14: Optimal solution of the problem in Figure 3.13

nonempty subaisles having items not picked yet, and moves to the nearer one. Then, all the items in that block are picked, and so on. When picking of the items in the downmost block is finished using the same procedure, the picker moves to the depot, and the route is completed. Figure A.2 in Appendix A shows the S-shape heuristic solution for the warehouse given in Figure 3.1, whose optimal solution is shown in Figure A.1 in Appendix A. The optimal solution has a total length of 164, whereas the resulting S-shape solution has a total travel distance of 192, with an optimality gap of 17.1%.

Largest gap heuristic starts by moving to the left end of the middle aisle nearest to the back cross aisle. Then, all the items in the upmost block are picked using the largest gap heuristic for the 2-OPP. When the picking of all the items in the block is completed, the picker moves to the back end of the leftmost subaisle of the next block with picks, and picks all the items on that block using the largest gap heuristic for the 2-OPP, and so on. When picking of items in the downmost block is completed, the picker completes the tour by returning to the depot. Figure A.3 in Appendix A illustrates the resulting largest gap solution to the example problem shown in Figure 3.1. The solution has a total travel distance of 190, which indicates an optimality gap of 15.9%.

The largest gap heuristic, like the S-shape heuristic, is simple to implement in practice, but the optimality gap is generally large. The heuristic fails to find good solutions especially when the pick items are located near the middle of the subaisles. Additionally, due to excessive travel on the middle aisles, solution quality is also bad when the relative length of the middle aisles to the length of the subaisles gets larger.

Vaughan and Petersen [82] have proposed a DP-based heuristic that limits the number of visits to each pick aisle to one. Pick aisles constitute the stages and in each stage of the algorithm, for each cross aisle i, one needs to determine the distance required to start at the depot, pick all the items in the pick aisles up to i, and exit aisle i from cross aisle j. The optimal solution is the one that picks all the items and ends at the front end of the rightmost pick aisle. Figure A.4 in Appendix A provides the resulting solution for the example problem given in Figure 3.1. Total distance is 190, with an optimality gap of 15.9%.

The main drawback of the procedure stems from the fact that all pick aisles can be visited exactly once. This may decrease the quality of solutions especially when the items are located near the front and back ends of the pick aisles only. This requires that all pick aisles be com-

pletely traversed (and most of the time, almost twice), resulting in excessive travel especially when the relative length of the pick aises relative to the cross aisles is large.

Another DP-based heuristic has been proposed by Roodbergen and De Koster [73], called the *combined* heuristic, and is based on dividing the problem into blocks (a horizontal division), as opposed to dividing the problem into pick aisles (a vertical division) in the aisle-by-aisle heuristic procedure. For each block, the solution procedure is similar to that of Ratliff and Rosenthal [71] for 2-OPP. However, the main idea is whether to enter and exit each subaisle from the front or back end. Additionally, similar to the idea of Vaughan and Petersen [82], each subaisle (instead of a pick aisle) can only be visited once. Again, the solution procedure starts by moving to the left end of the middle aisle nearest to the back cross aisle. Then, the DP-based procedure is applied for the upmost block. When the front end of the last nonempty aisle is reached, the picker checks the distance to the back ends of the leftmost and rightmost nonempty subaisles having items not picked yet, and moves to the nearer one. Items in the next blocks are picked in the same manner, and the picker moves to the depot to complete the tour when all items are picked. Figure A.5 in Appendix A gives the combined heuristic solution for the problem given in Figure 3.1, whose total travel distance is 182, with an optimality gap of 11.0%.

A further improvement procedure on the combined heuristic has also been proposed by Roodbergen and De Koster [73]. Based on the drawbacks of the heuristic, two main improvements are put forward. First, picking the items from left to right in the downmost block results in the picker ending his route on the rightmost end of the block, requiring a considerable part of the front cross aisle back to the depot. Consequently, the route is modified so that the downmost block is traversed from right to left, which provides a solution not worse than the original procedure. A second improvement considers traversal of the first pick aisle until reaching the upmost block, which might require traversal of empty aisles. In order to avoid this, the route is considered in two phases. Items on the first x aisles on the left are picked while moving upward, and items on the remaining aisles are picked on the downward route. The value of x is determined using dynamic programming, varying the value until the number of pick aisles. This improved procedure is named the *combined*$^+$ heuristic. For the example provided in Figure 3.1, this procedure cannot improve the solution value of 182, which is obtained for x values 1 and 3.

On average, the combined$^+$ procedure performs better than its counterparts in the literature, as will be discussed in Section 4.5. However, for certain problem types, namely when the item density in the warehouse is large, the procedure is observed to come up with relatively high optimality gaps. In these cases, the aisle-by-aisle and largest gap heuristics may provide better solutions.

3.3 Complexity of the k-OPP with $k \geq 4$

This section provides a summary for the literature survey conducted on the background and complexity of the order-picking problem in parallel-aisle warehouses. Although it has been proven two and a half decades ago by Ratliff and Rosenthal [71] that the problem with no middle aisles is polynomially solvable, and by Roodbergen and De Koster [72] that the problem can be polynomially solved when a single middle aisle exists, the complexity of the case with multiple middle aisles is still an open problem, as stated before.

The organization of this section is as follows: Section 3.3.1 gives computational complexity results in the literature related to the Traveling Salesman Problem (TSP) and analyzes the special cases of TSP on series-parallel and grid graphs, as well as mentioning the Graphical Traveling Salesman Problem (GTSP). Section 3.3.2 discusses the literature on the more general Steiner Traveling Salesman Problem (StSP), where the complete definition of the related polyhedron by Baiou and Mahjoub [5] is given for the case of series-parallel graphs. The section also includes the Steiner Tree Problem (STP), which is closely related to the StSP. The last part of the section is on the Graphical Steiner Traveling Salesman Problem (GStSP), which includes the OPP as a special case. The section is then concluded in Section 3.3.3.

3.3.1 The Traveling Salesman Problem

The Traveling Salesman Problem can simply be stated as follows:

PROBLEM: TSP
INSTANCE: Integer $n \geq 3$ and $n \times n$ integer matrix $C = (c_{ij})$, where each $c_{ij} \geq 0$. Let $\Pi(i)$ denote the integer following i in a permutation.
QUESTION: Which cyclic permutation Π of the integers from 1 to n minimizes the sum

$\sum_{i=1}^{n} c_{i\pi(i)}$?

The relation of the TSP to the order-picking problem is as follows: the k-OPP is a generalized version of the TSP on a specially structured graph, that is, the OPP allows for multiple visiting of the nodes and not all the nodes have to be visited. If the TSP is NP-complete on this specially structured graph, so is the k-OPP. The converse is not necessarily true: if the TSP is polynomially solvable on the graph, we cannot draw any direct conclusion on the complexity of the k-OPP.

Johnson and Papadimitriou [52] and Garey and Johnson [33] have shown that the problem is NP-complete in general graphs. Their proofs of NP-completeness of the TSP have been provided in Appendix B.

Although the TSP is NP-complete in general graphs, there are various special cases in which the problem is polynomially solvable. Burkard et al. [13] provide a survey of polynomially solvable cases of TSP and among these, related to the OPP is the case of k-line TSP in the Euclidean plane. If n vertices of the graph lie on k lines parallel to each other, then for a fixed value of k, there exists an $O(n^k)$ algorithm for finding the optimal tour. The difference of the OPP from the k-line TSP in the Euclidean plane is that the OPP has vertices (denoting items) on the lines parallel to each other as being in the k-line TSP, but is not in the Euclidean plane. An even more related case in [13] to the OPP is the Graphical TSP, but this will be discussed later in this section.

In Kabadi [53], more emphasis is given to the case of *Gilmore-Gomory scheme*, where the optimal solution is found by applying the subtour patching algorithm discussed in [13] to a patching pseudograph. An alternative definition of the Gilmore-Gomory TSP, which is called the minimum cost connected directed pseudograph problem with node deficiency requirements (MCNDP), is given. A node deficiency function is defined as $d : V \to \mathbb{Z}$ such that (i) $\sum_{i \in V} d_i = 0$ and (ii) $d_i = 0$ for all arcs not incident to i. The 2-OPP is an undirected version of this problem with no node deficiency requirements. Because of these restrictions, the two-step algorithm presented for the general MCNDP reduces to a DP scheme with a complexity of $O(k)$, where k denotes the number of pick aisles. This is another path that leads to the polynomially solvable status of 2-OPP. Since the k-OPP with $k \geq 3$ cannot be reduced to the MCNDP, there is no conclusion on the complexity of the general k-OPP depending on this study.

The TSP on Specially Structured Graphs

The TSP on Series-Parallel Graphs

Our interest in the complexity of the TSP on series-parallel graphs stems from the fact that the 2-OPP is a more generalized case of the TSP on a series-parallel graph.

Series-parallel graphs, due to their special structure, provide tractability opportunities for some classes of combinatorial problems that are NP-complete for general graphs. Takamizawa et al. [78] observe that three types of combinatorial problems are polynomially solvable on S-P graphs:

- The decision problem with respect to a property characterized by a finite number of induced or homeomorphic graphs, in which one would like to decide whether the input graph satisfies the property
- The minimum edge deletion problem with respect to the same property as above
- The generalized matching problem

The *minimum vertex cover problem* consists of finding the set of edges with minimum weight that cover all the vertices in a graph. Since the minimum vertex cover problem is a decision problem satisfying the first property above, as the decision problem is to find whether there exists a vertex cover of size less than some K for the graph, it is polynomially solvable on S-P graphs. It has been shown in [33] that the vertex cover problem transforms to Hamiltonian circuit and hence the TSP. Vertex cover being polynomially solvable in S-P graphs implies that the TSP may not be NP-complete on S-P graphs. Indeed, we will see later in Section 5 that the more general Steiner Traveling Salesman Problem is polynomially solvable in S-P graphs, leading to the conclusion that the TSP is also polynomially solvable in S-P graphs.

The TSP on Grid Graphs

The discussion of grid graphs relates to the k-OPP in the sense that the case of the k-OPP with no items to be picked (or items on the corners of the sub-aisles) reduces to a more generalized version of the TSP on a rectangular solid grid graph. If the TSP is NP-complete on such a graph, then so is the k-OPP. Again, the converse is not necessarily true.

As discussed in Appendix B, the complexity of finding the optimal TSP route on a graph is equivalent to finding whether there exists a Hamiltonian cycle on this graph. For the TSP on grid graphs, three cases will be examined, from more general to more specific.

The initial study on the complexity of finding a Hamiltonian cycle on grid graphs is by Itai et al. [49], in which they prove that finding such a cycle, hence solving the TSP on general grid graphs is NP-complete. This immediately leads to the fact that the Euclidean TSP is also NP-complete.

For the more restricted case of solid grid graphs, that is, grid graphs with no "holes", Umans and Lenhart [80] prove that the TSP is polynomially solvable. Before summarizing how the procedure works, it will be useful to give some additional definitions:

Definition 3.6 *A 2-factor of G is a spanning subgraph with all vertices having degree two.*

Definition 3.7 *If F is a 2-factor of G, an alternating cycle is a cycle whose edges are alternately in F and not in F. Flipping an alternating cycle means removing the edges of the cycle in F and adding the ones not in F.*

Definition 3.8 *An alternating strip is one of the subgraphs shown in Figure 3.15. The ones in (a) are odd alternating strips, whereas the ones in (b) are even alternating strips.*

Flipping an odd alternating strip reduces the number of components of the 2-factor by one, while flipping an even alternating strip can in the best case keep the number of components in the 2-factor constant.

Definition 3.9 *A sequence of alternating strips is static if: 1) the strips do not share an area, and 2) no two strips share an edge, except two consecutive ones.*

The idea is as follows: Suppose we have a 2-factor of G. It is proven that if G is Hamiltonian, then it contains an alternating strip sequence. Flipping an alternating strip sequence may cause overlaps between the strips. However, it is also proven that if G is Hamiltonian, then it contains a static alternating strip sequence, and flipping the strips in this type of strip sequence does not cause overlaps. In order to decrease the number of components in the 2-factor, an odd

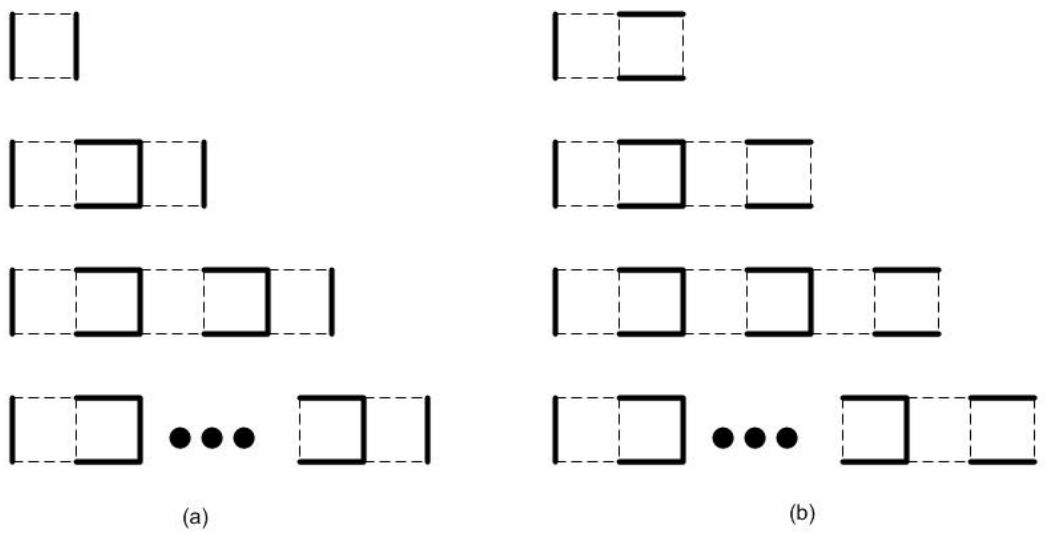

Figure 3.15: Odd (a) and even (b) alternating strips

alternating strip must be flipped. If the initial 2-factor does not contain any odd alternating strips, then a sequence of flips is sure to end up with one. Therefore in each iteration, a static alternating strip sequence is found and consecutive flips are performed. If G is Hamiltonian, then the procedure ends up with a Hamiltonian tour. Otherwise, the procedure terminates when no more static alternating strip sequences are found.

If G contains n vertices, then a static alternating strip can be found in $O(n^3)$ time. The procedure repeats at most n times, increasing the overall complexity to $O(n^4)$. Therefore finding a Hamiltonian cycle in a solid grid graph takes polynomial time. However, the complexity of finding a Hamiltonian cycle in solid grid graphs with holes of restricted form is an open problem.

For the case of rectangular solid grid graphs $R(m, n)$, the Hamiltonicity problem (which is certainly polynomially solvable due to [80]) is much easier to solve. Salman et al. [76] provide necessary and sufficient conditions for a rectangular solid, *eta* and *omega* grid graph. The following definitions and theorems are due to [76].

Theorem 3.10 *An $R(m, n)$ grid graph is Hamiltonian only if $m \cdot n$ is even.*

Definition 3.11 *An eta grid graph $\eta(s, t)$ is a subgraph of $R(3s-2, 3t-2)$ induced by $V(3s-2, 3t-2) \backslash \{(uv) | u = s+1, s+2, \ldots, 2s-2$ and $v = 1, 2, \ldots, 2t-2\}$.*

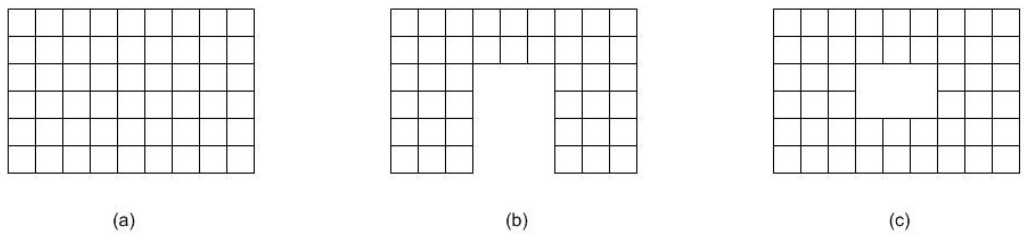

Figure 3.16: (a) $R(10, 7)$ (b) $\eta(4, 3)$ (c) $\omega(4, 3)$

Definition 3.12 *An omega grid graph $\omega(s, t)$ is a subgraph of $R(3s - 2, 3t - 2)$ induced by $V(3s - 2, 3t - 2)\backslash\{(uv)|u = s + 1, s + 2, \ldots, 2s - 2$ and $v = t + 1, t + 2, \ldots, 2t - 2\}$.*

Figure 3.16 shows a rectangular solid grid graph $R(10, 7)$, an eta grid graph $\eta(4, 3)$ and an omega grid graph $\omega(4, 3)$.

Theorem 3.13 *An eta grid graph $\eta(s, t)$ is Hamiltonian only if $s \cdot t$ is even. An omega grid graph $\omega(s, t)$ is always Hamiltonian.*

The k-OPP with $k \geq 3$ contains the TSP on a rectangular solid grid graph as a special case. Although the latter problem is polynomially solvable as shown by [76] and [80], we cannot draw any conclusion about the complexity of k-OPP.

An interesting problem useful in the analysis of the complexity of the OPP is the *center cycle problem*. The objective in this problem is to find a cycle that minimizes the maximum distance of a vertex from the cycle. Foulds et al. [31] prove that the problem is NP-complete in general graphs, as the decision problem of finding whether a graph contains a center cycle with maximum distance 0 from all nodes is equivalent to finding out whether the graph contains a Hamiltonian cycle, which is shown to be NP-complete by [33] and [52]. However, Foulds et al. [31] also show that the problem is trivially solvable in rectangular solid grid graphs. The problem is as follows: If the graph has a Hamiltonian cycle (trivially solvable due to [76]), then the optimal solution is 0. Otherwise, it is always possible to construct a tour that misses only one vertex. Therefore the optimal solution is 1 for this case.

The connection of the center cycle problem to the OPP is by the vertex cover problem. The result in [31] indicates that the decision problem related to the vertex cover problem always

has the answer "yes" for rectangular solid grid graphs, hence is polynomially solvable. This conclusion prevents us from arriving at a result on the complexity of the k-OPP using the center cycle problem.

The Graphical TSP

The Graphical Traveling Salesman Problem (GTSP) can be defined as follows [30]: Find a minimal tour in $G = (V, E)$ in the form:

$T = [v_1, e_1, \ldots, v_i, e_i, e_{i+1}, \ldots, v_p, e_p, v_{p+1}]$ such that:

1. The end nodes of e_i are v_i, v_{i+1} for $i = 1, \ldots, p$.
2. $v_1 = v_{p+1}$.
3. Each node of V appears **at least once** in T.

It can be observed here that the only difference from the TSP is the chance of using every node more than once. In this sense, the GTSP generalizes the TSP, implying that in every case the TSP is NP-hard, so is the GTSP. This bears the immediate conclusion that the GTSP is NP-hard on general graphs and general grid graphs.

The main motivation of studying the Graphical TSP is due to the fact that the k-OPP belongs to this more general class of Traveling Salesman Problem, rather than the classical TSP itself. This is caused by the availability of visiting each node in the warehouse graph more than once, which is not allowed in the classical TSP. Therefore it is useful to go into detail for the GTSP on specially structured graphs.

The GTSP on Series-Parallel Graphs

It has been discussed before that the graph for the 2-OPP has S-P graph property and the TSP is polynomially solvable on series-parallel graphs. If GTSP is polynomially solvable on series-parallel graphs, this implies that 2-OPP is also polynomially solvable.

Cornuejols et al. [21] try to define the polyhedron defined by the GTSP. They find facet-defining inequalities, namely the path inequality, bicycle and wheelbarrow inequalities. Then four different polyhedra are defined, for the last two of which the TSP polyhedron is included as a face. An attempt to completely define the polyhedron in S-P graphs, in which the problem

is polynomially solvable, is made. However, a complete definition cannot be reached, and this definition problem remains open up to now. Lastly, the DP-based algorithm by Ratliff and Rosenthal [71] is extended to the case of S-P graphs. It is also stated that the algorithm can be modified to solve the Steiner Tree Problem (which will be discussed in the next section) in S-P graphs by disregarding some of the possible cases which would not occur in a tree.

Fonlupt and Nachef [30] study the cases where the GTSP is polynomially solvable. They first define a generalized form of GTSP, which they call the GETSP. This problem assings costs to edges depending on the number of times they are used in the tour. Note that the GTSP is the special case of this, where the cost of using an edge is an n-multiple of the cost of using it once if the edge is used n times. It is obvious that the GTSP is polynomially solvable in every case the GETSP is also polynomially solvable. Then, they define deletion and contraction operations, claiming that if the underlying graph is S-P, then solving the GTSP is equivalent to solving the GETSP on the reduced graph. Consequently, a dynamic programming algorithm is proposed, similar to that by [71].

As a consequence, the GTSP on S-P graphs is polynomially solvable, implying that 2-OPP is also polynomially solvable. A complete polyhedral definition for this problem remains open.

3.3.2 The Steiner Traveling Salesman Problem (StSP)

The Steiner Traveling Salesman Problem (StSP) can be defined formally as follows:

PROBLEM: StSP
INSTANCE:
A set $T = \{1, 2, \ldots, m\}$, $S = \{m+1, m+2, \ldots, n\}$ and an $n \times n$ cost matrix $C = (c_{i,j})$.
QUESTION:
Which cyclic partial permutation Π (with cardinality $n(\Pi)$) of the integers from 1 to n that includes all the integers in T minimizes the sum $\sum_{i=1}^{n(\Pi)} c_{i\pi(i)}$?

The set T is the set of terminal nodes, which have to be visited, and set S is the set of Steiner nodes, which do not necessarily have to be visited. The problem is to find a cycle spanning all terminal nodes. It may or may not include all (or any) Steiner nodes. Obviously, the problem is NP-complete, as with $S = \emptyset$, we have the classical Traveling Salesman Problem, which is NP-complete.

The importance of StSP for the k-OPP is similar to the importance of TSP for GTSP. The k-OPP is actually the graphical (and therefore more generalized) version of StSP on a specially structured graph. If StSP is NP-complete on this class of graphs, then so is the k-OPP.

The StSP is closely related to the *minimum-weight two-connected spanning network problem*. Monma et al. [65] characterize the optimal solution to this NP-hard problem when the graph is complete and triangle inequalities are satisfied. The optimal solution has all vertices with degree 2 or 3, and deleting any edge or pair of edges from the optimal solution leaves a bridge. Note that the optimal solution to this problem need not be a tour, therefore the problem is different from the Hamiltonian cycle problem. A further finding is that $\tau \leq \frac{4}{3}Q_2$ when $S = \emptyset$, where τ is the optimal solution to the Steiner tour and Q_2 is the optimal solution to the 2-edge connected subgraph.

Baiou and Mahjoub [5] prove that the problem is polynomially solvable for S-P graphs, and also provide the associated polyhedron. For achieving this, they first define the problem of r-TSP as follows: Given a root node r, weights w_e on edges and c_v on vertices, find a cycle with minimum cost that spans r. If $x_e = 1$ if edge i is in the tour and 0 otherwise, $y_v = 1$ if vertex v is included in the tour and 0 otherwise, the integer formulation is given for the polytope. After proving that the integer relaxation of this problem gives the convex hull for S-P graphs, they project this polytope onto the edge variables, and selecting any arbitrary terminal node as the root node gives the polytope for StSP. Naturally, integer relaxation of this formulation gives integer results for S-P graphs.

The StSP being polynomially solvable in S-P graphs directly implies that the TSP is also polynomially solvable in S-P graphs, as the first problem generalizes the latter. Yet there is no direct conclusion on the k-OPP from this either, since the k-OPP allows multiple visits to nodes and generalizes the StSP. In addition, for $k \geq 3$, the S-P structure no longer exists.

The Steiner Tree Problem (STP)

Closely related to the StSP, the Steiner Tree Problem (STP) is the problem of finding the minimum-weight tree that spans the subset T of terminal nodes of V. The problem is NP-complete for general graphs [55], and the transformation providing NP-completeness comes from Satisfiability → 3-Satisfiability → 3-Dimensional Matching → Exact Cover by 3-Sets → STP.

The problem is known to be NP-hard for general planar graphs due to [34], general bipartite graphs and general grid graphs. It is also NP-complete for the problems when distances are rectilinear. Aho et al. [3] provide a polynomial algorithm for a special case on a grid graph, where the distance metric is rectilinear and the terminal nodes are on the edges of the graph. Ho et al. [47] present a minimum spanning tree-based heuristic for the case with rectilinear distances assuming an underlying grid graph. The heuristic runs $O(n^2)$ time. Grötschel et al. [42] consider the more general case when N Steiner trees are to be constructed on a single rectangular grid graph and the terminal nodes lie either on two opposite or all four sides of the rectangle. After giving facet-defining inequalities, they propose a cutting-plane based heuristic procedure that runs in polynomial time.

Chopra and Rao [15] present two integer programming formulations of the general problem, the first being undirected and the second being directed. It is proven that the directed formulation is tighter than the undirected one.

Chopra and Rao in [16] focus on the NP-hard special cases grid graphs and bipartite graphs. They introduce the odd-wheel inequality, which is a facet-defining inequality on grid graphs. Additionally, they show that for rectangular grid graphs, the polyhedron also has Steiner partition and odd-hole facets. The bipartite inequalities are shown to be facet-defining for bipartite graphs. It is conjectured that odd-hole and Steiner partition facets completely define the tree polyhedron on a 2-tree. Margot et al. have given a complete description of the STP polytope in 2-trees in [61], almost at the same the time the conjecture was made.

Goemans [39], in a similar study to [5], tries to characterize the polytope defined by the STP on S-P graphs. As in [5], [39] formulates an r-tree problem, in which the tree only has to span a root node r and by projecting the polytope onto the edge variables, a large (but not complete) class of facet-defining inequalities is obtained. Yet a complete characterization of the Steiner tree polyhedron on S-P graphs still remains open.

The path from the complexity of the STP to that of OPP is a long and complicated one. The STP being NP-complete on general graphs, general grid graphs, general bipartite graphs implies that the TSP, StSP and GStSP are also NP-complete on these graphs, yet all these have been separately proven. However, the difficulty of the case with rectangular solid grid graphs (recall that polynomial algorithms have been proposed only for the cases where terminal vertices are on the edges of the rectangle) could be helpful for proving the NP-completeness of

the OPP, although this transformation is still "open".

The Graphical Steiner TSP (GStSP)

The Graphical Steiner TSP combines the properties of StSP and GTSP into the same problem: It is a Steiner problem in the sense that not all the vertices of the graph have to be visited. It is also graphical owing to the fact that the vertices can be visited more than once. To the best of our knowledge, the only problems studied that can be classified into the GStSP class are the order-picking problems mentioned beforehand.

3.3.3 Conclusion on Complexity

In this section, we have tried to provide the computational complexity results of the problems in the literature related to the order-picking problem. These include the traveling salesman problem (TSP), the Steiner TSP (StSP), the Steiner tree problem (STP), the graphical TSP (GTSP) and the Graphical Steiner TSP (GStSP). All of these problems are NP-complete in general graphs and grid graphs. We have explored the special cases of solid grid graphs, rectangular solid grid graphs and series-parallel graphs, in which some of these problems are polynomially solvable. We have seen that the TSP and the STP are polynomially solvable on solid grid graphs and rectangular solid grid graphs and all the problems mentioned are polynomially solvable on S-P graphs. The 2-OPP shows the structure of S-P graphs, hence is polynomially solvable. For the case of k-OPP with $k \geq 4$, however, no direct result on complexity can be obtained depending on the results in the literature. As a conjecture, we claim that although a polynomial time algorithm for fixed k can be found by extending the algorithm proposed in [71], the problem is NP-complete in terms of the number of cross aisles.

Figure 3.17 summarizes the findings of the studies discussed in this section, along with the studies that have come up with the corresponding complexity result. The open status of the complexity of the k-OPP with $k \geq 4$ can also be observed from this figure, as the problem does not generalize an NP-complete problem or is not generalized by a tractable problem, no direct conclusion can be drawn on its complexity.

Thick and thin solid lines in Figure 3.17 represent NP-complete and tractable problems respectively. Thick dashed lines representing the problems whose NP-completeness can be

deduced from the ones in this figure and thin dashed lines representing the problems whose tractability can be deduced from the ones in this figure. The k-OPP with $k \geq 4$ is open.

In order to discuss the complexity results in Figure 3.17, we start with the NP-completeness of the TSP on general graphs, due to Johnson and Papadimitriou [52] and Garey and Johnson [33]. Since TSP is generalized by the StSP, the latter problem is also NP-complete on general graphs [5]. However, Baiou and Mahjoub [5] have shown that the StSP is polynomially solvable in series-parallel graphs. The TSP is also generalized by the GTSP and the GStSP, implying the NP-completeness of the problems on general graphs [21]. The STP is shown to be NP-complete on general graphs by Karp [55], . In turn, the RTSP is also shown to be NP-complete by Garey and Johnson [34], implying the NP-completeness of the STP [34], the TSP [49], the GTSP and the GStSP in grid graphs as well. Due to Cornuejols et al. [21], the GTSP is polynomially solvable in solid grid graphs and rectangular solid grid graphs, which brings about the tractability of the TSP [80], the STP and the RSTP on solid grid graphs and rectangular solid grid graphs as well. Cornuejols et al. [21] also report that on series-parallel graphs, the STP and the GTSP are polynomially solvable.

As for the complexity of the k-OPP is concerned, Ratliff and Rosenthal [71] have come up with a polynomial algorithm for the 2-OPP. The tractability of 2-OPP can also be proven by the fact that the GTSP, which generalizes the 2-OPP, is polynomially solvable on series-parallel graphs. The 3-OPP is polynomially solvable due to Roodbergen and De Koster [72]. The complexity of the k-OPP with $k \geq 4$ cannot be derived from these results, as we only know that it generalizes the polynomially solvable 2-OPP, 3-OPP, the GStSP on solid rectangular grid graphs, and the TSP in solid rectangular grid graphs.

In order to proceed with reaching a decision on complexity, one should either try to find an algorithm that is polynomial in terms of both the cross aisles and the pick aisles, or to try to show that solving the problem is as difficult as solving a known NP-complete problem. Depending an all the results discussed in this chapter, we conjecture that although for a fixed number of cross aisles the problem is polynomial, it is NP-complete in terms of the number of cross aisles. However, trying to reduce the problem to known NP-complete problems does not help to prove or disprove this.

The conjecture that the problem is NP-complete implies that unless $P = NP$, the problem cannot be solved to optimality using a polynomial time algorithm, and motivates the usage of

heuristic procedures to come up with good solutions in polynomial time. In Chapter 4, we propose such a heuristic procedure.

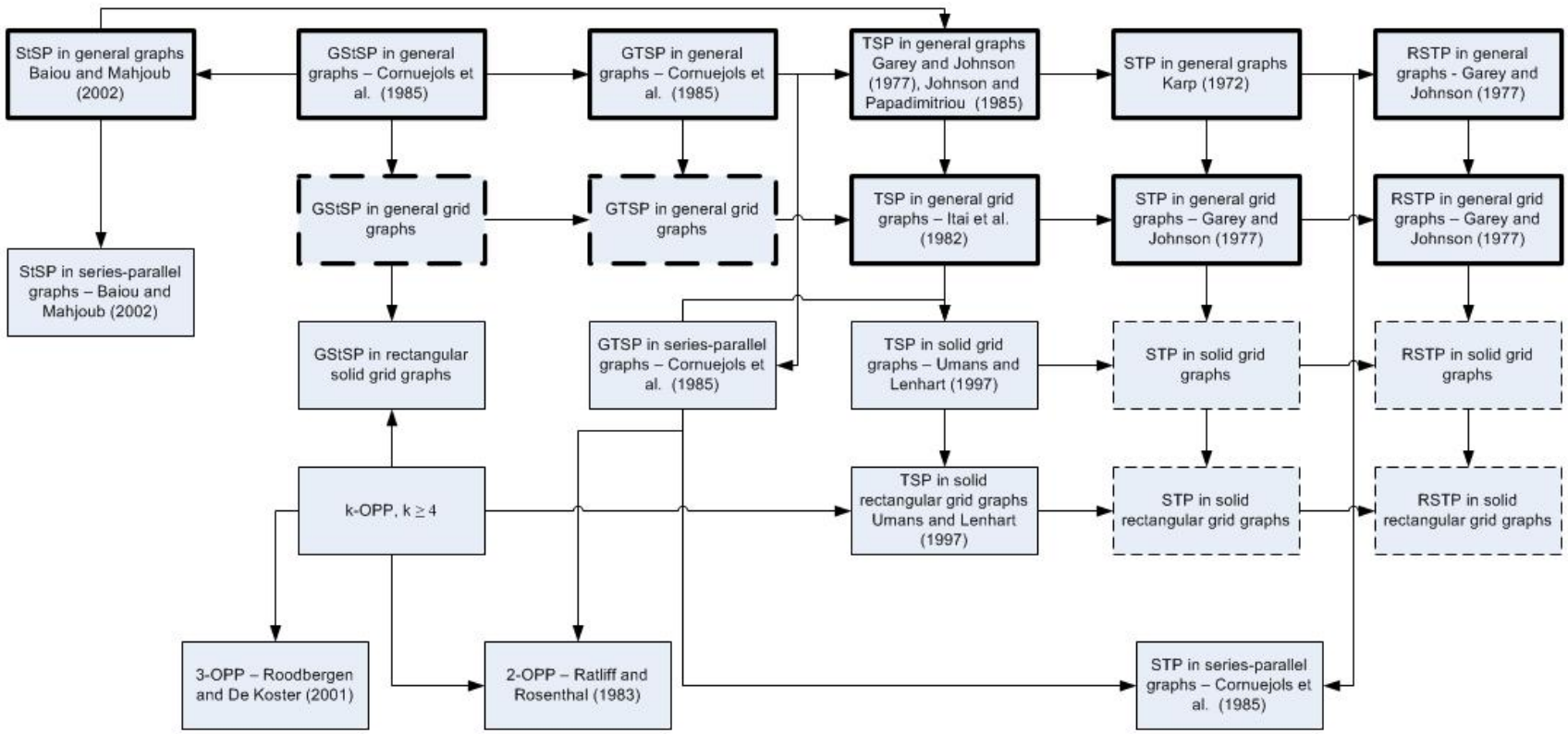

Figure 3.17: Summary of the complexity of the OPP-related problems

CHAPTER 4

THE ORDER-PICKING PROBLEM IN A PARALLEL-AISLE WAREHOUSE: SINGLE-PICKER CASE

This chapter mainly deals with the routing problem of a single picker in a parallel aisle warehouse where manual off-the-shelf order picking is employed, denoted as the k-OPP. For this case, two heuristic procedures will be proposed. The first one, called the *merge-and-reach* heuristic, makes use of the tractability of the 2-OPP. It is based on dividing the problem into subproblems using random cuts, solving each subproblem resulting from these cuts by first solving each block separately, then joining these to end up with a single tour for each subproblem, and lastly joining these tours into a single tour for the whole problem. The second one, called the *merge-and-reach*$^+$ heuristic, is an improvement on the merge-and-reach heuristic by applying 3-opt procedure.

The heuristics discussed in Section 3.2.4 work well for a set of problem parameters, yet all of them end up with relatively worse solutions for a certain set of problem parameters. The need for a more robust heuristic procedure which will retain its good performance for all set of problems has motivated the development of the *merge-and-reach* heuristic, which will be discussed throughout this section.

The main idea of the merge-and-reach heuristic bases itself on the tractability of 2-OPP. The procedure can be summarized as follows: First, c cut aisles are selected randomly and the problem is cut into $c + 1$ subproblems. Each subproblem is further divided into its blocks and each of these $k - 1$ blocks is solved to optimality using the algorithm by [71]. The next step is to bring together the solutions of each block in each subproblem to a single solution. In order to achieve this, in each subproblem, starting from the downmost pair of blocks, it checks whether the solutions to these two blocks overlap (share a common edge or vertex) or

not. If they do, then it "merges" the two solutions by deleting the unnecessary edges from the union of the two solutions. Otherwise, it finds the shortest way of joining these disjoint solutions, thereby making one "reach" the other. It proceeds until reaching the upmost block of the subproblem. The procedure is then reversed and the same steps are taken starting from the upmost pair of blocks, and ending with the downmost block. A solution starting from the downmost block is called a *bottom-up solution*, whereas a solution starting from the upmost block is called a *top-down solution*. The best of these solutions is then reported to be the solution to the subproblem. The last step performs the same procedure for the whole problem, treating solutions of the subproblems instead of solutions of the blocks. Again, the best of the two solutions is reported to be the merge-and-reach solution to the problem.

As stated before, the merge-and-reach$^+$ heuristic starts with a merge-and-reach solution and applies 3-opt improvement scheme. In each step, the move that gives the best improvement is selected as the next solution. The algorithm terminates when no further improvement is possible. The organization of this chapter is as follows: In Section 4.1, we give the subroutines used throughout the heuristic procedure; followed by Section 4.2, which defines the merge-and-reach heuristic procedure in detail; Section 4.3 implements the heuristic procedure on two example problems; Section 4.4 discusses the improvement procedures over the merge-and-reach heuristic to define the merge-and-reach$^+$ procedure; and lastly, Section 4.5 gives the computational results in order to observe the performance of both merge-and-reach and merge-and-reach$^+$, and to compare them with their counterparts in the literature.

4.1 Subroutines of the Algorithm

The merge-and-reach heuristic requires that a number of subroutines be called throughout the process. These include *procedure Ratliff_Rosenthal*, which solves the blocks to optimality; *procedure merge*, which joins two solutions when these solutions overlap (share a common edge and/or vertex); *procedure reach*, which joins two non-overlapping solutions; *procedure invert*, which converts a problem by inverting the blocks, so that a top-down solution can be accomplished using the same procedure as the bottom-up solution; and *procedure improve*, which is basically a 3-opt improvement procedure that is used when applying the merge-and-reach$^+$ heuristic.

The following definitions are used while explaining the subroutines throughout this section. When two partial neighboring solutions are considered, the lower solution is denoted as A and the upper one as B. VA_j and VB_j refer to the vertical edges (edges that correspond to the pick aisles) incident to the node corresponding to the jth aisle of the lower and upper solutions respectively. AB_j simply denotes (VA_j, VB_j). Furthermore, A_j and B_j denote the number of horizontal edges between the vertices corresponding to the jth and $j+1^{st}$ vertices of the lower and upper solutions respectively.

4.1.1 Procedure *merge*

The merge procedure is called for a pair of overlapping solutions. When two partial solutions overlap, there are two cases that might occur.

In the first case, there are no single vertical edges in any of the two solutions, that is, $VA_j \neq 1$ and $VB_j \neq 1$ for each j. This can be interpreted as follows: The lower block has a solution that collects all the items from its back cross aisle, and the upper block has a solution that collects all the items from its front cross aisle. In other words, these two partial solutions are the solutions to 1-OPPs in their corresponding block, where the single cross aisle in each problem refers to the back cross aisle of the lower block and the front cross aisle of the upper block. It can be shown that the merging problem in such a case is equivalent to solving 1-OPP on the lower (upper) block, where the "mirror images" of the items on the upper (lower) block with respect to the cross aisle between the two blocks are taken to the lower (upper block). The optimal sequence for this 1-OPP, which can easily be found, is also the optimal sequence for the merging problem. It can also be shown that the resulting solution is the optimal solution to the 3-OPP on these two blocks. Figure 4.1 illustrates such a case, where two overlapping partial solutions do not include any single vertical edge. Figure 4.2 shows the resulting merged solution.

In the second case, there exist single vertical edges in at least one of the partial solutions, which indicates that some of the items are picked using the front cross aisle of the lower solution and/or the back cross aisle of the upper solution as well. Therefore, it can be shown that the problem of merging the two partial solutions optimally is equivalent to solving the corresponding OPP on these two partial solutions. Hence, we concentrate our effort on the middle aisle, and assume that none of the vertical edges (either single or double) can be deleted. Un-

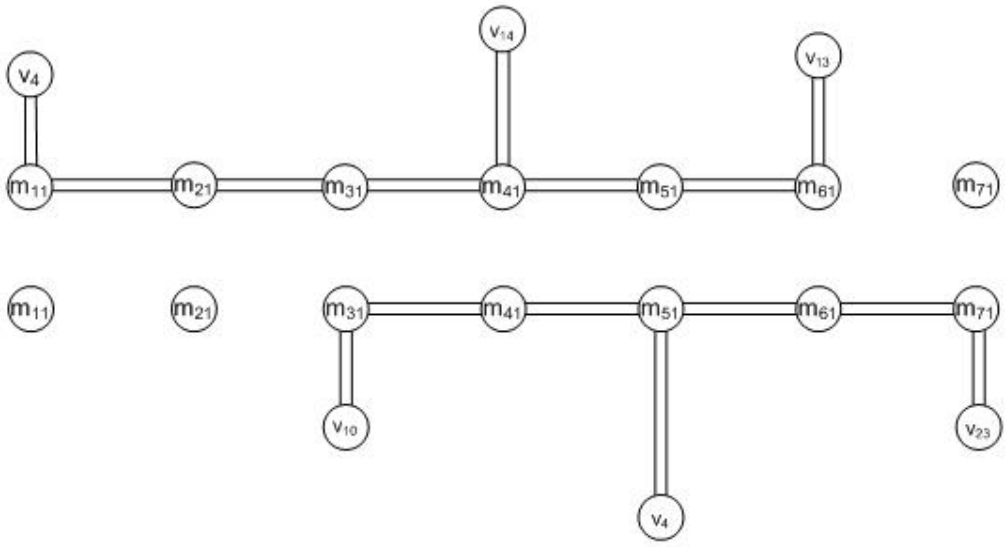

Figure 4.1: Two overlapping partial solutions with no single vertical edge

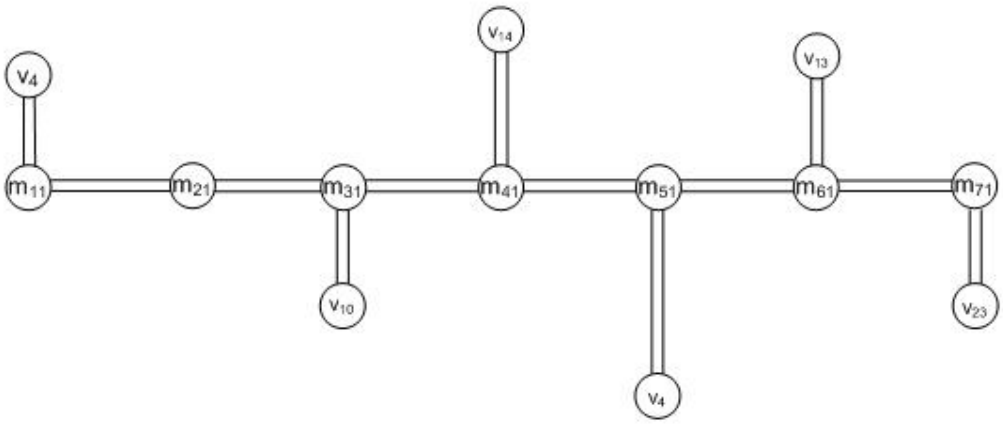

Figure 4.2: The resulting merged solution for the subproblem in Figure 4.1

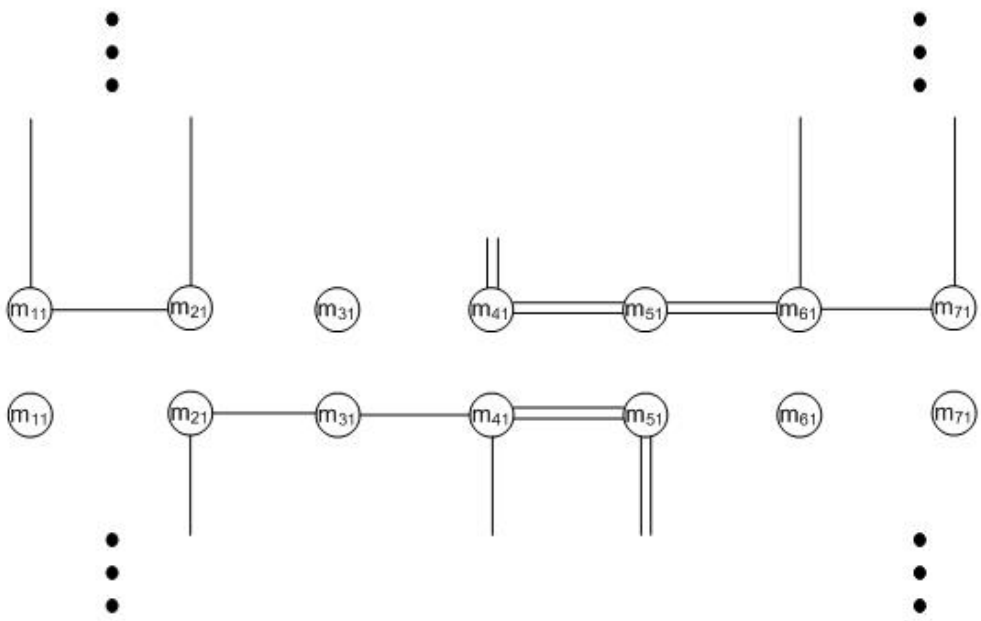

Figure 4.3: Two overlapping solutions with six single vertical edges

der this assumption, the merging problem reduces to minimizing the distance travelled on the middle aisle of the two blocks while keeping the Eulerian property of the graph corresponding to the OPP tour on the two partial solutions. This problem can be interpreted in graphical terms as minimizing the number of horizontal arcs on the middle aisle. Since the number of arcs corresponding to the same vertex pair exceeding two results in excessive edges, we delete a pair of edges when $AB_j = (2, 1)$ or $AB_j = (1, 2)$. We also know that if vertical edges cannot be modified, the pairs $AB_j = (0, 1)$ and $AB_j = (1, 0)$ also have to exist in the final solution in order to preserve the Eulerian property. Suppose that we delete all the remaining edge pairs on the middle aisle. Then the problem becomes one of matching unconnected double vertical edges to either connected double vertical edges or single vertical edges. It is possible that at the end of this procedure, we might end up with a disconnected graph. In order to overcome this, we use a greedy procedure that connects the unconnected subgraphs using double horizontal edges. This procedure optimally merges the two partial solutions when the vertical edges of the solutions cannot be deleted. In Figure 4.3, an example for this case is provided. The partial solutions include six vertical edges in total. Figure 4.4(a) depicts the situation when edge deletion is complete and (b) shows the resulting solution after matching.

Figure 4.5 summarizes the merge procedure. The if statement checks whether the first case

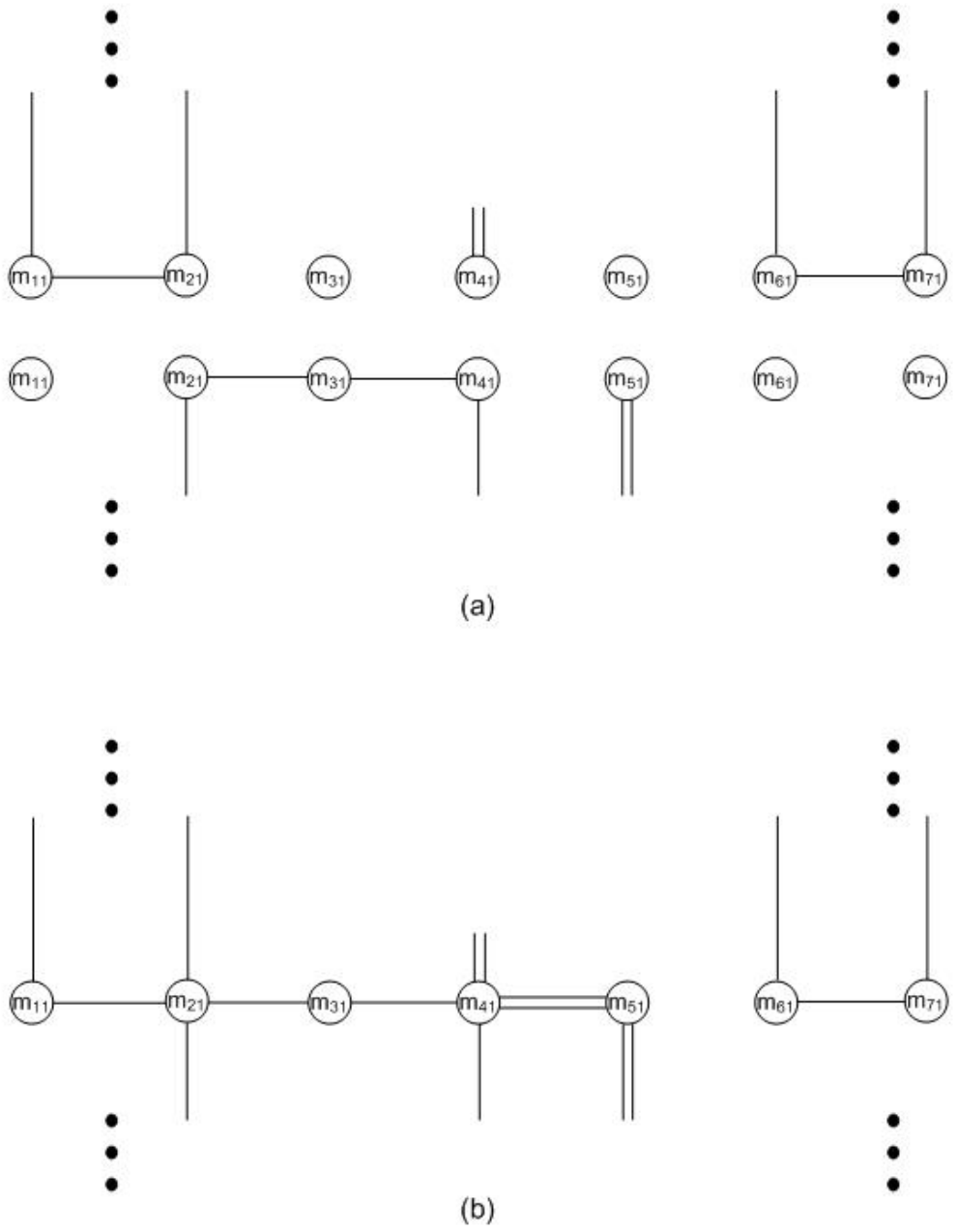

Figure 4.4: (a) The partial solution to the example subproblem in Figure 4.3 when edge deletion is complete, (b) the resulting solution after matching

is satisfied. If so, starting with the leftmost item(s), the picking process uses double vertical edges and ends after picking the rightmost item(s). Otherwise, we delete all AB_j pairs except those with $AB_j = (1,0)$ or $AB_j = (0,1)$, and solve the matching problem. With n aisles, the complexity of the merge procedure is $O(n^2)$, determined by the last *while* loop: k can be at most n, and we have to search for connectedness of vertical double edges at most n times, ending up with an overall complexity of $O(n^2)$.

Procedure merge

begin

if $VA_j \neq 1$ and $VB_j \neq 0 \ \forall j$

- set $A_j = 2$ starting with arg min $_{j \leq n}\{A_j, B_j\} \ni VA_j = 2$ or $VB_j = 2$;
- until arg max $_{j \leq n}\{A_j, B_j\} \ni VA_j = 2$ or $VB_j = 2$;

endif

else

- j = 1;

while j ≤ n **do**

if $AB_j = (2,1)$ or $AB_j = (2,0)$

- delete the two edges in A_j and set $A_j = 0$;

endif

else if $AB_j = (1,2)$ or $AB_j = (0,2)$

- delete the two edges in B_j and set $B_j = 0$;

endif

else if $AB_j = (1,1)$ or $AB_j = (2,2)$

- delete all the edges in A_j and B_j, set $A_j = B_j = 0$;

endif

- $j \leftarrow j+1$;

endwhile

- p = 1;

while $k \leq$ m **do**

- connect all unconnected nodes j with $VA_j = 2$ or $VB_j = 2$ to j′ at distance $|j'-j| = p$ with VA_j or $VB_j = 1$ or with $VA_{j'}$ or $VB_{j'} = 2$ connected by horizontal edges to a vertical edge;

endwhile

endif

end

Figure 4.5: Procedure *merge*

4.1.2 Procedure *reach*

The reach procedure is called when two partial solutions do not overlap, that is, they do not share a common vertex or edge. Unlike the merge procedure, we allow modification of the vertical edges in this situation. We have the following definition.

Definition 4.1 *A boundary of an upper (lower) block to the lower (upper) block is the set of downmost (upmost) vertices in each aisle. A boundary of a solution s is denoted by B_s. An extended boundary EB_s additionally consists of the first items on the edges corresponding to the aisles adjacent to the boundary vertices. A portion P_{v_i,v_j} of an extended boundary with end vertices v_i and v_j is defined such that deletion of the portion as a result of a 2-opt move with the other solution does not break the tour into more than one component.*

The idea is as follows: Without loss of generality, assume that the current solution procedure is bottom-up. For each vertex of the boundary of the upper solution, we consider extending 2 edges from this vertex until the end vertices of a portion in the lower solution, and delete the edge between the ends of the portion. In the case where an extension of the 2 edges creates 3 edges in total, we delete 2 of these as well. The vertex-portion pair that makes this move by increasing the travel distance minimally is selected and the reach move is made between this vertex-portion pair. In the case of ties, priority is given to the pair that deletes the longest edge from the portion. In Figure 4.6, the reach procedure is summarized.

The complexity of this procedure is $O(n^3)$, as the boundary of the upper solution can contain $O(n)$ number of vertices for the first for loop, and there can be $O(n^2)$ portions in the lower block for the second loop, ending up with an overall complexity of $O(n^3)$.

As an example, consider the subproblem in Figure 4.7. Here, B_{upper} consists of a single vertex, namely v_5. The boundary B_{lower} consists of m_{14}, m_{24}, m_{34}, v_3, m_{33}, m_{43}, m_{53}, v_4, m_{44}, m_{54}, m_{64} and m_{74}. The extended boundary also consists of v_1 and v_2. Here, the best reach move is found to be from v_5 to the ends of $P_{m_{34},m_{74}}$. The move results in the graph shown in Figure 4.8. Then, we delete a pair of edges from the ones occurring thrice in the intermediate solution, as well as deleting the path between m_{34} and m_{74}. At the end of this procedure, we get the solution in Figure 4.9.

```
begin
-   opt = ∞;
-   bv* = ∅;
-   v*_j = ∅;
-   v*_k = ∅;
for each vertex bv_i on the boundary of the upper solution do
        for each portion P_{v_j,v_k} of the lower solution do
                if w_(bv_i,v_j) + w_(bv_i,v_k) − w_(v_j,v_k) ≤ opt
-                       opt = w_(bv_i,v_j) + w_(bv_i,v_k) − w_(v_j,v_k);
-                       bv* = bv_i;
-                       v*_j = v_j;
-                       v*_k = v_k;
                endif
        endfor
endfor
-   Delete edge (v*_j, v*_k) and add edges (bv*, v*_j) and (bv*, v*_k);
end
```

Figure 4.6: Procedure *reach*

4.1.3 Procedure *invert*

The invert procedure aims to convert a problem (or subproblem) so that the top-down solution procedure can be carried out in the same manner as the bottom-up procedure. The process consists of redefining the locations of the items by taking their "mirror images" with respect to both the x- and y-axes. Let each item i be defined by its aisle number an_i and y-coordinate y_i. Then, in the inverted problem, the new aisle number is $n - an_i$, where n is the number of aisles and the new y-coordinate is $aisle_length * blocks - y_i$, where $aisle_length$ denotes the length of aisles and $blocks$ denotes the number of blocks. Figure 4.10 summarizes this procedure.

4.1.4 Procedure *improve*

Procedure improve is a 3-opt improvement procedure over the merge-and-reach heuristic. Here, the nodes corresponding to the items are considered for exchange. In each step, the

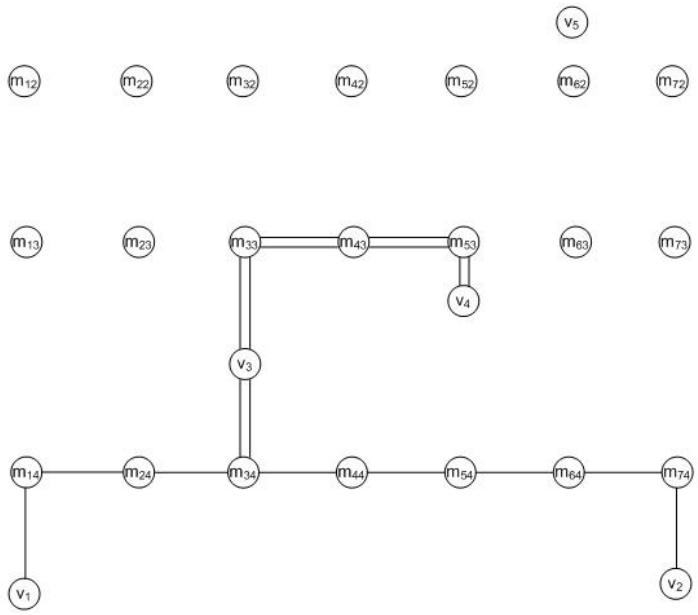

Figure 4.7: A pair of non-overlapping solutions

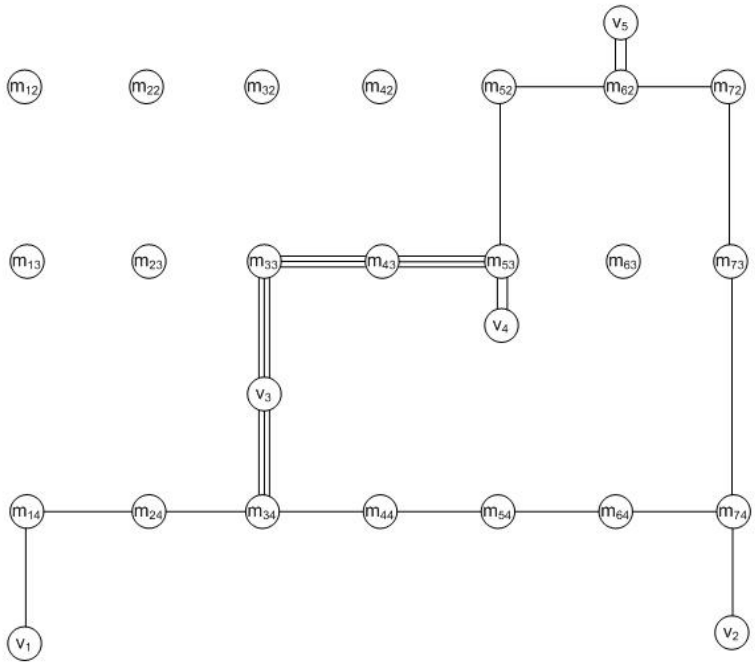

Figure 4.8: Intermediate solution to the subproblem in Figure 4.7 after the reach move

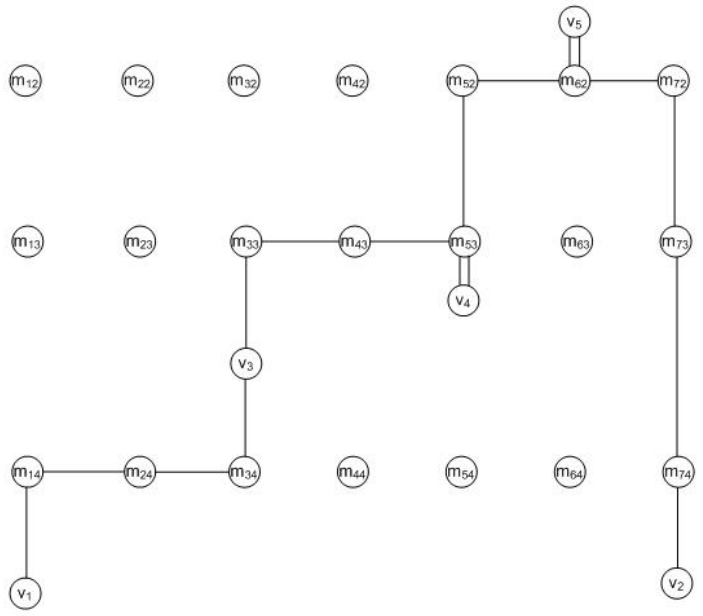

Figure 4.9: The resulting reach solution to the subproblem in Figure 4.7

begin

for each item i with an_i and y_i

- $an_i = n - an_i;$
- $y_i = aisle_length * blocks - y_i;$

endfor

end

Figure 4.10: Procedure *invert*

exchange with the largest improvement is made. The procedure is terminated when there is no further improvement. The improvement process together with the merge-and-reach heuristic is called the merge-and-reach$^+$ heuristic, and is discussed in Section 4.4. A summary of the improvement procedure is given in Figure 4.11.

The complexity of the improve procedure is $O(m^3)$, where m corresponds to the number of items. The complexity is determined by the number of possible ways to define 3-opt moves.

```
begin

-   Let T be the current tour;
-   For every item i, compute a set of nodes N(i);

while failure is not declared do

    for each item i do

-       Examine all possibilities to perform a 3-opt move which eliminates three edges having at least
        one endnode in N(i). If it is possible to decrease the tour length in this way, then choose the
        best of such 3-opt moves and update T;
-       Otherwise declare failure;

    endfor

endwhile

end
```

Figure 4.11: Procedure *improve*

4.2 The Merge-and-Reach Heuristic

Having defined the subroutines called throughout the heuristic procedure, the heuristic itself can now be described. Figure 4.12 summarizes the merge-and-reach heuristic.

In the initialization part, random cuts are generated to break the problem into subproblems. Following this, the first *while* loop solves the 2-OPP in each block using the polynomial time algorithm. The first of the two nested *while* loops controls the subproblems, whereas the second one joins the solutions to the blocks in each subproblem using both the top-down and bottom-up approaches and reporting the best solution for each subproblem. The last *while* loop joins the subproblems and ends up with a single solution for the problem.

The overall complexity is determined by the nested *while* loops. The first loop runs in $O(c)$ time, whereas the second one requires $O(kn^3)$ time: $O(n^3)$ coming from the reach procedure, which might be called k times in the worst case, making the overall complexity $O(ckn^3)$

4.3 Two Example Problems

In this section, we implement the merge-and-reach heuristic on two example problems. The first problem is the one illustrated in Figure 3.1, which is a 4-OPP with 8 aisles and 25 items.

```
begin
-   b = 1;   //block counter
-   Generate c − 1 distinct random cuts to form the subproblems. Let b_s denote the number of blocks
    in subproblem s;

while b ≤ k − 1 do

-   apply algorithm ratliff_rosenthal to the 2-OPP in block k;
-   b ← b + 1;

endwhile

- b = i = s = r =1;

while s ≤ c do
    while b ≤ b_s − 1 do

        if the b + i^th block is empty
-           i ← i + 1;
        endif

        else if the b^th and b + i^th solutions overlap
-           apply procedure merge to the solutions b and b + i;
-           i = 1;
        endif

        else if the the b^th and b + i^th solutions do not overlap
-           apply procedure reach to the solutions b and b + i;
-           i = 1;
        endif

-       b ← b + 1

        if b = b_s and r = 1
-           apply procedure invert to the subproblem s;
-           b = 1;
-           r = 2;
        endif

    endwhile
    s ← s + 1;

endwhile

s = r = 1;

while s ≤ c do

        if the s^th and s + 1^st solutions overlap
-           apply procedure merge to the solutions s and s + 1;
        endif

        else if the s^th and s + 1^st solutions do not overlap
-           apply procedure reach to the solutions s and s + 1;
-           i = 1;
        endif

-       s ← s + 1

        if s = c + 1 and r = 1
-           apply procedure invert to the whole problem;
-           s = 1;
-           r = 2;
        endif

endwhile

end
```

Figure 4.12: Heuristic *merge_and_reach*

This problem is a relatively "dense" problem, whose 2-OPP solutions tend to overlap and require the merge procedure to be joined. The second problem is a 9-OPP with 10 items, which indicates a relatively "loose" problem, with the requirement of the reach procedure to join the solutions. For the first problem, we will assume that no cuts are used. For the second problem, we will use a single cut to divide it into two subproblems.

The graphical representation of the first problem is shown in Figure 3.2. Figure 4.13 illustrates the 2-OPP solutions to each of the four blocks of the problem, whereas Figure 4.14 shows the resulting bottom-up solution, with a total travel distance of 172, and an optimality gap of 4.9%. With no cuts and only merge procedure required, the top-down solution also yields the same result as the bottom-up solution.

The second example problem, whose graphical representation is given in Figures 4.15 through 4.17. In Figure 4.15 is randomly cut from the middle aisle separating the 4th and 5th blocks. Figure 4.16 gives the resulting solutions for the two subproblems. Both of these solutions are obtained by the bottom-up procedures. As a last step, these two partial solutions have to be joined so that a single tour is obtained. Consequently, the route in Figure 4.17 is obtained. The resulting solution has a total travel distance of 207, which coincidentally is the optimal solution.

4.4 Improvement: The Merge-and-Reach$^+$ Heuristic

The improved version of merge-and-reach, called the *merge-and-reach*$^+$, applies a 3-opt improvement procedure over the merge-and-reach solution. The motivation behind this procedure is that the node sequence of the merge-and-reach solutions tend to resemble the optimal ones, with the exception of singular nodes or a sequence of nodes. Therefore in order to get closer to the optimal solution, it becomes important to move these nodes or sequences to their "closer to optimal" places on the route. Consequently, the procedure *improve* is called after execution of merge-and-reach in order to get the merge-and-reach$^+$ solution.

For the example given in Figure 3.1, the resulting merge-and-reach sequence is given as $0 \rightarrow 12 \rightarrow 11 \rightarrow 7 \rightarrow 10 \rightarrow 14 \rightarrow 17 \rightarrow 16 \rightarrow 13 \rightarrow 9 \rightarrow 8 \rightarrow 20 \rightarrow 18 \rightarrow 19 \rightarrow 24 \rightarrow 23 \rightarrow 22 \rightarrow 21 \rightarrow 25 \rightarrow 15 \rightarrow 6 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 0$. The resulting travel distance is 172. The first 3-opt iteration replaces node 7 between the nodes 8 and 20. This reduces the total travel

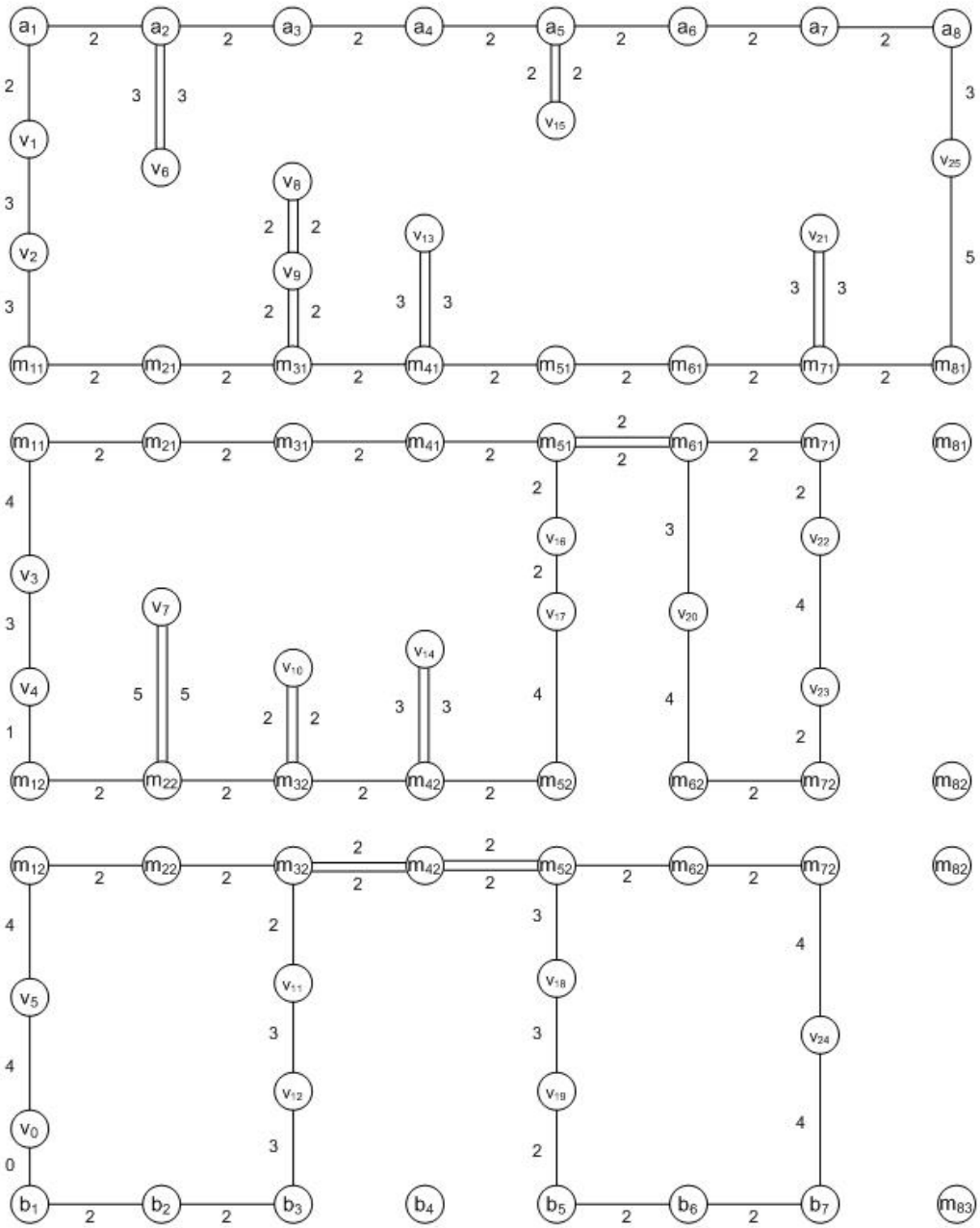

Figure 4.13: The set of 2-OPP solutions to each block of the problem in Figure 3.1

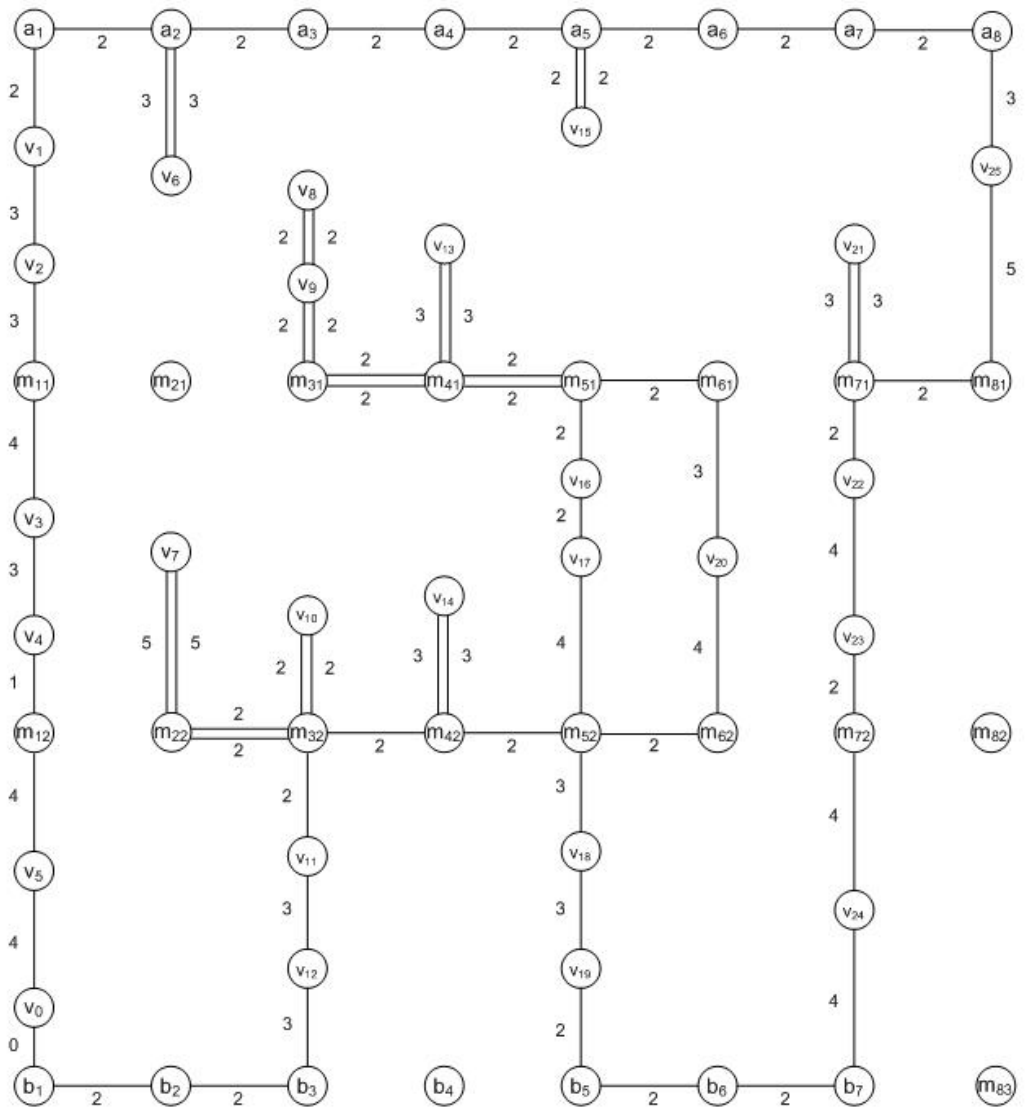

Figure 4.14: The merge-and-reach heuristic solution for the warehouse in Figure 3.1

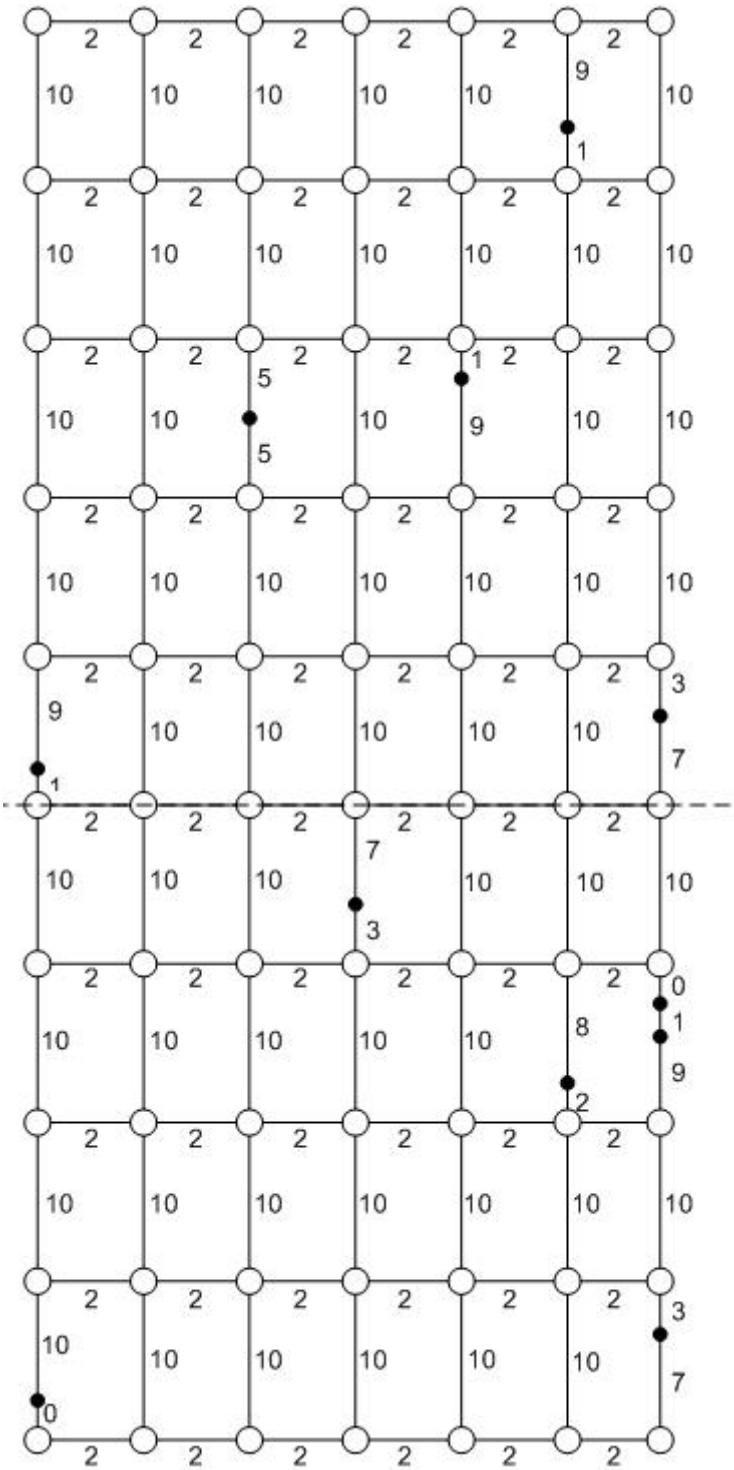

Figure 4.15: An example problem with 9 blocks, 7 aisles and 10 items

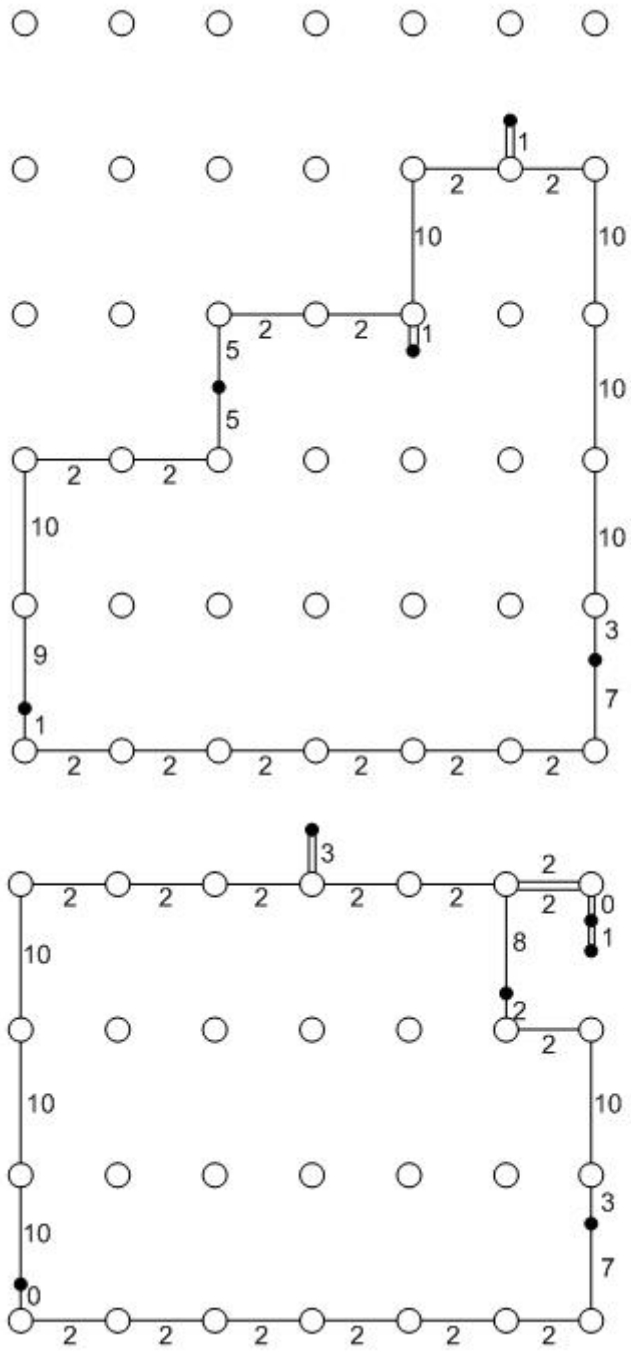

Figure 4.16: Intermediate solution to the problem in Figure 4.15

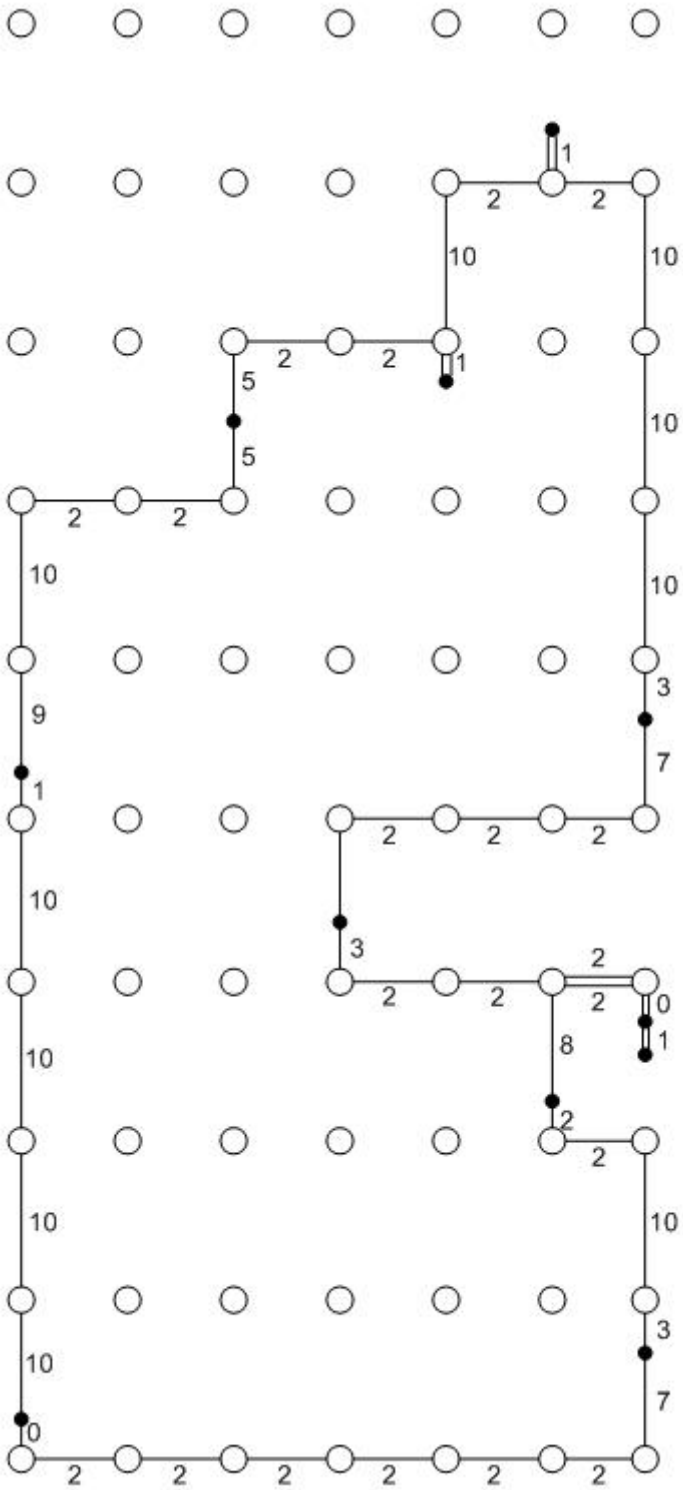

Figure 4.17: The merge-and-reach route for the problem in Figure 4.15, with a total length of 207 units

distance to 170. The second 3-opt iteration replaces the sequence $13 \rightarrow 9 \rightarrow 8 \rightarrow 7$ between the nodes 15 and 6, reducing the total distance to 164, which is the optimal solution.

4.5 Computational Experiments

This section includes the descriptions and results of various computational experiments in order to test the performance of the merge-and-reach and merge-and-reach$^+$ heuristic procedures and to compare them with their counterparts in the literature.

4.5.1 Test Instances

The problem settings used in the computational experiments are in line with those of Roodbergen and De Koster [73], where it is stated that in order to estimate the mean travel time within 1% relative error with a probability of 95%, a set of 2,000 instances for a setting is sufficient. In all of the settings, the length between the centers of two neighboring pick aisles is 2.5 m, the width of each cross aisle is 2.5 m, and the walking speed of each picker is assumed to be 0.6 m/s. The problem settings are given as follows. The number of pick items is set as 10 or 30, the length of the aisles can be 10 or 30 m, and the number of pick aisles is set as 7 or 15. Each setting is solved for the number of blocks varying from 1 to 10. For each cross aisle added in order to increase the number of blocks, the total vertical length of the warehouse is increased by the width of the cross aisle. Random storage of items is assumed and orders are generated according to uniform distribution throughout the warehouse. Note that the generated problem instances are different from those of [73] because their instances were generated on-the-fly and consequently, they are not available.

Roodbergen and De Koster [73] test the performances of their combined and combined$^+$ on the problem set as discussed above, and compare the results with those of the largest gap and S-shape heuristic as well as the aisle-by-aisle heuristic by Vaughan and Petersen [82]. The results are given in Tables C.1 and C.2 in Appendix C. Combined$^+$ heuristic gives the best performance in 74 of the 80 settings generated. Largest gap heuristic obtains better solution quality in 5 of the remaining 6 settings, whereas for the remaining setting, aisle-by-aisle performs the best. For the remainder of this chapter, we compare the performances of the merge-and-reach and merge-and-reach$^+$ heuristics with those of the combined$^+$ heuristic, as

it performs significantly better than its counterparts. However, it should also be noted that our heuristics outperform the largest gap, S-shape, aisle-by-aisle and combined heuristics in all of the problem settings. Therefore, whenever our heuristics perform better than combined$^+$, they obtain the best performance among all these heuristics.

4.5.2 Computational Results of the Suggested Heuristics

The heuristic procedures proposed here have been run on a computer with an AMD Turion 64x2 1.9 GHz processor and 1 GB RAM. Table 4.1 gives the average travel times of the merge-and-reach and merge-and-reach$^+$ heuristic solutions as well as the averages of the optimal travel times. Each instance is solved for 0, 1 and 2 random cuts, and the best solution is reported. It has been noted in [73] that the branch-and-bound algorithm takes a significant amount of computational time for finding the optimal solutions. For this reason, we use Concorde, a specialized TSP solver [38] to find the optimal solutions. As in the case of [73], the average run-time for each instance is less than 0.1 seconds for both heuristics. Additionally, the optimal solution finding procedure for a single instance also lasts less than 0.1 seconds on average.

There are several points to be noted on the performance of the merge-and-reach heuristic. First of all, the procedure (as well as the merge-and-reach$^+$ heuristic) yields the optimal solution when there is a single block. This is because of the fact that at the beginning of the process, each block is solved to optimality using the algorithm by Ratliff and Rosenthal [71]. Another aspect of the algorithm to be noted is that it is relatively robust when compared to its counterparts in the literature. In the best case, when there are 7 pick aisles, aisle length is 10, the number of items to be picked is 30, and there are 3 blocks, the percent deviation from optimal is 1.6%. In the worst case, when there are 15 pick aisles, aisle length is 30, with 10 pick items and 5 blocks, the percent deviation becomes 5.4%.

Table 4.1: Resulting average travel times (in seconds) by the merge-and-reach and merge-and-reach$^+$ heuristic procedures as well as the average optimal times for our problem set, with the best heuristic results indicated in bold (2,000 instances for each setting)

				Number of blocks									
Method	No. of aisles	Length	No. of items	1	2	3	4	5	6	7	8	9	10
Merge-and-reach	7	10	10	**139.8**	133.0	135.4	139.3	145.9	152.5	160.3	166.7	175.9	182.6
	7	10	30	**187.6**	196.3	203.0	213.6	223.3	232.1	242.6	250.7	259.7	268.0
	15	10	10	**223.3**	210.3	212.1	212.9	222.3	228.4	239.7	246.3	254.5	263.9
	15	10	30	**340.2**	324.4	322.6	330.6	339.9	349.7	362.0	372.9	384.4	398.6
	7	30	10	**269.5**	230.4	219.7	218.0	221.1	226.1	231.9	237.5	244.0	250.3
	7	30	30	**397.4**	369.9	353.2	347.8	346.8	348.6	348.0	353.2	355.9	360.0
	15	30	10	**379.9**	322.8	307.2	305.2	310.7	312.8	317.4	322.0	328.3	333.3
	15	30	30	**667.9**	562.6	521.6	506.9	501.1	498.5	503.5	508.4	513.2	517.3
Merge-and-reach$^+$	7	10	10	**139.8**	**131.4**	**134.2**	**138.0**	**144.7**	**150.6**	**157.9**	**163.9**	**173.1**	**179.6**
	7	10	30	**187.6**	**193.8**	**201.8**	**210.3**	**219.5**	**228.8**	**237.8**	**247.3**	**256.5**	**265.2**
	15	10	10	**223.3**	**208.3**	**207.7**	**208.9**	**217.6**	**222.3**	**233.7**	**240.0**	**248.6**	**258.1**
	15	10	30	**340.2**	**320.0**	**317.0**	**323.1**	**332.5**	**341.3**	**355.0**	**365.4**	**375.5**	**390.5**
	7	30	10	**269.5**	**226.4**	**214.7**	**213.5**	**216.1**	**220.2**	**225.8**	**231.4**	**237.5**	**244.0**
	7	30	30	**397.4**	**364.0**	**346.5**	**340.3**	**338.4**	**338.8**	**340.8**	**346.2**	**347.4**	**351.7**
	15	30	10	**379.9**	**315.5**	**299.8**	**295.8**	**300.6**	**303.5**	**309.7**	**312.4**	**318.8**	**324.8**
	15	30	30	**667.9**	**552.9**	**509.2**	**493.7**	**488.0**	**482.7**	**489.8**	**494.5**	**499.7**	**504.1**
Optimal	7	10	10	139.8	130.3	133.1	137.0	143.3	149.5	156.9	162.8	171.1	178.2
	7	10	30	187.6	192.1	199.8	208.5	217.2	225.6	236.0	244.1	253.5	262.3
	15	10	10	223.3	205.6	204.8	205.4	213.7	219.1	229.4	236.6	245.0	254.3
	15	10	30	340.2	316.9	313.2	319.0	327.0	336.2	348.7	359.1	371.4	385.6
	7	30	10	269.5	223.2	211.6	209.8	212.0	216.4	221.8	228.0	234.5	240.9
	7	30	30	397.4	359.8	341.8	335.0	332.6	333.3	333.5	339.7	343.2	347.6
	15	30	10	379.9	311.4	295.3	290.1	294.9	297.0	302.4	307.6	314.3	320.0
	15	30	30	667.9	546.9	501.2	485.5	478.1	475.6	481.0	486.7	492.2	496.7

Table 4.2: Percent deviations for the combined$^+$ heuristic by Roodbergen and De Koster [73], the merge-and-reach and merge-and-reach$^+$ heuristics, with the best heuristic performance indicated in bold (2,000 instances for each setting)

				Number of blocks									
Method	No. of aisles	Length	No. of items	1	2	3	4	5	6	7	8	9	10
Combined$^+$ [73]	7	10	10	7.1	2.7	3.6	3.3	3.0	2.8	2.5	2.4	2.2	2.0
	7	10	30	2.9	2.4	13.2	12.0	12.9	12.6	10.9	10.6	9.4	8.5
	15	10	10	7.1	2.5	6.5	7.6	7.7	7.4	6.8	6.6	6.1	5.5
	15	10	30	5.7	2.9	19.8	20.2	23.7	24.6	22.9	22.9	21.4	19.9
	7	30	10	13.0	5.6	3.9	2.8	2.2	1.9	1.8	1.5	1.4	1.4
	7	30	30	5.2	5.7	10.6	8.7	8.6	7.9	7.3	6.7	6.3	6.0
	15	30	10	13.2	4.9	4.9	4.5	4.1	3.7	3.6	3.4	3.2	3.2
	15	30	30	10.1	6.8	14.5	12.5	13.8	13.5	13.0	12.7	12.2	12.0
Merge-and-reach	7	10	10	**0.0**	2.1	1.7	1.7	1.8	2.0	2.2	2.4	2.8	2.5
	7	10	30	**0.0**	2.2	1.6	2.4	2.8	2.9	2.8	2.7	2.4	2.2
	15	10	10	**0.0**	2.3	3.6	3.7	4.0	4.2	4.5	4.1	3.9	3.8
	15	10	30	**0.0**	2.4	3.0	3.6	3.9	4.0	3.8	3.8	3.5	3.4
	7	30	10	**0.0**	3.2	3.8	3.9	4.3	4.5	4.6	4.2	4.1	3.9
	7	30	30	**0.0**	2.8	3.3	3.8	4.3	4.6	4.3	4.0	3.7	3.6
	15	30	10	**0.0**	3.7	4.0	5.2	5.4	5.3	5.0	4.7	4.5	4.2
	15	30	30	**0.0**	2.9	4.1	4.4	4.8	4.8	4.7	4.5	4.3	4.1
Merge-and-reach$^+$	7	10	10	**0.0**	**0.8**	**0.8**	**0.7**	**1.0**	**0.7**	**0.7**	**0.7**	**1.0**	**0.8**
	7	10	30	**0.0**	**0.9**	**1.0**	**0.8**	**1.1**	**1.4**	**0.8**	**1.3**	**1.2**	**1.1**
	15	10	10	**0.0**	**1.3**	**1.4**	**1.7**	**1.8**	**1.5**	**1.9**	**1.4**	**1.5**	**1.5**
	15	10	30	**0.0**	**1.0**	**1.2**	**1.3**	**1.7**	**1.5**	**1.8**	**1.7**	**1.1**	**1.3**
	7	30	10	**0.0**	**1.4**	**1.5**	**1.8**	**1.9**	**1.8**	**1.8**	**1.5**	**1.3**	**1.3**
	7	30	30	**0.0**	**1.2**	**1.4**	**1.6**	**1.7**	**1.6**	**2.2**	**1.9**	**1.2**	**1.2**
	15	30	10	**0.0**	**1.3**	**1.5**	**1.9**	**1.9**	**2.2**	**2.4**	**1.5**	**1.4**	**1.5**
	15	30	30	**0.0**	**1.1**	**1.6**	**1.7**	**2.1**	**1.5**	**1.8**	**1.6**	**1.5**	**1.5**

Table 4.3: The number of times each cut procedure gives the best solution for our problem set, bold entries indicating statistical significance of that cut value on the performance of the heuristic, $\alpha = 0.99$ (2,000 instances for each setting)

			Number of blocks										
			3		4			5			6		
No. of aisles	Length	No. of items	$c = 0$	$c = 1$	$c = 0$	$c = 1$	$c = 2$	$c = 0$	$c = 1$	$c = 2$	$c = 0$	$c = 1$	$c = 2$
7	10	10	**918**	**1,072**	323	**821**	**856**	341	**781**	**878**	332	**751**	**917**
7	10	30	623	**1,377**	510	**778**	**712**	366	**843**	**791**	382	**756**	**862**
15	10	10	561	**1,439**	194	782	**1,024**	326	744	**930**	256	**835**	**909**
15	10	30	701	**1,299**	482	**715**	**803**	408	**768**	**824**	290	**834**	**876**
7	30	10	589	**1,411**	336	678	**986**	330	765	**905**	234	721	**1,045**
7	30	30	661	**1,339**	464	**746**	**790**	472	**742**	**786**	330	**776**	**894**
15	30	10	422	**1,578**	236	711	**1,053**	200	792	**1,008**	144	767	**1,089**
15	30	30	498	**1,502**	305	**803**	**892**	252	**837**	**911**	198	**856**	**946**
Averages			621.6	1,377.1	356.3	754.3	889.5	336.9	784.0	879.1	270.8	787.0	942.3

			Number of blocks											
			7			8			9			10		
No. of aisles	Length	No. of items	$c = 0$	$c = 1$	$c = 2$	$c = 0$	$c = 1$	$c = 2$	$c = 0$	$c = 1$	$c = 2$	$c = 0$	$c = 1$	$c = 2$
7	10	10	277	**777**	**946**	200	**824**	**976**	131	**867**	**1,002**	101	**902**	**997**
7	10	30	346	**851**	**803**	285	**803**	**912**	217	**832**	**951**	181	**865**	**954**
15	10	10	236	**852**	**912**	181	**765**	**1,054**	126	**838**	**1,036**	78	**876**	**1,046**
15	10	30	256	**818**	**926**	249	**905**	**846**	201	**856**	**943**	155	**900**	**945**
7	30	10	186	747	**1,067**	153	792	**1,055**	99	799	**1,102**	89	822	**1,089**
7	30	30	289	**823**	**888**	222	**862**	**916**	202	**876**	**922**	193	**877**	**930**
15	30	10	129	771	**1,100**	111	793	**1,096**	93	793	**1,114**	52	843	**1,105**
15	30	30	190	**929**	**881**	166	**878**	**956**	143	**954**	**903**	116	**886**	**998**
Averages			238.6	821.0	940.4	195.9	827.8	976.4	151.5	851.9	996.6	120.6	871.4	1,008.0

When the number of blocks is increased, the quality of the solution tends to decrease to some extent. This can be attributed to the following: In dense problems (such as those with fewer blocks), the merge procedure finds good solutions; whereas for sparse problems (such as the ones with many blocks), it is the good performance of the reach procedure that leads to better solutions. When the number of blocks is at a mediocre level so that the merge and reach procedures must work together, the quality of the solution diminishes to a certain extent, although not very significantly. Furthermore, the merge-and-reach heuristic tends to find marginally better solutions when the number of items is larger, aisle length is shorter and the number of aisles is smaller. This is due to the slightly better performance of the merge procedure over the reach procedure.

In order to investigate the effect of the number of cuts on the performance of the merge-and-reach heuristic, the results in Table 4.3 are provided. The first thing to be noted here is that when the number of blocks is 1 or 2, the application of cuts is not possible, as there is no middle aisle to cut for a single block, and cutting from the middle aisle in the 3-OPP does not change the solution procedure. Additionally, when there are 3 blocks, the application of at most a single cut is possible, due to the same reasons. The table gives the number of times each cut gives the best solution out of the 2,000 problem instances for each setting. The effect of the number of cuts on the performance of the heuristic procedure has been examined using Wilcoxon Signed Rank Test with $\alpha = 0.99$, and the number of cuts with significant effect on performance have been shown in bold in Table 4.3.

Table 4.3 indicates that of all the cases given, using no cuts significantly affects the performance of the heuristic procedure for only one setting (7 aisles, 10 items, aisle length 10 meters and 3 blocks). This justifies the usage of cuts, instead of simply solving the whole problem using the bottom-up and top-down approaches. Furthermore, in every case it can be applied, usage of two cut aisles significantly improves the performance of the heuristic. Usage of the cuts performs significantly better than using one especially when the problems are "loose", that is, the items are farther apart from each other. In all other cases, the application of 1 and 2 cuts together has significant effect on the performance. This is due to the fact that applying both cuts means applying two solution procedures.

In order to justify the usage of the 3-opt procedure as the improvement step for the merge-and-reach$^+$ heuristic instead of a simpler procedure like the 2-opt, the performance of the 2-opt

heuristic has been tested in preliminary runs, where 200 random instances from each problem setting were solved. Table 4.4 shows the results. The first part shows deviations of the 2-opt improvement procedure from the optimal, and the second part shows the percent improvement of the 2-opt scheme on the merge-and-reach results. When the deviation values are compared to those of the merge-and-reach procedure, it can be observed that the improvement is about 0.1% on average. This is due to the high disruption caused by the 2-opt procedure, where the structure of the route is significantly altered by each 2-opt move. Hence for the merge-and-reach$^+$ procedure, 3-opt moves are used as improvement moves.

Table 4.2 gives the performance of the combined$^+$, merge-and-reach and merge-and-reach$^+$ heuristics. When the performance of the merge-and-reach$^+$ procedure is analyzed, it is clearly observed that the performance of merge-and-reach is significantly improved by the 3-opt procedure. In the worst case, where there are 15 aisles, aisle length is 30, the number of items is 10 and the number of blocks is 7, the average deviation from optimal is 2.4%. In the best case, this deviation can be as low as 0.7%, excluding the 2-OPP, where optimal solutions are found. Hence the improvement procedure not only increases the performance of the heuristic, but it also increases the robustness of merge-and-reach, that is, the effect of problem parameters on the heuristic performance is insignificant. The optimality gaps for the combined$^+$ heuristic can be quite different under different settings of the problem parameters. In some cases, especially when the number of items is low, aisle length is high and the number of aisles is low, the optimality gap can get as low as 1.4%. However, when the number of aisles increases, the aisle length decreases and the number of pick items increases, the deviation from optimal can increase up to 24.6%.

When the performance of merge-and-reach$^+$ is compared to that of combined$^+$, merge-and-reach$^+$ is seen to dominate the combined$^+$ heuristic in all of the 80 problem settings. The closest gap between the procedures, which is within a thousandth, occurs when there are 7 aisles, aisle length is 30, number of items is 10 and there are 10 blocks. This coincides with the best performance of the combined$^+$ heuristic. Due to the robustness of the merge-and-reach$^+$ heuristic, the relative performance of merge-and-reach$^+$ over combined$^+$ is high when the performance of combined$^+$ is low, which especially occurs in "dense" problems, as discussed before. In "sparse" problems, where combined$^+$ works well, the performances get closer to each other.

Table 4.4: Percent deviations of the 2-opt improvement procedure from the optimal travel times, and the percent improvement of the 2-opt scheme over the merge-and-reach solutions for the preliminary runs (200 instances for each setting)

				Number of blocks									
Method	No. of aisles	Length	No. of items	1	2	3	4	5	6	7	8	9	10
Deviation (%) for 2-opt from optimal	7	10	10	0.0	2.0	1.6	1.6	1.7	2.0	2.1	2.3	2.8	2.4
	7	10	30	0.0	2.1	1.5	2.4	2.8	2.8	2.8	2.6	2.4	2.1
	15	10	10	0.0	2.2	3.5	3.6	4.0	4.2	4.4	4.1	3.8	3.7
	15	10	30	0.0	2.3	3.0	3.6	3.9	3.9	3.8	3.8	3.4	3.4
	7	30	10	0.0	3.1	3.8	3.9	4.2	4.4	4.5	4.1	4.0	3.8
	7	30	30	0.0	2.8	3.2	3.7	4.2	4.6	4.3	3.9	3.6	3.5
	15	30	10	0.0	3.6	4.0	5.1	5.3	5.3	4.9	4.6	4.4	4.1
	15	30	30	0.0	2.8	4.1	4.3	4.8	4.8	4.6	4.4	4.2	4.1
Improvement (%) of 2-opt on merge-and-reach	7	10	10	0.0	0.0	0.1	0.1	0.1	0.0	0.1	0.1	0.0	0.1
	7	10	30	0.0	0.1	0.1	0.0	0.0	0.1	0.0	0.1	0.0	0.1
	15	10	10	0.0	0.1	0.1	0.1	0.0	0.0	0.1	0.0	0.1	0.1
	15	10	30	0.0	0.1	0.0	0.0	0.1	0.0	0.0	0.1	0.1	0.0
	7	30	10	0.0	0.1	0.1	0.1	0.2	0.1	0.1	0.1	0.0	0.1
	7	30	30	0.0	0.0	0.1	0.1	0.1	0.1	0.0	0.1	0.1	0.1
	15	30	10	0.0	0.1	0.1	0.1	0.2	0.1	0.0	0.1	0.0	0.0
	15	30	30	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.1	0.0	0.1

CHAPTER 5

THE ORDER-PICKING PROBLEM IN A PARALLEL-AISLE WAREHOUSE: MULTIPLE-PICKER CASE

In this chapter, we mainly discuss the multiple-picker version of the order-picking problem discussed in Chapter 4, which we denote as k-OPP(s), where s represents the number of pickers. These types of problems mostly arise in situations where an order is too large to be picked by a single picker. Usage of multiple pickers can also be motivated by the warehouse being too large for a single picker to complete the picking of an order (as in the example given in Figure 4.15) or when the warehouse is divided into zones. In any case, the picked items have to go through an accumulation process when picking is completed. It may be common for warehouses employing multiple pickers to have multiple number of depot points, which is also the case throughout this chapter. As in Chapter 4, we assume manual off-the-shelf order-picking.

The problems we deal with in this chapter can be distinguished into three cases. In the first case, there is a capacity limit for each picker on the amount of items that can be picked in a single sequence. For this case, we develop a cluster-first, route-second evolutionary algorithm. As a second case, the pickers have capacities on the time they can travel in a single picking sequence. This is a common case especially when there are due times on the deliveries of the orders to the customers. This can also happen in storage areas that feed the production lines. In these cases, there are cycle times of the production line, which limit the total time to pick the orders. In these cases, employing a single picker may not be enough to satisfy the cycle time constraint. We develop a route-first, cluster-second evolutionary approach for tackling such problems. The last case includes capacities on both the loads and the travel times of the pickers. We use a combined approach using the cluster-first, route-second and route-first,

cluster-second approaches developed for the first two cases for these types of problems.

The outline of this chapter is as follows: Section 5.1 is devoted to the relevant literature on the algorithmic approaches to the order-picking problem with multiple pickers, as well as relevant problems. In Section 5.1.1, we discuss the relevant literature on k-OPP(s), whereas in Sections 5.1.2 and 5.1.3, we make references to the relevant literature on the evolutionary algorithms for the Vehicle Routing Problem (VRP) and the Steiner Tree Problem (STP). Section 5.2 puts forward an evolutionary approach for k-OPP(s) with load capacities and limits on travel time. The algorithm makes use of two different evolutionary approaches. In Section 5.2.1, a cluster-first, route-second evolutionary algorithm for the case with load capacities is given, whereas in Section 5.2.2 a route-first, cluster-second evolutionary algorithm for the case with time capacities is discussed. Section 5.2.3 gives the parameter settings that are used in both of the evolutionary approaches. The steps and procedures of the evolutionary algorithm are explained Section 5.2.4, and computational results are provided in Sectioncomputational.

5.1 Literature Survey

The literature on the order-picking problem with multiple pickers is very limited when compared to the single-picker case. The first part of this section deals with the studies on k-OPP(s). It has been mentioned in Chapter 3 that the OPP resembles a special type of the Traveling Salesman Problem, called the Steiner Traveling Salesman Problem. Analogously, the multiple-picker case resembles the Vehicle Routing Problem (VRP), with the relaxation that each node can be visited more than once and there are nodes that do not have to be visited. As we will discuss evolutionary algorithms for the solution of various versions of k-OPP(s) in the following sections, we also review the literature on the evolutionary algorithms for VRP. It has also been noted in Chapter 3 that there is a strong connection between the Steiner Tree Problem (STP) and the OPP. As a consequence, we review the literature on the evolutionary algorithms for STP as well.

5.1.1 Literature on k-OPP(s)

The only algorithmic study on k-OPP(s), to the best of our knowledge, is due to Geng et al. [37], who name the problem as the *capacitated warehouse routing problem*. It is assumed that

a single depot exists, where s pickers have to start their routes from. After completing their routes, they return to the depot. The problem is shown to be strongly NP-hard, by reduction to the strongly NP-hard 3-partitioning problem.

Geng et al. [37] propose a neighborhood search approach, called the *very large-scale neighborhood search* (VLSN) approach, to the problem. For a problem with m items, a neighborhood structure is said to be *very large* if the size of the neighborhood increases exponentially as m increases, or if the neighborhood is too large to search in practice, even if the increase is polynomial (e.g. $O(m^3)$ with m greater than a million).

The neighborhood of a solution is defined as the exchange of a single item to be picked by a picker to another picker. Since the neighborhood size is too large to explore completely, a heuristic method to explore a sufficiently high portion of the neighborhood is proposed. There are two possibilities of exchanges in the neighborhood: A *path exchange* considers the exchange of the item from picker i to $i+1$, from $i+1$ to $i+2, \ldots$, and from $i-2$ to $i-1$. A *cyclic exchange* also considers the exchange from $i-1$ to i. The idea of an efficient search of the neighborhood lies in the notion of an *improvement graph*. Given a solution graph $G(V, A)$ for the current solution, an improvement graph $G_1(V', A')$ is defined in a manner such that it is identical to G, only with the condition that the original each cyclic or path exchange in the current solution defines a cycle in G_1. Therefore in the improvement graph, each cycle corresponds to a move in the neighborhood. Since the total arc weight of each cycle determines the cost of making that move, we are interested in making a move if the corresponding cycle has a negative total arc weight (in which case the cycle is called to be *valid*). A hill-climbing procedure is proposed as a result, and is given in Figure 5.1.

The authors test their algorithm on randomly generated test problems. They generate *loose* problems, where the capacity of a picker is much larger than the number of pick-items, *tight* problems on the contrary; *uniform* problems, where the distribution of pick-items in the warehouse is uniform, *non-uniform* problems on the contrary; and additional *antiS* problems with items that fulfill exactly one picker, but located at farther aisles to accommodate neighborhood search opportunities. Their problem set contains problems with no or one middle aisle, aisle length as 50 meters, the number of aisles as 40 and vehicle capacity as 100 items. Since no other solution approach to the problem exists, the results obtained are compared with those obtained from a branch-and-price algorithm that is developed by the authors (yet not discussed)

```
Algorithm VLSN hill-climbing
begin

- obtain an initial solution;
- construct the improvement graph;

while there is a valid cycle in the improvement graph do

-       record the top 3 profitable cycles as candidates;
-       randomly choose one of the candidates;
-       update the TSP partitions along the cycle;
-       update the current improvement graph;

endwhile

end
```

Figure 5.1: Procedure *hill − climbing*

and guarantees an optimality gap of at least 1%, and a naïve simulated annealing algorithm with a 2-opt-like neighborhood definition. For about 60% of the cases, VLSN hill-climbing produces results within 1% of the branch-and-price algorithm and is found to be superior to the simulated annealing algorithm.

The VLSN hill climbing algorithm proposed in [37] is the only comparable heuristic procedure to the ones proposed in Section 5.2.1. However, the results given in the study are the ones associated with single experiments of each parameter (number of middle aisles, aisle length, number of aisles, vehicle capacity and number of pick-items). Owing to this, exactly the same problems in the set have to be solved to provide a statistically justifiable comparison. Unfortunately, the problem set is not available; hence no such comparison can be made.

To the best of our knowledge, no algorithmic study exists on the multiple-picker case with capacities on the time that the pickers travel nor on the case with limits on both time travelled and loads.

5.1.2 Literature on the Evolutionary Algorithms for VRP

The Vehicle Routing Problem (VRP), first put forward by Dantzig and Ramser [22], consists of finding a route for a fleet of capacitated vehicles starting from a depot such that all the demand of the customers is satisfied, demand of a single customer is served by a single vehicle, and a certain cost measure (the total distance travelled by the vehicles, the maximum distance

travelled by a single vehicle, etc.) is minimized.

The k-OPP(s) can be stated as a modified version of the VRP in the sense that the nodes of the corresponding graph can be travelled more than once and by more than a single vehicle (picker). Additionally, not all the nodes have to be visited on the corresponding graph. For the case discussed in this chapter, we also assume that multiple depot nodes exist. In any case, there is a strong resemblance between the VRP and the k-OPP(s), which makes it worthwhile to analyze the relevant literature on the evolutionary algorithms for VRP.

Gendreau et al. [35] provide the most up-to-date review of the literature on the metaheuristics for the VRP. Although the literature on the metaheuristic procedures for the vehicle routing problem is vast, the development of evolutionary algorithms for the VRP is rather new. Tabu search has been the dominant metaheuristic method for the problem and due to both the efficiency of the procedure and interest in its application to VRP; it has come up with very good results. For a more extensive review on the evolutionary algorithms on the VRP, the reader is referred to Demir [27].

Evolutionary algorithms have first been proposed by Holland [48], and the first study of an evolutionary algorithm for the capacitated VRP is suggested by Baker and Ayechew [6], whose main aim is to define a framework and a rather straightforward approach for an evolutionary algorithm to the VRP. The coding scheme consists of the index number of vehicle assigned to each item. The fitness value of the individual is determined by solving all the TSPs, each of which corresponds to the customers assigned to a single vehicle. As infeasibility is allowed, an *unfitness* measure is defined for each individual as the total amount of capacity (on the distance travelled by the vehicles) exceeded by the vehicles. Population size is 30 for small-sized problems and 50 for large-sized ones. Initial population is generated either randomly or using a structured approach. For the parent selection process, a tournament selection approach is applied. Two individuals are randomly selected from the population, and the better of these is selected as the first parent. The procedure is repeated for the second parent. For reproduction, a 2-point crossover is used. As a mutation operator, the swap of two genes is applied. A steady-state replacement scheme is employed, in which the two offspring replace the two most inferior solutions in the population. The algorithm terminates when a certain number of generations (depending on the problem size) is reached.

Berger and Barkaoui [10] use an ordered list of customers as the coding scheme, and use

the total travel distance as a fitness value. Infeasibility is not allowed, therefore whenever a crossover or mutation operation ends up with an infeasible individual, it is applied again until a feasible individual is obtained. Two populations are used, each of size 15, but 5 individuals from each population are allowed to migrate between these populations. Initial populations are generated using sequential insertion. Parent selection is made using a roulette-wheel based scheme, that is, the parents are selected in such a way that individuals with higher fitness values have higher probability of being selected. An insertion-based crossover operator and a neighborhood search-based mutation operator are employed. A steady-state replacement method with migration is applied for replacement. Stopping conditions are based on either number of generations or convergence (depending on problem size).

Jaszkiewicz and Kominek [50], as in Berger and Barkaoui [10], employ an ordered list of customers as an encoder, whereas the total distance travelled is used as the fitness function. Infeasibility is not allowed, and the population size is 50. Initial population consists of structured individuals, each generated using randomized heuristic procedure solutions. In each generation, parents are selected using a roulette-wheel based scheme and with probability 0.95, a distance preserving recombination operator is applied as a crossover operator. The authors propose no mutation operator. For replacing the parents, steady-state replacement method is used, and the stopping condition is based on the convergence criterion, in which the population is said to converge if all the individuals are identical.

Prins [69] takes a closer approach to the evolutionary algorithms for the TSP, and applies a route-first, cluster-second evolutionary approach to the VRP using the route obtained by the crossover and mutation operators. Therefore, the encoding of the chromosome consists of the ordered list of the customers resulting from the TSP tour. Fitness function is based on the total distance travelled by the vehicles, and is calculated using the approach due to Beasley [8]. The idea in this approach is to formulate a shortest-path problem for the initial tour in order to determine the assignments of the customers to the vehicles. Given a tour of the customers, the shortest-path graph is constructed in such a way that the order of the customers in the tour is preserved. The distance corresponding to the arc between two customers is calculated as the total distance required to move from the depot to the customer at the beginning of the arc, then travel the customers in the given tour until the customer at the end of the arc is reached. The distance to reach the depot from the latter customer is added to obtain the weight of the arc. The solution of the shortest path problem gives the fitness of the individual.

The algorithm does not allow infeasibility. For small-sized problems, a population size of 25 has been found sufficient, and for large-sized problems, the population size is increased to 50. Initial population is generated randomly and parent selection is made by binary tournament selection. The crossover operator is the widely-used order crossover (OX) operator in the evolutionary approaches for the TSP. The operator is given in Figure 5.2.

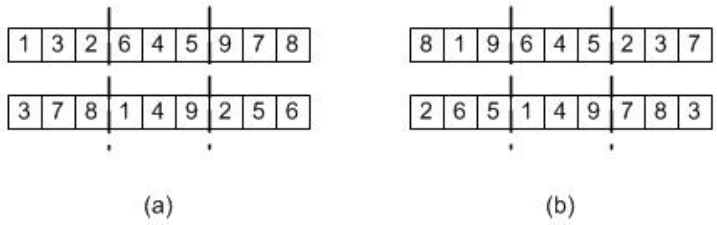

Figure 5.2: An example of order crossover (a) the parents and (b) the offspring

The order crossover operator works as follows: Given two individuals and two random cut points, the genes between these two cut points are preserved in the two offspring that correspond to each of the two parents. Starting with the next gene following the second cut point, the order in the other parent is tried to be preserved. For the example in Figure 5.2, the sequence (6,4,5) is preserved for the first parent. Then, starting with the 7th gene of the second parent, the customers are inserted to the first offspring. 2 is inserted without problem. The 8th and 9th genes, which are 5 and 6 are already used in the first offspring, so they are skipped. The insertion proceeds with the first gene of the second parent, which is 3. The remaining genes of the first offspring are inserted in this manner until all genes are filled. The same procedure is applied for the second offspring. The mutation operator in [69] depends on the swap of customers on the route. The replacement method is steady-state, and the stopping condition is based on the number of iterations.

Mester and Bräysy [64] use an *active-guided evolution strategy* in order to solve the VRP, and employ an evolutionary approach during the solution process. The coding scheme consists of the ordered list of customers for each vehicle. As in all studies discussed in this section, the fitness of each individual is based on the total distance travelled by the vehicles. Infeasibility is not allowed, and the population size is set as 100. The initial population is generated on a random basis. The parents are selected using the roulette-wheel method, and an insertion-

based crossover operator is employed. Steady-state replacement is applied and the algorithm stops after reaching a fixed number of iterations.

When the evolutionary algorithms for the VRP are investigated in terms of the parameter and algorithm settings they use, it can be observed that encoding consists either of an ordered list of customers, or the indices of vehicles assigned to the items. Fitness function consists of the total travel distance, the population size varies between 25 and 100, initial population is either generated randomly, or partially using heuristic procedures. Parent selection is either by tournament selection, or by roulette-wheel. Crossover operators widely vary, as do the mutation operators. An important note here is that all the studies use steady-state replacement, hence a crossover probability of 1. Mutation probability is less than 5%, and stopping conditions are based on convergence or number of iterations.

5.1.3 Literature on the Evolutionary Algorithms for the Steiner Tree Problem

The connection between the OPP and the STP has been discussed in Section 3.3.2. Because of the similarity of problems and their solution approaches, it will be of use to provide a brief review of the literature on the evolutionary approaches for solving the Steiner Tree Problem. As opposed to the evolutionary algorithms for the Vehicle Routing Problem, the Steiner Tree Problem has been more widely studied and solved using evolutionary algorithms. Due to this reason, it should be noted here that the review provided in this section is not exhaustive. The main aim, as in Section 5.1.2, is to investigate, and if possible, borrow the ideas used in solving the Steiner Tree Problems using evolutionary algorithms.

To the best of our knowledge, the first evolutionary algorithm for the Steiner Tree Problem has been put forward by Kapsalis et al. [54], who propose an algorithm that allows infeasibility, and uses different parameter settings for different instances of the problem. The study by Esbensen [29] is motivated by the study by Kapsalis et al. [54], and aims to come up with an algorithm that uses deterministic parameters for each problem instance tackled. In the first phase of the algorithm, the size of the graph is reduced using four different reduction techniques, due to Winter [83] and Winter and Smith [84] in sequence. The genotype consists of the x- and y- coordinates of the selected Steiner nodes in the solution and the fitness of each individual is determined using a distance network heuristic, which is based on the construction of a minimum spanning tree. Population size is 200, and the initial population consists of

solutions that include randomly selected Steiner points. Parent selection is roulette-wheel based. A special crossover operator is employed, in which one of the parents is chosen at random, the other is reordered so that it becomes homologous to the former one, and the standard 1-point crossover is applied. Mutation consists of simple change on a gene, and an invertion operator is also used for further exploration. Replacement is steady-state. The algorithm terminates when there is no improvement on the average and best fitness values for a certain number of iterations. Computational results reveal that the algorithm is superior to the one by Kapsalis et al. [54], and is compatible with the existing heuristics at the time.

The Steiner Tree Problem is widely considered in the area of electronics. As an application of evolutionary algorithms to these problems in this field, Haghighat et al. [44] use the idea of *Prüfer numbers* in order to encode the genotypes of the solutions. A Prüfer number $P(T)$ is defined as follows: Let i be the lowest numbered leaf in the graph and j be the predecessor of i. The Prüfer number is built up by appending j to the right of $P(T)$ and removing i and the edge (ij) from the graph, continuing in this manner until all nodes are considered. The edges that do not satisfy the bandwidth requirement are immediately removed from the graph in the preprocessing phase. For generating the initial population, a modified randomized depth-first search algorithm is applied. For this purpose, a linked list is constructed from the source node s to one of the destination nodes. Then, selecting one of the unvisited nodes at random, the algorithm proceeds until all the remaining nodes in the subtree is visited. The procedure is repeated for all subtrees. A special penalty function is used as the fitness function, and the parent selection is made by spinning a roulette-wheel. Two new crossover schemes are put forward, the first of which works using a local search algorithm that finds the bes improvement among the neigboring solutions, and the second of which repairs the two solutions generated by the traditional 1-point crossover. As in the crossover case, two operators are proposed for mutation: the first one randomly breaks the tree into subtrees and applies a repair algorithm to come up with a new tree, whereas the second one selects an infeasible chromosome among three classes, and repairs the chromosome. Mutation probability is set as 1%, steady-state replacement is used and the algorithm terminates on convergence.

A more recent study, specific to the case of the Rectilinear Steiner Tree Problem (RSTP), is due to Yang [86], and depends on the construction of minimum spanning trees. The algorithm is based on the theorem by Hanan [46], which states that for any RSTP instance, an optimal tree exists in which every Steiner node lies at the intersection of two orthogonal lines

that contain terminal nodes. The encoding of the individual consists of the Steiner nodes on Hanan's grid. The first gene stores the number of such Steiner nodes, and the remaining pairs represent the x- and y- coordinates of the Steiner nodes. The fitness function on such a set of Steiner nodes and the terminal nodes is calculated by solving the minimum spanning tree on these nodes using the algorithm by Prim [68]. The crossover operation simply consists of the random exchange of two Steiner points in the two parent individuals. The mutation operator randomly changes the coordinates of a number of Steiner points. As a further mutation operator, the insertion of new Steiner points to a randomly selected individual is considered. In this way, reachability of all the solutions in the solution space is guaranteed. For parent selection, the k-tournament selection method is used, in which k individuals are selected from the population, and pairwise comparisons are made between the selected individuals until two remain for the crossover. A population size of 200 is applied, and the initial population is generated using a randomized version of the minimum spanning tree algorithm. The crossover rate is given as 9%, the mutation probability as 1%, and the insertion probability as 31%. Replacement and termination strategies are not discussed.

The ideas used in the evolutionary algorithms for the STP are rarely directly applicable to the case of k-OPP(s) in terms of encoding, fitness function evaluation, mutation or crossover. All in all, they provide a framework especially for the mutation and crossover probabilities, parent selection, replacement and termination strategies.

5.2 An Evolutionary Algorithm for k-OPP(s) with Load Capacities and Limits on Time

It has been discussed in Section 5.1.1 that there exists a single algorithmic study on the k-OPP(s), which employs a very-large scale neighborhood search approach and assumes capacities only on the travel times of the pickers. Despite the quality of the solutions obtained (more than 60% of the solutions are within 1% of their corresponding optimal solutions), the computational time requirements are fairly large. The need to adapt a more comprehensive algorithm that also considers capacities on the loads that the pickers can carry and that incurs shorter computation times provides the motivation for the formulation of the evolutionary algorithm discussed in this section.

Before discussing the main aspects of the algorithm, it might be useful to state the main assumptions made throughout the section. It is first assumed that, as stated before, the warehouse can have multiple depot points. A picker can use any of these depot points as the starting point, but must return to the same depot point to complete the tour. As in the single picker case in Chapter 4, demands of unit loads are assumed.

The algorithm makes use of two well-known heuristic approaches to the VRP. The first one, called the cluster-first, route-second approach, first determines the assignments of the customers to the vehicles, then determines the route of each vehicle taking these assignments as given. The second approach, called the route-first, cluster-second approach, first comes up with a TSP tour for the whole customers, and then tries to break the tour into subtours so that the assignments of the customers to the vehicles are determined, hence the routes of the vehicles.

The idea used in the evolutionary approach is as follows. In each generation of the algorithm, one of the cluster-first, route-second or the route-first, cluster-second approaches is employed. Furthermore, the algorithm takes into account the specifics of the order-picking problem and determines the routes and/or assignments accordingly.

5.2.1 A Cluster-First, Route-Second Approach for the Case with Load Capacities

The main idea in the cluster-first, route-second approach is to determine the assignments of the items to the pickers using the crossover and mutation operators of the evolutionary algorithm, and to determine the routes of each picker using the merge-and-reach heuristic proposed in Chapter 4. Throughout the following of this section, the settings specific to the cluster-first, route-second approach are discussed.

Encoding Scheme

A chromosome used in the cluster-first, route-second approach consists of m genes, each representing the index number of the picker that is assigned to that item. Figure 5.3 shows an example chromosome for the case with 10 items.

The encoding scheme is valid as long as the demands are in terms of unit loads. When there is demand of more than a single load for the items, the encoding scheme must include the

Item No	1	2	3	4	5	6	7	8	9	10
Picker No	3	2	1	2	1	2	1	3	2	3

Figure 5.3: An example chromosome for the case with 10 items

amount of the item carried by each of the pickers. In the case of non-unit loads, the encoding scheme has to be modified, but the remaining aspects of the algorithm are still valid for the non-unit load assumption in that case.

Fitness Function

The fitness of each individual is given as the total travel time of all the pickers, as is the case in all the evolutionary approaches discussed in Section 5.1.2. The calculation of the fitness function for each individual is made using the merge-and-reach heuristic in the following manner: For each picker, the merge-and-reach heuristic calculates the travel time of the picker. The travel times obtained are then added up to obtain the total travel time of the pickers. Figure 5.4 summarizes the procedure *cfrs_fitness*.

```
begin

-   picker_index = 1;
-   fitness = 0;

while picker_index ≤ no_of_pickers do
-   Apply procedure merge_and_reach to the items assigned to picker_index along with the depot. Let
    the total distance be dist;
-   fitness ← fitness + dist;
endwhile

end
```

Figure 5.4: Procedure *cfrs_fitness*

Handling of Infeasibility

Since the encoding of the solution is made up of the assignments of the pickers to the items and since there are load capacities on the amounts of the items each picker can carry, there is a possibility that a given solution can be infeasible. In order to handle the possibility of infeasibility, we take the following two options.

1. Infeasibility is completely prohibited. In such a case, a repair procedure is applied to

each infeasible individual whenever encountered. Figure 5.5 summarizes procedure *repair*.

```
begin

-   picker_count = 0;

while ∃ picker_count ∋ excess_capacity_used[picker_count] > 0 do

-       for all items assigned to picker_count, try re-assignment to a different picker with
    excess_capacity_used [picker_count] > 0;
-       reassign item i to picker p with available capacity such that the cost of making the move is
    minimum;
-       excess_capacity[picker_count] ← excess_capacity[picker_count] – 1;

endwhile

end
```

Figure 5.5: Procedure *repair*

2. Infeasibility is allowed with penalty. Whenever this is valid, the encoding of the individual does not change, but the fitness value is calculated using procedure *repair* in Figure 5.5. This policy brings about the risk of cases where infeasibility is too high. In order to avoid going too deep into the infeasible region, we prohibit exceeding the capacities of the pickers by more than 50%. Whenever this amount is exceeded, the procedure *repair* is called with the capacity value 50% more than the original.

Crossover Operator

The main goal of the crossover operator is to come up with a new offspring that has an encoding which resembles those of its parents. Since the coding involves only assignments of the pickers to the items, we stick with the traditional crossover operators of 1-point, 2-point and uniform crossover, which are well applicable to this case. Another observation to make here is that since the items are not ordered in terms of their closeness to each other or the depot, the traditional crossover operators do not stand a great deal of difference among themselves. Based on this fact, we select the 2-point crossover operator to perform the crossovers and do not consider any other crossover type. An example 2-point crossover operation is shown in Figure 5.6.

As can also be observed from Figure 5.6, the 2-point crossover first generates two random cuts. Offspring 1 inherits the genes outside the cuts from Parent 1, and the remaining from

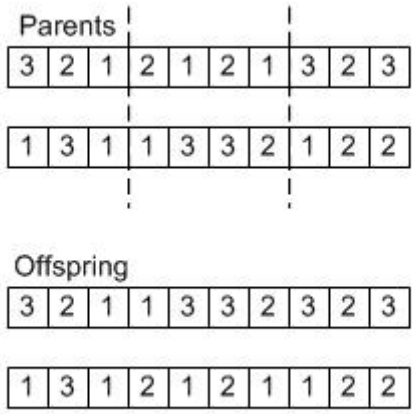

Figure 5.6: 2-point crossover between parents including 10 items

Parent 2. Offspring 2, on the other hand, inherits the genes outside the cuts from Parent 2, and the remaining from Parent 1.

Mutation Operator

The main aim of the mutation operator is to provide exploration opportunities and also to ensure reachability of the entire solution space. Actually, whenever infeasibility is not allowed, the repair operator, which allows a local search procedure to re-assign a picker to the items of the picker with excess capacity, in itself provides exploration opportunities. However, it does this in a more systematic manner, considering also exploitation of the solution space. As a mutation operator, two items are selected at random and their picker assignments are swapped. This operator, unlike the repair operation, does not consider changes in the fitness value resulting from this re-assignment and hence provides a better form of exploration. In Figure 5.7, an example swap operation is illustrated. In this example, the pickers assigned to items 3 and 10, which are 1 and 3 respectively, are swapped.

It might be useful to note here that reachability of solutions with the crossover and mutation operators is maintained in the procedure. The reachability condition for the evolutionary algorithm requires that every item can be assigned to every picker, which can be satisfied even by the mutation operator alone.

Offspring

3	2	1	1	3	3	2	3	2	3

Mutated Offspring

3	2	**3**	1	3	3	2	3	2	**1**

Figure 5.7: An example swap operation with 10 items

1 → 8 → 5 → 2 → 6 → 7 → 10 → 3 → 4 → 9 → 1

(a)

1	8	5	2	6	7	10	3	4	9

(b)

Figure 5.8: An example of the encoding scheme with 10 items: (a) the tour, (b) the corresponding individual

5.2.2 A Route-First, Cluster-Second Approach for the Case with Limits on Travel Time

The route-first, cluster-second evolutionary approach mainly aims to determine the main TSP route using the crossover and mutation operators of the evolutionary algorithm, and the assignment of pickers to the items is carried out using a modified approach of Beasley [8], who proposes a route-first, cluster-second approach for the Vehicle Routing Problem. The remaining of this section provides the settings specific to the route-first, cluster-second evolutionary approach.

Encoding Scheme

Since the aim of this approach is to come up with a TSP tour that provides a basis for the assignments of the pickers to the items, the encoding schemes specific to the TSP are applicable for encoding the individuals in this approach. Among various schemes such as adjacency representation, ordinal representation and path representation, which is the most natural representation of a tour. Simply put, the path representation gives the list of cities visited in the tour as a list, preserving the order in which they are visited. Figure 5.8 gives the path representation of an example tour with 10 items.

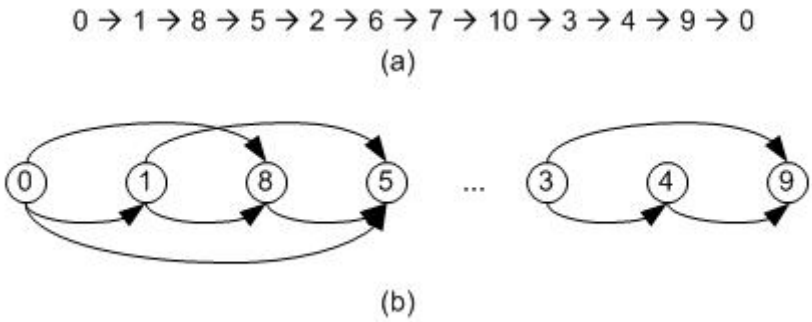

Figure 5.9: (a) An example tour with 10 items and the depot, (b) The corresponding order is represented as a graph whose shortest path problem solution gives the assignment of the items to the pickers and their picking orders

Contrary to the case in the cluster-first, route-second approach, the unit load assumption is not needed for such a representation scheme. This is due to the fact that the route-first, cluster-second approach does not consider the capacity on loads, therefore a single picker can pick up as many items as needed at a single visit to that item.

Fitness Function

As natural to the evolutionary algorithms for the routing problems, the fitness function of an individual is the total travel time (or distance) incurred by the pickers. In the route-first, cluster-second approach, once the main route is determined by the crossover and mutation operators, the fitness function then depends on the assignment of the items to the pickers. As mentioned before, the assignment is determined by a modified approach to the heuristic procedure by Beasley [8]. Before discussing the modifications to the algorithm, it is more useful to give the procedure due to Beasley [8].

The heuristic approach initializes by solving the TSP for the nodes of the graph that represents the problem. Then, the nodes are ordered in the same manner as they are in the given TSP tour, starting with the depot node to come up with a graph whose each arc represents the assignment of the ordered items to a picker. In Figure 5.9, such an ordering procedure and its graph representation are depicted for an example with 10 items and the depot.

Following the ordering procedure, the shortest path between the depot node and the last node in the graph is tried to be found. The weight c_{ij} on an arc (ij) is given by the total travel time required to start at the depot, visit the item i, then all the items in the TSP tour between i and

j, then item j and to return to the depot. The Shortest Path Problem is polynomially solvable, for instance with a dynamic programming approach. The number of arcs in the solution to the Shortest Path Problem gives the required number of pickers. Their corresponding routes are given by the order of items they visit on the graph for the Shortest Path Problem, starting with the depot node.

In order to improve the performance of the route-first, cluster-second heuristic, De Boer [23] proposes modification of the routes corresponding to the given assignments. This could be done using two methods. The first one finds the weight of each arc (ij) by solving the TSP corresponding to the depot node, nodes i and j, and the nodes between i and j on the TSP solution. The second method finds the optimal TSP tour for each picker, considering only the items assigned to that picker.

Our fitness function evaluation is similar to the procedure given by De Boer [23], with two modifications. First, the initial route is not found by solving a TSP on the VRP graph, but is rather determined by the encoding of the individual in consideration. Secondly, the weights of the arcs are not determined by solving the TSP on the subgraph corresponding to the items contained by the arc, but rather by applying the merge-and-reach heuristic approach, which is more efficient than solving the TSP optimally, and whose performance is near-optimal as discussed in Section 4.5.

Handling of Infeasibility

When the route-first, cluster-second approach is applied, as there are limits on the travel times of the pickers, the only possibility of infeasibility arises when the assignment of items to a picker brings about a required travel time which is more than the limit. In other words, this means that the weight of the arc is more than the time limit given. We simply reject such possibilities and impose that the weight of an arc is infinite when it exceeds the time limit.

Crossover Operator

Usually, the crossover operators for the encoding schemes specific to the TSP depend on which encoding scheme is used. For the path representation scheme, there are various crossover operators such as the partially mapped crossover (PMX), order crossover (OX) and cycle crossover (CX). In the route-first, cluster-second approach, we employ the nearest neighbor crossover (NNX), which is independent of the encoding scheme, and which ends up with a

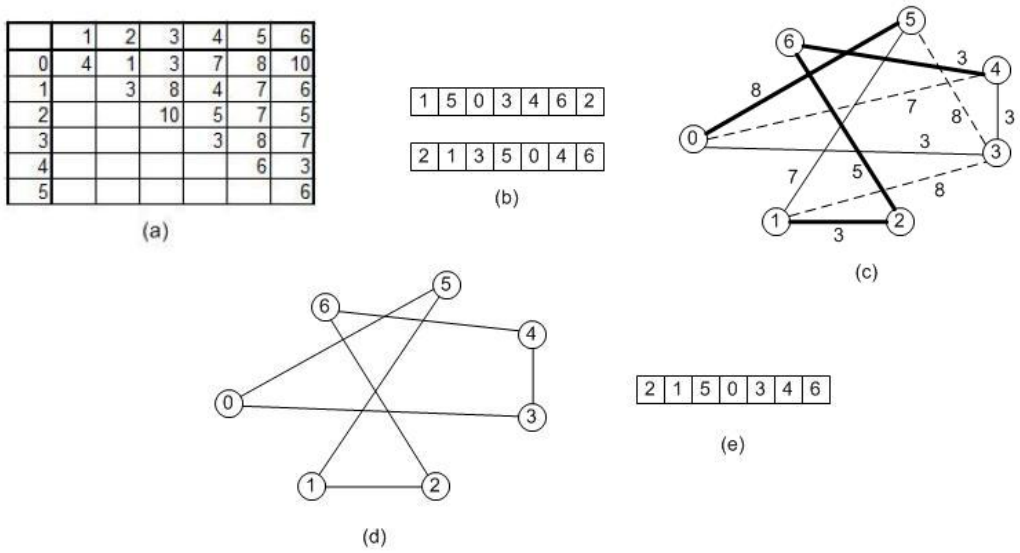

Figure 5.10: An example nearest neighbor crossover: (a) the distance matrix with the first row and column indicating the node index numbers (b) the two parents, (c) the union graph G, (d) the resulting tour, (e) the offspring

single offspring rather than two. It allows more than two parents to be used in the crossover operation, but here we stick with the traditional 2-parent approach.

NNX initializes by unifying the graphs corresponding to the tours given by the two parents, say G' and G'', into a single graph G. Then, it starts with an arbitrary node, say i, finds its nearest neighbor on G, say j, and adds edge (ij) to the solution. Then it looks for the nearest node k to j in G such that $(jk) \in G$ and addition of (jk) does not create a cycle. If such k exists, it becomes the next node for consideration and (jk) is added to the solution. If it does not, the nearest neighbor $l \notin G$ to j is found such that addition of (jl) does not create a cycle. The procedure proceeds in this manner until all nodes are in G. As a final step, the edge between the last and first nodes added to the solution is inserted and the tour is complete. The offspring is the path representation of the tour given by the solution. Figure 5.10 illustrates a case with 6 items and the depot, labeled as node 0.

Figure 5.10(a) tabulates the distance matrix for the example problem with seven nodes. In (b), the two parents selected for reproduction are given, with respective fitness values of 30 and 49. (c) illustrates the union graph G for the two parents. Arbitrarily starting with node 2, the nearest node to 2 in G is 1, hence (21) is added. The nearest node to 1 in G that does not create a cycle is 5, and (15) is added. The nearest nodes to 5 in G not creating a cycle

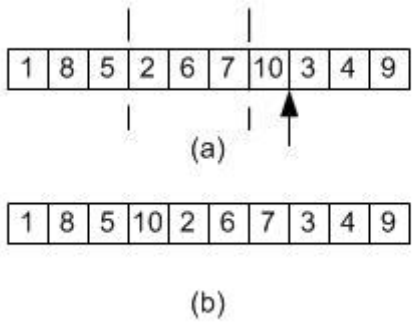

Figure 5.11: An example for the displacement operator: (a) the individual, (b) the mutated individual

are 0 and 3, selecting arbitrarily 0 as the next node, (50) is added. The nearest node to 0 in G that does not create a cycle is 3, and (03) is added. The nearest node to 3 in G not creating a cycle is 4, hence (34) is added. The nearest node to 4 in G that does not create a cycle is 6, and (46) is added. The tour is finally completed by adding the edge (62), so that a cyclic graph is obtained. The resulting tour is shown in (d), and the offspring, whose fitness is 32 is given in (e). The individual is not only better than one of the parents in terms of fitness, but it is also close in fitness to the other parent, and also its edges are completely inherited from at least one of the parents, which is a desirable feature of the crossover operators for TSP when the main concern is exploitation. The crossover operator also allows for the edges from the general graph to be added to the solution, which is desirable in terms of exploration of the solution space.

Mutation Operator

For the route-first, cluster-second approach, the displacement operator is selected as the mutation operator. The displacement operator simply selects a random portion of an individual and moves the portion to another randomly selected point in the individual. Figure 5.11 illustrates such an example.

In Figure 5.11(a), the sequence (2,6,7) is randomly selected to be replaced at the 7th position of the chromosome, which yields the mutated individual in (b).

5.2.3 General Parameter Settings

Having defined the parameter settings that are specific to each of the cluster-first, route-second and route-first, cluster-second approaches, the parameter settings that are common for both approaches can now be given. It should be noted here that these common settings are borrowed mostly from the literature on the evolutionary algorithms for the VRP in the literature, which has also been discussed in Section 5.1.2.

Population Size

In selecting the population size, we have gone through the literature survey on evolutionary algorithms for the VRP and have found out that the size ranges from 30 to 100. After our preliminary runs, small population size of 30 has ended up with premature convergence and high variations of solutions when the same problem was tried to be solved for a set of replications. The population sizes of 50 and 100 seemed to produce reasonable convergence and we have also decided to use an even higher size of 200 to provide better exploration. Hence the population sizes that were tried were 50, 100 and 200.

Generation of Initial Population

As can also be observed from the studies in Section 5.1.2, there are two main approaches to generate the initial population. The first approach is to generate the population randomly, whereas the second approach generates structured solutions as the initial population, usually by means of heuristic procedures. The advantage with the first approach is that the initial population tends to cover a more diverse region of the solution space, which allows better exploration. Furthermore, it takes less computational effort to generate the initial solutions randomly, when compared to generating structured solutions. However, it might take longer time for the algorithm to converge to a solution compared to the second approach. The advantage of the second approach is faster convergence, but with the higher risk of converging to a local optimum.

In our approach, we use both approaches for generating the initial population. The procedure is as follows: We generate solutions using the merge-and-reach heuristic (using all possible number of cuts and all possible upward and downward move combinations) until either all such solutions are generated or half of the population size is reached. Then, the remaining

solutions are generated at random.

Naturally, there are differences between the representations of the solutions for the cluster-first, route-second and route-first, cluster-second approaches. If the former approach is applied, the structured tours are represented by dividing the tour of the single picker into s subtours of equal size (in terms of the number of items), and a depot is assigned to each subtour in such a way that the assignment creates the cheapest insertion. If the latter approach is taken, the structured tours are represented by the given merge-and-reach tour. The random solutions are generated for the first approach by randomly assigning pickers to the items (and repairing if needed) and finding the subtours using the merge-and reach heuristic. For the second approach, the assignments are made using the approach in Section 5.2.2. The random solutions of the second approach are based on generating random routes and making the assignments using the modified shortest path algorithm by Beasley [8]. The fitness functions for the two approaches are then calculated accordingly.

Parent Selection

Basically, there are two main approaches for the selection of the parents for the reproduction process. The first of these is the roulette-wheel based method, in which the individuals are given weights that are determined by their fitness values, and the selection of the parents is made probabilistically, based on these weights: the higher the weight given to the individual, the higher the probability of selection. The second approach, called the *tournament selection*, generally selects two pairs of individuals from the population, applies pairwise comparison for both pairs and selects the better individuals for reproduction. The choice between the two methods again depends on the trade-off between exploration and exploitation. Solutions with better fitness function values generally have a higher probability of being selected with the roulette-wheel based method, which indicates that this method favors exploitation of the solution space. In tournament selection, a more random selection process is applied, therefore exploration is favored more.

As in the case of initial population generation, we try to find a compromise between the two approaches. Our parent selection approach is similar to the one applied by Yang [86], where the k-tournament selection approach is used. The approach, as discussed before, consists of selecting k individuals from the population and applying pairwise comparisons until two of them remain for reproduction. We have selected k to be eight for our approach. Since these

eight individuals are randomly selected from the population, this approach allows exploration to some extent. The pairwise comparisons allow for exploitation among the eight individuals, since better fitness values are favored.

Replacement Strategy

The replacement strategy used in the algorithm is the steady-state replacement strategy. This strategy has been found to be very successful when applied on the combinatorial problems, and is the only strategy used by the studies on developing evolutionary algorithms for the vehicle routing problem.

The replacement strategy to be used in the experiments is as follows: In each generation, only two parents are selected for reproduction. If 2-point crossover is applied, the two children replace the two worst individuals in the population. If NNX is applied, the offspring replaces the worst member of the population.

Crossover and Mutation Probabilities

Since the replacement system is steady-state, the crossover rate is selected to be 1, as the possibility of making no crossover does not create any change in the whole population in that iteration.

For the mutation probability, the two most widely used mutation probabilities in the literature on the evolutionary algorithms for the VRP, which are 1% and 5%, are tried.

Stopping Condition

The studies in the literature mainly use two different approaches for the stopping condition: the first one is based on the number of generations, where the algorithm is stopped when a certain number of reproductions are made; whereas the second one is based on convergence, where the algorithm stops when all (or a certain percentage) of the individuals in the population are identical. The first approach has the disadvantage of stopping before convergence, whereas the second approach has the risk of too long computational times, or not stopping at all due to lack of convergence.

For our parameter settings, the algorithm has never failed to converge to a solution. Therefore we have employed convergence as the stopping condition of the algorithm. In order to

avoid too long computational times while seeking complete convergence, we have set the convergence rate to be 95%.

5.2.4 The Algorithm

The evolutionary algorithm proposed here is a combined approach making use of both the cluster-first, route-second and route-first, cluster-second approaches. The algorithm basically proceeds in the following manner: During the generations, with probability α, the cluster-first, route-second approach is applied. With the remaining probability, the route-first, cluster-second approach is applied. Figure 5.12 illustrates the steps of the algorithm.

As can also be seen in Figure 5.12, the algorithm initializes with the generation of the initial population. Here, the following approach is used: If $\alpha \leq 50\%$, the initial population that is generated consists completely of cluster-first, route-second individuals. Otherwise, the population completely consists of route-first, cluster-second individuals. As the second step, in order to decide which of the evolutionary approaches to use in the current generation, a random number is generated. If the random number is less than α, the cluster-first, route-second approach is employed. Otherwise, the algorithm proceeds using the route-first, cluster-second approach in the current iteration. Depending on the evolutionary approach employed, the individuals may or may not be converted into the encoding scheme and fitness values of the current approach, if it is different from the approach used in the previous generation.

Once the evolutionary scheme is determined, the parents are selected for reproduction using the 8-tournament selection method discussed in the preceding section. The comparisons are made using the current fitness values and the parents are determined. Then, if the current scheme is cluster-first, route-second, 2-point crossover is applied to the parents, and two offspring are generated. The offspring replace the two worst individuals in the population. For the route-first, cluster-second approach, nearest neighbor crossover is applied to the parents to generate a single offspring. The offspring replaces the worst member of the population.

Following the crossover procedure, a new random number is generated in order to determine whether mutation is to be made. If the random number is greater than p_m, the mutation step is skipped and the algorithm proceeds to the convergence checking step. If the random number is less than p_m, swap mutation is applied if the current method is cluster-first, route-second

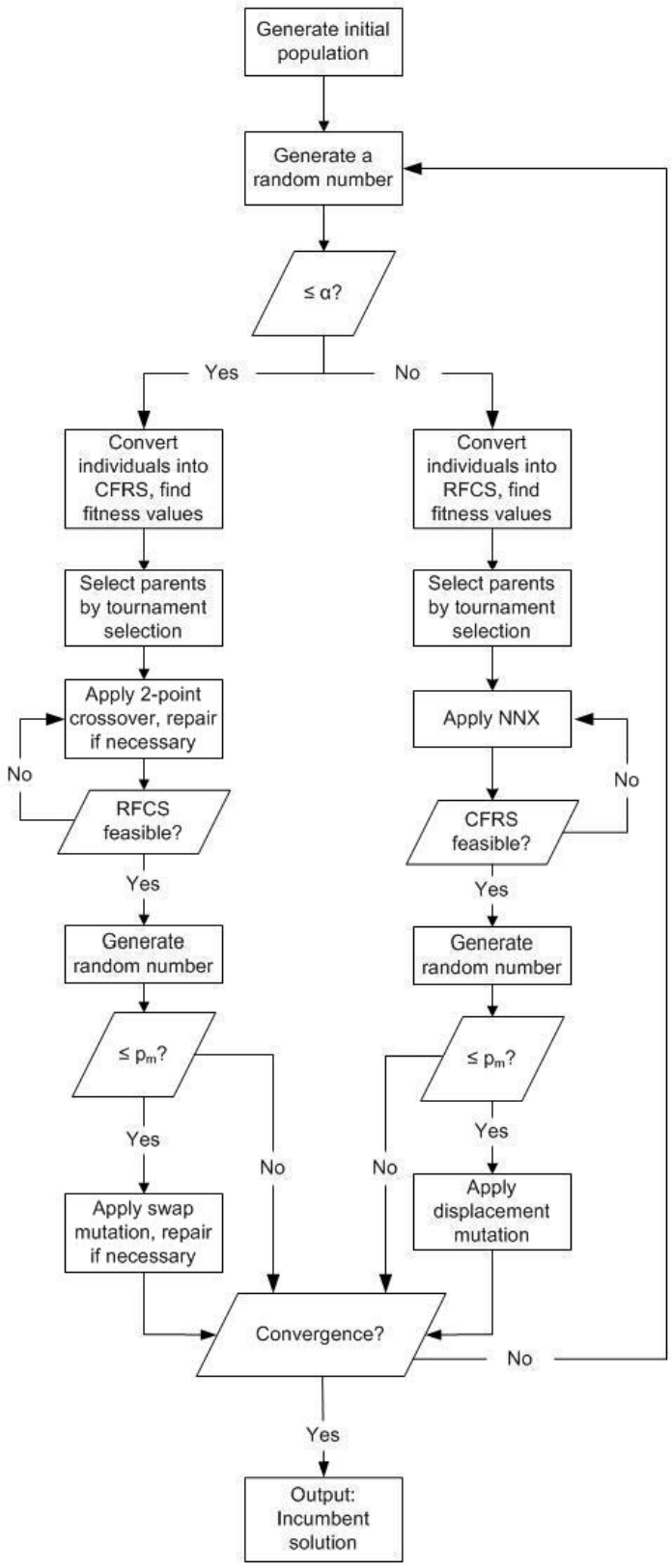

Figure 5.12: The steps of the combined evolutionary algorithm

and displacement mutation is applied if the current method is route-first, cluster-second.

Since the generation is complete after the crossover and (possibly) mutation steps, the algorithm has to check whether the stopping condition is reached by checking whether 95% of the individuals in the population are identical. If convergence holds, the algorithm is terminated and the incumbent solution throughout the procedure is given as the output. Otherwise, another random number is generated in order to determine the evolutionary method to be used in the next generation, and the algorithm proceeds in the manner discussed above.

5.2.5 Computational Experiments

As discussed in Section 5.1.1, the only algorithmic study comparable to the evolutionary approach proposed in this chapter is due to Geng et al. [37], which is a very large-scale neighborhood search algorithm. However, there are several drawbacks of the computational experiments in [37]: First of all, the problem instances are not available for common use; secondly, the results are given for single instances for each problem setting; thirdly, important details on the problem instances, such as how many pickers are employed for each case and whether the number of pickers are exogenously or endogenously determined are not specified; and lastly, the authors assume only limits on time, disregarding the load capacities of the pickers. Due to these reasons, the computational experiments have been made on a randomly generated set of problems.

Test Instances

The random problem set used for the computational experiments is a subset of the one given in Section 4.5. The subset consists of larger problems (with more items and aisle lengths) so that the usage of multiple pickers is justified. It consists of the number of blocks set as 1, 2, 4 and 8. As in Section 4.5, the number of blocks is increased for each problem instance by increasing the total vertical length of the warehouse by the width of the cross aisle. Therefore each instance with multiple blocks is obtained by adding cross aisles to the single-block instance. Number of aisles are set as 7 or 15, pick-aisle length is 30 m, number of items is 30, and the length between two neighboring aisles is 2.5 m, as is the width of the cross aisles. We again assume that the walking speed of each picker is 0.6 m/s. Random storage of items is assumed and orders are generated according to uniform distribution throughout the warehouse, as is

also the case in Section 4.5.

The number of pickers is set as 2 for a single block; 2 or 3 pickers are employed for the cases with 2 or 4 blocks; and 3 or 4 pickers are assumed for the 8-block case. As was stated before, we assume a multiple-depot warehouse for the k-OPP(s). In Figure 5.13, the locations of the depot points within the warehouse for each block setting are depicted.

As can be observed from Figure 5.13(a-d), when a single block exists, the depot points are at the left and right ends of the front cross aisle. With the case of 2 blocks, the depots are on both ends of the middle aisle. For the case with 4 blocks, the depots are on both ends of the middle-most middle aisle and for 8 blocks, the first 2 depots are on both ends of the 2nd middle aisle, whereas the remaining 2 depots are on both ends of the 6th middle aisle. In the cases with 2 depots, we assume that there are doors on both sides of the warehouse, one across the other, so that a vehicle, such as a truck, can enter from one door, collect the items in the depot on that side, and exit from the one across after collecting the items in the depot on this side. For the case with 8 blocks, we assume that the same case happens with 2 doors on each side of the warehouse, one across the other.

It is assumed that the pickers are homogeneous, that is, they have identical load capacities, and identical limits on their travel times. This is applicable especially when the pickers have identical picking carts, which is not an unusual case. The load capacities of the pickers are assumed to be tight, that is, the capacity of the pickers is calculated by the function $\left\lceil \frac{no_of_items}{no_of_pickers} \right\rceil$. In this case, the number of pickers is automatically imposed on the algorithm.

The situation is a little more difficult for the limit on travel time. The approach to this situation in the literature is using a multiplier larger than 1 for the single vehicle optimal and then dividing this value by the number of vehicles. Determination of the multiplier is the crucial part in this step. Setting it too low may end up with infeasibility, whereas setting it too high may lead the solution to end up with fewer pickers than desired. Following a set of preliminary runs, we have observed that a multiplier of 1.20 avoids both possibilities for our problem set. Consequently, we use the function $\frac{1.20 z^{MR}}{s}$ to evaluate the capacities for each picker in each problem instance. Here, z^{MR} corresponds to the merge-and-reach solution value to the single-picker version of the problem, which is already known from the computational experiments in Section 4.5.

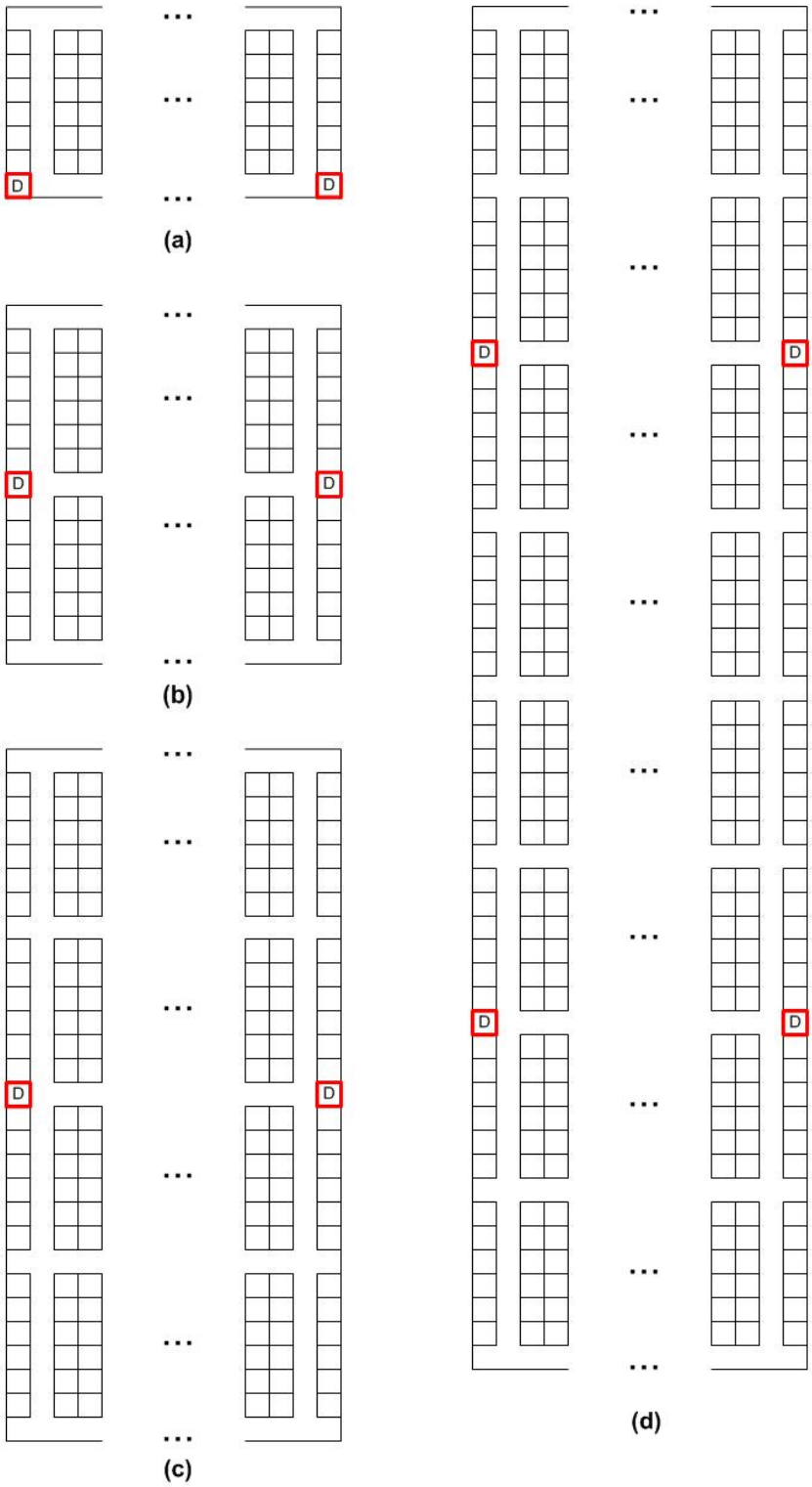

Figure 5.13: The depot points for each of the block settings: (a) single block, (b) 2 blocks, (c) 4 blocks, (d) 8 blocks

Implementation Issues

Having defined the problem parameters, it might be of use to recall the parameter settings of the algorithm: For the cluster-first, route-second approach, we try prohibition of infeasibility or penalizing it with penalty. Three different population sizes of 50, 100 and 200 are tested. Two possible probabilities of mutation are set as 1% and 5%.

For each problem setting, we solve 30 instances. The number of instances for each setting is lower compared to the single-picker problem set, due to the reasons on computational time. However, selection of 30 theoretically justifies the normality of the data due to the Central Limit Theorem, hence statistical comparisons are valid. Throughout the following of this section, the computational runs were made on a computer with an AMD Turion 64x2 1.9 GHz processor and 1 GB RAM.

The rest of this section is organized as follows: First, we try the pure cluster-first, route-second (α = 100%) and pure route-first, cluster-second (α = 0%) approaches in order to see their own performances. Then, we try the values α = 75% and α = 25% to see how different settings of the combined approach behave.

Computational Results with α = 100% and α = 0%

In this part, we aim to assess how the two evolutionary approaches individually behave to see how the cluster-first, route-second and route-first, cluster-second approaches perform on their own.

For the pure cluster-first, route-second approach (α = 100%), we assume for the time being that there are no limits on travel time. With the given alternative settings in the preceding part of this section, we solve 5,040 problem instances in total. These runs aim to serve the purpose of fine-tuning the cluster-first, route-second algorithm parameters. A Taguchi analysis is made on the results of the computational experiments, and the results are shown in Figure 5.14.

In Figure 5.14, factor A corresponds to the number of blocks and pickers, factor B represents the number of aisles, factor C corresponds to the population size, factor D represents infeasibility, and factor E corresponds to the mutation probability. It can be observed from Figure 5.14 that all the problem parameters (number of blocks, aisles and pickers) significantly affect the quality of the solutions. This is nothing but expected, and can also be observed

Results for: Worksheet 2

Taguchi Analysis: C6; C7; C8; C9; C10; C11; C12; C13; ... versus A; B; C; D; E

* NOTE * Design is not orthogonal.

Response Table for Signal to Noise Ratios
Smaller is better

Level	A	B	C	D	E
1	-54,34	-51,59	-52,59	-52,43	-52,48
2	-51,28	-53,59	-52,59	-52,76	-52,70
3	-55,13				
4	-52,39				
5	-52,25				
6	-50,54				
7	-52,22				
Delta	4,59	2,00	0,00	0,33	0,22
Rank	1	2	5	3	4

Response Table for Means

Level	A	B	C	D	E
1	537,4	386,3	433,5	425,3	428,1
2	365,9	488,9	441,7	450,0	447,1
3	570,3				
4	421,1				
5	418,1				
6	336,0				
7	414,5				
Delta	234,3	102,6	8,3	24,7	19,0
Rank	1	2	5	3	4

Response Table for Standard Deviations

Level	A	B	C	D	E
1	24,27	21,45	25,77	26,57	24,42
2	18,33	28,66	24,35	23,54	25,69
3	25,11				
4	26,58				
5	30,38				
6	18,95				
7	31,77				
Delta	13,44	7,21	1,42	3,03	1,27
Rank	1	2	4	3	5

Figure 5.14: Taguchi analysis results for $\alpha = 100\%$

for the single-picker case in Section 4.5. The figures also show that the effects of the algorithm parameters (population size, handling of infeasibility and probability of mutation) do not significantly affect the solution quality, when compared to the problem parameters. This is also expected, as the solution values depend mostly on the problem parameters. When the algorithm parameters are compared within themselves, it can be seen that the handling of infeasibility and mutation probability have a more significant effect than the size of the population.

Based on the fine-tuning experiments, we finalize the parameters of the cluster-first, route-second algorithm as follows: Population size is set at 200, infeasibility is allowed with penalty and mutation probability is determined as 5%.

Next, the pure route-first, cluster-second approach ($\alpha = 0\%$), as opposed to the previous case, it is assumed for the time being that pickers are uncapacitated in terms of the loads they can carry. Since there is no issue of infeasibility for this case, we solve half the number of instances of the cluster-first, route-second problem set, which gives 2,520 instances in total. As in the previous case, a Taguchi analysis is made on the results of the computational experiments, with the results shown in Figure 5.15.

In Figure 5.15, factor A corresponds to the number of blocks and pickers, factor B corresponds to the number of aisles, factor C represents population size, and factor D corresponds to the mutation probability. Figure 5.15 reveals that, as in the cluster-first, route-second case and as expected, the problem parameters (number of blocks, aisles and pickers) have significant effects on the solution quality. When the parameters related to the algorithm are considered, it is observed that the probability of mutation and population size have comparable effect on the solution quality. Hence we assume that they are both significant.

Since increasing the mutation probability and population size increases solution quality, we set the probability of mutation for the route-first, cluster-second setting at 5%, and the population size is set at 200.

In order to justify the usage of the two evolutionary algorithms, the results will also be compared with those of an exhaustive random search for each evolutionary approach. In each instance of the cluster-first, route-second case, $2xno_of_generations$ random assignments are generated and the routes of each solution are determined by the merge-and-reach heuristic

Taguchi Analysis: C5; C6; C7; C8; C9; C10; C11; C12; ... versus A; B; C; D

* NOTE * Design is not orthogonal.

Response Table for Signal to Noise Ratios
Smaller is better

Level	A	B	C	D
1	-53,97	-50,48	-51,96	-52,06
2	-52,55	-54,09	-52,61	-52,51
3	-52,98			
4	-51,77			
5	-51,61			
6	-51,63			
7	-51,49			
Delta	2,48	3,60	0,66	0,45
Rank	2	1	3	4

Response Table for Means

Level	A	B	C	D
1	513,3	335,0	399,5	408,8
2	432,4	508,8	444,4	435,1
3	456,8			
4	393,9			
5	386,4			
6	387,3			
7	383,4			
Delta	129,9	173,8	44,9	26,3
Rank	2	1	3	4

Delta	2,48	3,60	0,66	0,45
Rank	2	1	3	4

Response Table for Standard Deviations

Level	A	B	C	D
1	20,96	19,07	25,28	25,37
2	23,96	30,46	24,25	24,16
3	21,13			
4	26,40			
5	28,15			
6	26,50			
7	26,26			
Delta	7,19	11,39	1,03	1,21
Rank	2	1	4	3

Figure 5.15: Taguchi analysis results for $\alpha = 0\%$

procedure. In each instance of the route-first, cluster-second approach, $no_of_generations$ random tours are generated and this procedure is repeated 30 times in order to get more reasonable results for comparison. The assignments related to the tours are found by the assignment procedure given in Section 5.2.2. The main aim in these random search operations is to create as many individuals as generated by the evolutionary algorithms, although the second random search approach is favored more for comparability.

Table 5.1 compares the two pure approaches in terms of three different measures: The first one measures the deviation from the average single picker optimal. As there are multiple depot points, it may be the case that the k-OPP(s) solution is better than the single picker version, in which case the deviation becomes negative. A negative value of the deviation justifies the usage of multiple pickers in that case. The second measure is the improvement over the average random search solution. The larger the improvement, the more is the justification in using the two approaches over random search. The last measure indicates how much improvement has been made over the best solution in the initial population. If this value is large, the evolutionary algorithms are successful in improving the initial solutions generated by the merge-and-reach heuristic.

Before interpreting the results, it might be useful to keep in mind that the deviations and improvements given in Table 5.1 are not directly comparable. In terms of the first measure, the two problems are different from each other because the capacity constraints are not identical. For improvement over random search, an exact comparison is not possible because the random search procedures for the two cases are different (in addition to the difference in capacity constraints). All in all, the deviations and improvements more or less indicate the performance of the procedures to a certain extent.

Table 5.1 indicates a significant dominance of the route-first, cluster-second approach over the cluster-first, route-second procedure. For all settings, the former approach is better than the latter both in terms of the deviation from the single picker optima and improvement over the random search solutions. In only one case, where there is a single block with 7 aisles and 2 pickers, the improvement of the latter approach over the initial population best is better than that of the former. Yet in all the other cases, the route-first, cluster-second approach better improves the initial population best.

Table 5.1: Percent deviations from single picker optima and percent improvements over random search solutions and initial population incumbents for the cases $\alpha = 0\%$ and $\alpha = 100\%$ (averages of 30 instances), bold entries indicating superiority over the other method

			% Deviation from single picker optimal		% Improvement over random_search		% Improvement over init_pop_best	
Blocks	Pickers	Aisles	$\alpha = 0\%$	$\alpha = 100\%$	$\alpha = 0\%$	$\alpha = 100\%$	$\alpha = 0\%$	$\alpha = 100\%$
1	2	7	**-0.51**	-0.16	**35.4**	27.0	0.52	**1.11**
		15	**-2.85**	2.00	**42.8**	25.7	**2.97**	1.40
2	2	7	**-4.48**	0.34	**42.1**	28.8	**7.34**	3.50
		15	**-4.43**	2.08	**41.9**	25.7	**7.28**	1.31
	3	7	**-1.29**	2.22	**37.8**	23.2	**3.85**	1.38
		15	**-2.02**	1.74	**39.2**	26.7	**4.76**	1.76
4	2	7	**-4.83**	4.69	**42.2**	25.0	**9.04**	2.09
		15	**-6.09**	6.97	**44.9**	23.3	**11.37**	2.29
	3	7	**-6.78**	7.97	**46.0**	23.5	**11.43**	1.88
		15	**-6.76**	9.35	**45.3**	24.1	**12.21**	3.36
8	2	7	**-4.41**	4.68	**40.4**	24.9	**8.58**	1.07
		15	**-5.26**	7.33	**41.5**	23.5	**10.64**	1.28
	4	7	**-7.54**	9.32	**46.7**	24.0	**12.42**	2.02
		15	**-5.64**	7.04	**45.6**	23.0	**11.25**	1.68

When the individual performance of the route-first, cluster-second approach is considered, it can be observed that as the problem size increases, the improvement of the procedure over the single picker solution also increases, up to a 7.54% improvement in the best case. This is due to the fact that this approach makes better use of multiple depots and the good performance of the merge-and-reach heuristic in order to avoid large detours when the problem size gets larger. The only exception is the case with 2 blocks and 3 pickers, where 3 pickers have to share 2 depots, which can end up with higher detours when compared to other problem settings. The improvement over the initial population best tends to go in parallel with the deviation from the single picker optima, which is expected. The amount of improvement over the random search seems to be indifferent as the problem size changes.

As for the individual performance of the cluster-first, route-second approach is concerned, we see an opposite trend. As the problem size increases, the deviation of this approach from the single picker optima increases as problem size increases. This is because the evolutionary approach actually solves the assignment problem, and therefore cannot make use of the performance of the merge-and-reach heuristic in order to avoid detours as problem size increases. The improvements over the best solutions in the initial population are also not promising, with no better result than 3.50%. As in the former approach, the improvement over random search solutions are fairly stable.

Computational Results with α = 75% and α = 25%

In this part, we impose both load capacity constraints and travel time limits on the pickers, and try a combined approach that employs both the cluster-first, route-second and route-first, cluster-second approaches.

In the preceding part, the route-first, cluster-second algorithm was observed to dominate the cluster-first, route-second algorithm significantly. However, when using a combined approach, we have to keep in mind the exploitation and exploration opportunities gained by using each of the algorithms. The algorithm settings are as follows: population size is 200, probability of mutation is 5% and infeasibility is allowed with penalty for the cluster-first, route-second approach.

Throughout this part, two combined approaches are considered: In the first one, we set α = 25% and start with a set of route-first, cluster-second solutions. In each generation,

we apply route-first, cluster-second approach with probability 75% and cluster-first, route-second approach with probability 25%. The main idea here is to make better use of good exploitation when $\alpha = 0\%$, and use exploration opportunities of the cluster-first, route-second approach to a certain extent.

As the second approach, we consider setting $\alpha = 75\%$ and start with a set of cluster-first, route-second solutions. In each generation, we apply cluster-first, route-second approach with probability 75% and route-first, cluster-second approach with probability 25%. As opposed to the first combined approach discussed, the idea is to make better use of good exploration when $\alpha = 100\%$, and use exploitation opportunities of the route-first, cluster-second approach.

For the computational experiments of the combined approach, we use a modified form of the random search procedure applied in the previous part. For the case with $\alpha = 75\%$, we record the number of individuals created during the reproduction processes of each instance, and generate that many cluster-first, route-second random search solutions. We then compare the result with the best of these solutions. Similarly, for $\alpha = 25\%$, we record the number of individuals created during the reproduction processes, and generate that many random route-first, cluster-second random individuals. We apply this procedure 30 times in order for the random search results to become comparable with those from the evolutionary algorithm.

The experiments are made using the problem settings and algorithm parameters discussed in the previous part. Table 5.2 shows the computational results for the experiments carried out for the cases with $\alpha = 25\%$ and $\alpha = 75\%$. As has been the case in the previous part, the comparisons are made in terms of the deviations from the single picker optima and the improvements over the random search and best initial population results.

When the individual performance of $\alpha = 25\%$, that is, the approach with 75% route-first, cluster second and 25% with cluster-first, route-second is considered in terms of the deviation from the single picker optima, it can be seen that the performance of the procedure increases with increasing problem size until 4 blocks except the case with 2 blocks and 3 pickers, then tends to stabilize around 10% improvement. As in the case of pure route-first, cluster-second approach, the improvement over the random search solutions are more or less independent of the problem parameters, with an average improvement around 52%. The improvements over the best initial population solutions are again parallel with the improvements over the single picker optima.

Table 5.2: Percent deviations from single picker optima and percent improvements over random search solutions and initial population incumbents for the cases $\alpha = 25\%$ and $\alpha = 75\%$ (averages of 30 instances), bold entries indicating superiority over the other method

			% Deviation from single picker optimal		% Improvement over random_search		% Improvement over init_pop_best	
Blocks	Pickers	Aisles	$\alpha = 25\%$	$\alpha = 75\%$	$\alpha = 25\%$	$\alpha = 75\%$	$\alpha = 25\%$	$\alpha = 75\%$
1	2	7	-0.84	**-1.24**	**41.9**	31.5	0.86	**2.93**
		15	-3.30	**-3.90**	**51.4**	34.7	**3.46**	2.97
2	2	7	-6.76	**-7.86**	**51.7**	44.4	**10.16**	7.18
		15	-6.26	**-8.82**	**51.6**	39.3	**9.50**	3.27
	3	7	-2.73	**-4.97**	**46.8**	32.2	**3.36**	3.49
		15	-3.77	**-4.98**	**46.6**	34.9	**6.58**	3.75
4	2	7	**-8.97**	-8.55	**54.9**	39.1	**13.73**	7.21
		15	**-9.98**	-9.50	**59.0**	40.0	**15.96**	8.66
	3	7	**-12.51**	-9.96	**56.5**	45.2	**15.71**	11.38
		15	**-12.64**	-9.54	**55.8**	45.0	**15.93**	12.59
8	2	7	**-7.31**	-7.04	**52.6**	36.4	**12.41**	7.74
		15	**-9.21**	-7.85	**53.0**	38.9	**15.94**	8.28
	4	7	**-11.60**	-9.39	**59.8**	41.6	**17.54**	9.02
		15	**-8.80**	-7.68	**58.0**	36.4	**15.31**	8.27

Considering the performance of $\alpha = 75\%$, that is, the approach with 25% route-first, cluster second and 75% with cluster-first, route-second procedures, the results are similar to those of the case with $\alpha = 25\%$ in terms of the improvement over the single picker optima, random search solutions and the best members of the initial population. For improvement over the single picker optima, the results tend to get better until 4 blocks with the same exception stated before, then tend to get stable. The improvement over random search is again almost independent of the problem settings, whereas improvement over initial population best increases with increasing problem size.

The most significant findings are obtained when the results are compared to the pure route-first, cluster second and cluster-first, route-second approaches. When the results in Tables 5.1 and 5.2 are compared, it is first seen that the combined approach with $\alpha = 25\%$ improves the solutions of the pure route-first, cluster-second approach in terms of improvements over the single picker optima, random search solutions and initial population best; hence the usage of exploration in this procedure is justified. What is more remarkable in this comparison is the performance of the approach with $\alpha = 75\%$. Although the pure cluster-first, route-second approach cannot improve on the single picker optima except one case, the approach with $\alpha = 75\%$ provides a significant amount of improvement over the single picker optima, which emphasizes the fact that using even a small amount of exploitation may significantly increase the solution quality. For both approaches, the improvements over the random search and initial population best are significantly better than those of the pure approaches.

When the performances of $\alpha = 25\%$ and $\alpha = 75\%$ are compared, noticeable results are observed. Although the pure route-first, cluster-second approach dominates the cluster-first, route-second approach in terms of the improvement over the single picker optima, for the combined approach, the case with $\alpha = 75\%$, which employs more cluster-first, route-second approach, performs better than the case with $\alpha = 25\%$ for especially the smaller sized problems. However, starting with the instances with 4 blocks, the $\alpha = 75\%$ approach starts to perform better than the other, which is again due to better avoiding of the detours by the route-first, cluster-second approach.

In short, our computational experiments show that with the problem settings given, the usage of multiple pickers is justified by the improvements over the single picker optima, the usage of an evolutionary approach is justified by the improvements over the random search solution

and initial population best, and usage of a combined approach making use of cluster-first, route-second and route-first, cluster-second approaches is justified by the improvements it makes over the pure approaches.

CHAPTER 6

THE ORDER-PICKING PROBLEM IN A PARALLEL-AISLE WAREHOUSE WITH TURN PENALTIES

In the preceding chapters, the order-picking problem (OPP) has been studied for the single and multiple picker cases by taking into account the total travel time (or distance) as the objective function. When the travel time is further divided into its components, one observes the time spent while travelling, entering and leaving the aisles, acceleration/deceleration of the picking vehicle, picking the items, and turning the corners or taking U-turns. The time spent while travelling is handled by the algorithms proposed in Chapters 4 and 5 for the single picker and multiple picker cases respectively. The time spent while entering/leaving the aisles, acceleration/deceleration of the picking vehicle, and picking the items is constant and unavoidable. Yet the time spent while corner turning or taking U-turns depends on the route of the picker. This is mostly applicable for warehouses where AS/RS are employed, unlike the manual order-picking assumption in Chapters 4 and 5.

In this chapter, we consider the effect of the turns in the travel time of the picker. We assume throughout the chapter that each right and left turn incurs 1 turn, whereas a U-turn incurs 2 turns. We first review the literature on turn minimization in general graphs in Section 6.1 and propose a polynomial time algorithm, linear in terms of the number of pick aisles for the 2-OPP graphs and k-OPP graphs with $k \geq 3$ in Sections 6.2 and 6.3 respectively.

6.1 Literature Survey on Turn Minimization in General Graphs

Despite the fact that the turn minimization problem has never been studied for the OPP, the problem has received considerable attention for various types of problems, due to its applica-

bility in many practical situations such as snow removal, street cleaning, garbage collection, mail delivery, etc.

To the best of our knowledge, the first study on turn minimization in a graph is due to Caldwell [14], who discusses the problem of finding the shortest route between two points in a street and freeway network of a city, such that a function of time and distance is minimized. The algorithm simply consists of modifying the travel distances (or times) between each pair of nodes and solving the shortest path problem, which can be done in polynomial time, between the origin and destination nodes.

Bodin and Kursh [11] discuss an application of turn and distance minimization for the routing and scheduling of street sweepers in a city road network. It is assumed that there are certain pickup points, and the objective is to complete the pickup operations of the sweepers in the shortest amount of time possible. The time incurred by the turns is embedded into the model by modifying the time required to travel between each pair of nodes. It is assumed that the sweeping operations can only be performed during time periods during which parking is prohibited on a street. Since this requirement differs among the streets of the city, the problem resembles the VRP with time windows. Consequently, a combined heuristic approach that makes use of both the cluster-first, route second and route-first-cluster-second is proposed. The algorithm is applied on a district of New York City and critical roads in Washington, D.C.

Roy and Rousseau [75] consider the Capacitated Chinese Postman Problem with the additional assumption that the depots are also to be located. The right, left and U-turns are penalized along with the travel distances and street crossings. The tour with turn penalties is transformed into a TSP by first transforming the arc routing problem into a node routing problem, then adding arcs to the new network to represent the penalty of changing arcs. The resulting problem on the given network with new arcs is solved using an allocation-routing-location based heuristic approach. The solution approach is applied for two different postal stations in the Canadian Postal Service network.

In Gendreau et al. [36], a transformation of the problem of distance minimization with turn penalties to a distance minimization problem is proposed. The transformation is made using artificial edges in order to represent the left, right and U-turns. The problem is then solved using the GENIUS heuristic for the TSP, which starts with a tour of three arbitrary vertices,

and iteratively adds the non-inserted nodes to the tour by inserting them between two of their closest neighbors on the tour. The algorithm is applied for 30 street networks in the Montreal suburbs, with consideration given to the deadheadings, left turns, U-turns, street crossings and street changes. Right turns are not considered, as they are complementary to the left turns, U-turns and street crossings.

Benavent and Soler [9] consider the case of a street network where some turns are prohibited, whereas others are allowed with penalty. In such a situation, the problem transforms into a directed rural postman problem with turn penalties. The problem is shown to be NP-hard by polynomial transformation to the strongly NP-hard directed Hamiltonian circuit problem, which tries to find the existence of a circuit which traverses every node of a graph exactly once. The algorithm they propose is an improvement heuristic over the construction heuristic in [11]. The procedure starts by polynomially transforming the rural postman problem to an asymmetric TSP. The construction heuristic by Bodin and Kursh [11] constructs a route that disregards the forbidden turns constraint. The improvement heuristic that follows the construction depends on two operations. The first one substitutes a non-required section of the tour by the appropriate shortest feasible chain. The second approach consists of crossing three appropriate turns made by the tour at the same node to obtain a new tour that differs from the original only by the turns made at this node.

Clossey et al. [17] propose an alternative transformation method for handling the turn penalties in directed graphs. Rather than transforming the problem into an asymmetric TSP, they propose an approach in which the graph is first transformed into an Eulerian graph. Following this, the Eulerian graph is traversed using the polynomial time algorithm by Edmonds and Johnson [28] with modifications that try to keep the turn penalties low. In all, the authors test six different strategies on random problems and observe that two of these perform significantly better than the others. These two strategies are compared against an exact algorithm and a patching heuristic. They are observed to perform faster than the exact algorithm and to have a better performance than the patching heuristic.

Arkin et al. [4] discuss a modified version of the problem, called the *minimum-turn milling* problem, in which the aim is to find a polygonal tour for a *cutter* so that it sweeps out a specified region (called the *pocket*), and the total number of turns is minimized. They consider several versions of the problem, which are called the *discrete milling problem*, the *thin discrete*

milling problem, the *orthogonal milling problem*, the *thin orthogonal milling problem* and the *integral orthogonal milling problem*. The NP-hardness of the minimum-turn milling problem is proven with reduction to the problem of finding Hamiltonian tours on grid graphs. Several heuristic procedures are proposed for both the minimum-turn milling problem as well as its versions, and the corresponding worst-case performances are given.

A further extension of the rural postman problem with turn penalties, which is its application on mixed graphs, is discussed in Corberan et al [18], who propose an algorithm for transforming this version of the problem into an asymmetric TSP. The transformation is a modification over the version in directed graphs. It is shown that such a transformation decreases the quality of the solutions for the problem. Consequently, a three-step heuristic procedure is proposed. In the first step, several feasible solutions are generated using a heuristic procedure for the mixed rural postman problem. The second step involves modifications on the solutions so that turn costs are decreased, and the third step tries to provide improvement on the solutions obtained by the second step. The procedure is tested on randomly generated problems and compared against exact and heuristic procedures.

As can be seen from the studies in the literature, the turn minimization is usually considered along with the distance minimization objective, and the problem is converted into a single objective problem. Additionally, the problems discussed throughout this section are usually more general, hence they are NP-hard. For the case of the OPP with turn penalties, the problem is polynomially solvable, as we will show in the next sections.

6.2 A Turn Minimizing Algorithm for 2-OPP

In this section, we propose a polynomial-time algorithm, which is linear in terms of the number of aisles, for the turn minimization problem with no middle aisles. First, it is proven that when the depot is on a cross aisle, the problem of minimizing the number of turns in a 2-OPP is equivalent to solving the Rural Postman Problem (RPP), with the required edges being the non-empty pick aisles. It is also shown that the solution of this problem is easy, and we propose a simple algorithm. Then we discuss several exceptional cases where this algorithm is not applicable. Considering the additional possible move types, the turn minimizing algorithm, which depends on the solution of the RPP, is modified so that the exceptional cases are

also handled.

Unless otherwise stated, it is assumed that the depot is on a cross aisle, and the picker starts the route without incurring any turns when starting from the depot, and ends the route without incurring any extra turns upon arriving at the depot. An *upward move* is defined to be a full traversal of a subaisle from its front end to the back end and a *downward move* is a full traversal of a subaisle from its back end to the front end.

6.2.1 Conversion to the Rural Postman Problem

In order to convert the turn minimization problem in a 2-OPP graph into the Rural Postman Problem, we first consider all possible move types (of which there are three), and prove that complete traversal of an aisle dominates partial traversal, thereby justifying the required edges in the Rural Postman Problem to be the edges corresponding to the non-empty aisles.

Assume that on an optimal route, the next vertex to be visited is v_i, which is on aisle k. Assume also that the route has entered aisle k from one of either of its ends (front or back). The purpose of defining these move types is to show that traversing a non-empty aisle performs at least as good as traversing it partially, that is, for any given route with partial aisle traversal, there exists an alternative route with complete traversal of that aisle which performs at least as good as the given tour.

Move Type 1 Consider vertex v_i and its adjacent vertex v_j, on aisle k. Suppose that v_i has been visited and v_j is the next vertex to be visited after v_i.

Observation 6.1 *When Move Type 1 applies, full traversal of aisle k performs as well as its partial traversal.*

Proof. There are two cases. Assume, without loss of generality, that the picker enters aisle k from the back end.

1. v_i is nearer to the back end than v_j. Then, v_j is simply reached after visiting v_i without incurring any turns and ending up facing b_k. Any other strategy ends up with more turns and end up b_k. Figure 6.1 depicts such a case.

Figure 6.1: An illustration of Move Type 1 - Case 1, where no turn is incurred

2. v_j is nearer to the back end than v_i. Then, there are two possible moves: The first visits v_i, makes a U-turn, then visits v_j. The second visits v_i, traverses the aisle until b_k, then makes a U-turn and visits v_j. Both moves incur 2 turns after visiting v_i and both end up facing a_k. Such a case is illustrated in Figure 6.2.

The first case does not consider full or partial traversal of the aisle, whereas the second case indicates that there is no difference in applying any of the two strategies. ■

Move Type 2 Consider vertex v_i on aisle k and vertex v_j on aisle l. Suppose that v_i has been visited and v_j is the next vertex to be visited after v_i.

Observation 6.2 *When Move Type 2 applies, full traversal of aisle k performs as well as its partial traversal.*

Proof. Assume, without loss of generality, that the picker enters aisle k from the back end. There are two options: The first one completely traverses aisle k, then makes a turn at b_k, enters aisle l from b_l and reaches v_j. This incurs 2 turns in total. The second option makes a U-turn after visiting v_i, makes a turn at a_k, enters aisle l from a_l and reaches v_j. This incurs a total of 4 turns. Figure 6.3 illustrates the two options.

It should be noted here that although complete traversal performs better than partial traversal in a single move, since the moves end up facing different ends of aisle l, the moves following

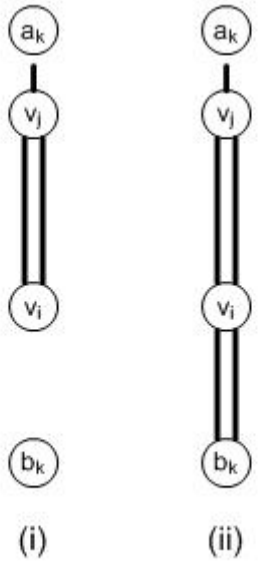

Figure 6.2: An illustration of move Type 1 - Case 2: (i) with partial traversal, (ii) with complete traversal. Both moves require 2 turns in total

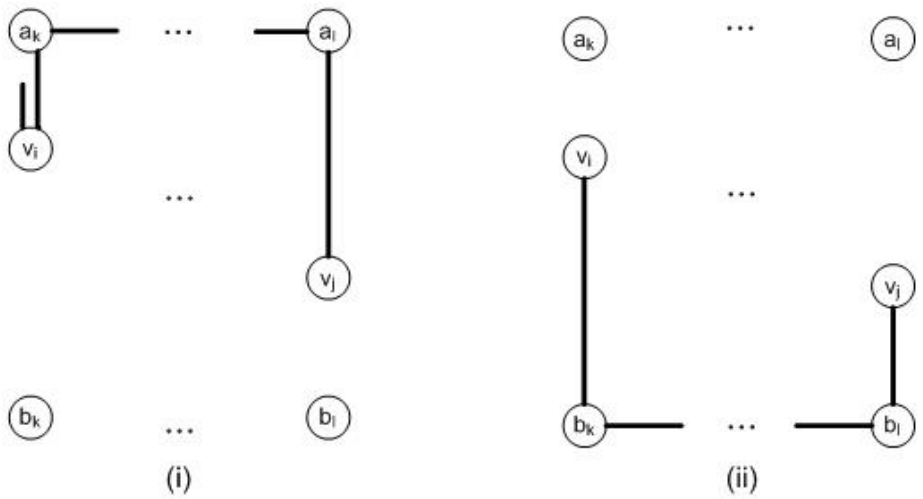

Figure 6.3: An illustration of Move Type 2: (i) with partial traversal, incurring 4 turns, (ii) with complete traversal, incurring 2 turns

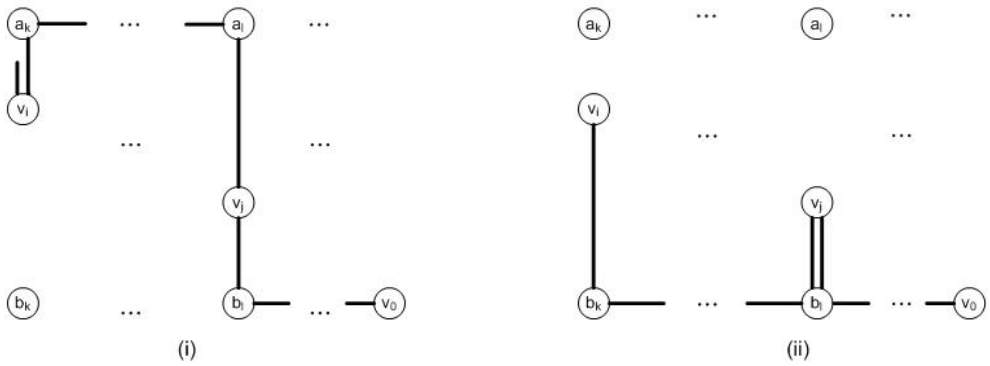

Figure 6.4: The only case when partial traversal can possibly perform better than complete traversal: (a) partial traversal of aisle k followed by reaching the depot incurs 5 turns, (b) complete traversal of aisle k followed by reaching the depot also incurs 5 turns

partial traversal may incur less turns than the ones following complete traversal. In fact, this is the only possibility of violation to the optimization of the complete route by optimizing the moves between the items. It can easily be shown that the only case when partial traversal can outperform complete traversal in the complete route is when the depot is on a cross aisle. This can occur when v_i is visited followed by a full traversal, then v_j is visited from aisle l and move type 3(i) (to be shown later) occurs, compared to visiting v_i followed by a partial traversal, then v_j is visited and move type 3(ii) (also to be shown later) occurs. However, both of these cases incur 5 turns in total, implying that partial traversal cannot outperform complete traversal in this case as well. Figure 6.4 summarizes the two possibilities. ■

Move Type 3 Consider vertex v_i on aisle k and vertex v_0, which is the depot node. Suppose that v_i has been visited and v_0 is the next vertex to be visited after v_i.

Observation 6.3 *When Move Type 3 applies, full traversal of aisle k performs as well as its partial traversal.*

Proof. There are two cases to be considered. Assume, without loss of generality, that the picker enters aisle k from the back end.

1. If the depot is on the back cross aisle, there are two options to traverse aisle k. The first one performs partial traversal, performs a U-turn after visiting v_i, exits aisle k using a_k and reaches the depot. This option incurs 3 turns. The second one performs complete

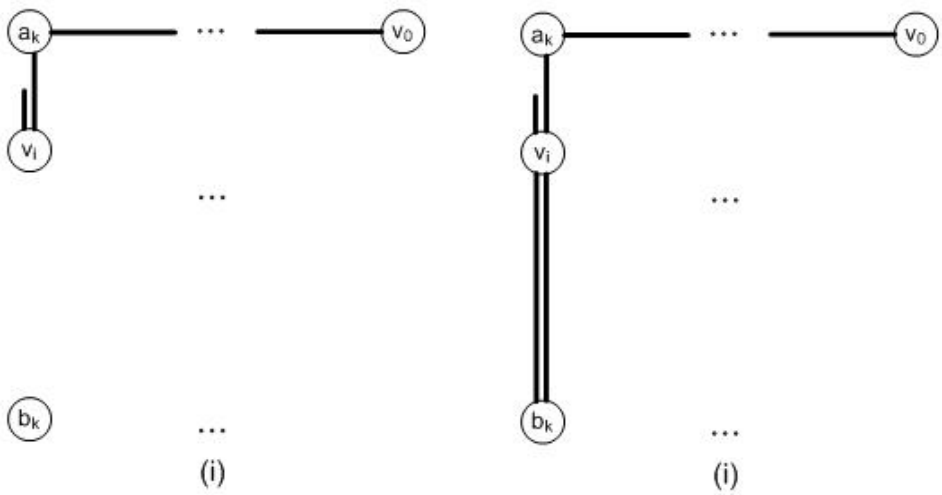

Figure 6.5: An illustration of Move Type 3 - Case 1: (i) with partial traversal, (ii) with complete traversal, both cases incurring 3 turns

traversal, visits b_k after visiting v_i, performs a U-turn at b_k and follows the same path as the first option. This also incurs 3 turns. Figure 6.5 illustrates such a case.

2. If the depot is on the front cross aisle, then exiting aisle k using b_k and reaching the depot incurs 1 turn. Any other strategy would end up with more turns. Figure 6.6 gives an example of this case.

■

Having defined all the applicable move types, we now give an important proposition.

Proposition 6.4 *A turn minimizing route for 2-OPP traverses all the non-empty pick aisles completely.*

Proof. Assume that at some point on the optimal route, the picker visits v_i on aisle k, and the next node to be visited is v_j on aisle l. Assume also that the picker enters aisle k from a_k.

1. If $l = k$ and $j \neq 0$, then by Observation 6.1, complete traversal of aisle k performs as well as its partial traversal.

2. If $l \neq k$ and $j \neq 0$, then by observation 6.2, complete traversal of aisle k performs better than its partial traversal.

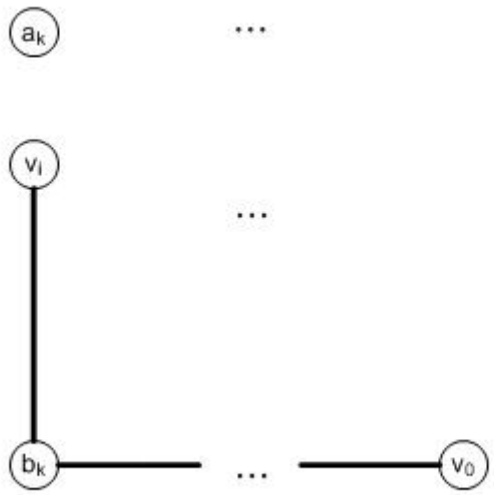

Figure 6.6: An illustration of Move Type 3 - Case 2, incurring 1 turn

3. If $j = 0$, then by Observation 6.3, complete traversal of aisle k performs better than its partial traversal.

Since we know that optimizing each single move leads to the optimization of the complete route, complete traversal of each aisle that contains items leads to the optimal tour that minimizes the number of turns. ■

The following theorem is due to Proposition 6.4.

Theorem 6.5 *The problem of minimizing the number of turns on a 2-OPP graph is equivalent to solving the Rural Postman Problem (RPP) with the objective of distance minimization, and the required edges as the ones corresponding to the non-empty aisles.*

Proof. Full traversal of the edges is justified by Proposition 6.4. Since the route starts and ends at the depot, we need to have: number of upward moves = number of downward moves. If the number of required edges is even, there is no need to duplicate any of them or add any new vertical edge. If the number of edges is odd, then certainly one of the required edges should be duplicated or a new edge is to be added. In any case, the number of vertical edges in the optimal solution to the RPP is fixed. Thus, since the RPP minimizes the total distance to be travelled, in this case, it tries to find the minimum number of horizontal edges that will

end up with a feasible Eulerian tour, when combined with the vertical edges.

Assume that a feasible RPP solution, i.e., a Eulerian tour with minimum number of horizontal edges, is found. If any new vertical edges are added to this solution without destroying the Eulerian property, the total number of turns obviously increases. If new horizontal edges are added without destroying the Eulerian property, the number of turns will not decrease, which is obvious. Therefore the solution with fixed required number of vertical edges and minimum number of horizontal edges is the one with minimum number of turns. ■

6.2.2 The Algorithm

The Rural Postman Problem has been known to be NP-hard on general graphs. However, due to the special structure of the OPP graph, we propose a polynomial time algorithm for the problem on the OPP graphs, which is of linear complexity in terms of the number of pick aisles. We make use of the following facts:

- The number of vertical edges in the optimal solution equals the number of non-empty aisles, if there is an even number of them. In the case when their number is odd, one of the vertical edges should be duplicated.

- If none of the left-most and right-most required edges is duplicated, then in the optimal solution, the degrees of the vertices incident to them will be 2 each. For the vertices incident to empty aisles, the sum of the degrees totals 4 (2 for the forward route, 2 for the backward route of the picker). For the vertices incident to the edges in between, the sum of the degrees is 6. This is due to the fact that a total of 2 degrees (1 for each) is added by the required edge; 2 degrees will be added to one for the forward route of the picker, whereas 2 more will be added to one for the picker‘s route back to the depot. For the aisle where there is duplication, total number of degrees increases by 2, which is obvious.

- If the depot is on a cross aisle to the left of the left-most nonempty aisle or to the right of the right-most nonempty aisle, it can be moved to the nearest corner of the nonempty aisle, as the total number of turns will not change.

```
begin

\\initialization
-   If the depot is to the left of the left-most non-empty aisle or to the right of the right-most non-
    empty aisle, move it to the nearest corner of the nearest non-empty aisle and add 1 turn to the
    solution;
-   Set all remaining non-empty aisles as required edges. If the number of non-empty aisles is odd
    and Move Type 4 – Case 3 does not hold, duplicate one of the required edges arbitrarily. Number
    the required edges from the left-most to the right-most disregarding the duplicated one;
-   j = 0;

\\subtour formation
while j ≠ no_of_nonempty_aisles and j is odd do

-   Disregarding the duplicated required edges, make all degrees of the vertices incident to the
    required edges even by forming a subtour between the j^th and j+1^st required edge;

endwhile

\\tour formation
while j ≠ no_of_nonempty_aisles and j is odd do

-   Join the adjacent subtours by using double edges from any arbitrary side (front or back cross
    aisle), with the restriction that the cross aisle containing the depot has to be included;

endwhile

end
```

Figure 6.7: Algorithm *turn_2-OPP*

Based on the given facts, we propose the algorithm given as Algorithm *turn_2-OPP* in Figure 6.7.

The algorithm initializes by moving the depot to the corner of the nearest non-empty pick aisle if necessary. Then, the required edges are formed and numbered. The first *while* loop forms subtours by joining the required edges using horizontal edges. The second *while* loop joins these subtours using double horizontal edges from arbitrarily selected cross aisles. The complexity of the algorithm is $O(m)$, where m is the number of pick aisles, due to the fact that there can be at most m non-empty pick aisles and each of the *while* loops can be called at most $\frac{m}{2}$ times.

6.2.3 Extensions

So far it has been assumed that the depot is at one of the front or back cross aisles, which is a very common assumption in the OPP. However, in the OPP with turn penalties, one cannot

assume it without loss of generality. For this purpose, we now show that if the depot is at one of the aisles, the algorithm given in the preceding section needs to be modified for this case. As in the previous case, it is shown that the problem of minimizing turns in a 2-OPP graph where the depot is on an aisle is equivalent to solving a Rural Postman Problem. In order to achieve this end, a fourth move type is defined.

Move Type 4 Consider vertex v_0 (the depot node) on aisle k and vertex v_i. Suppose that v_0 has been visited and v_i is the next vertex to be visited after v_0.

Observation 6.6 *Proposition 6.4 still holds for the case when the depot is on one of the aisles except for the special case in which there are odd number of non-empty pick aisles and there are items on both sides of the depot.*

Proof. For all cases, we need to keep in mind that the number of upward moves must be equal to the number of downward moves so that the tour ends at the depot. There are three possible cases. The first two prove that when the exceptional case is not valid, Proposition 6.4 still holds. The last case discusses why Proposition 6.4 does not hold for the exceptional case. Assume, without loss of generality, that the picker exits aisle k from a_k.

1. When there are even non-empty aisles, since the number of upward moves equals the number of downward moves, there always exists a route that visits all the required nodes by complete traversal of their pick aisles and re-enters aisle k using b_k before returning to the depot. Figure 6.8 depicts such a case.

2. When there are odd non-empty aisles but there exists at least one $v_i, i \neq 0$ on both sides of v_0, then due to the equality of upward and downward moves, the tour requires that there be a U-turn at one of the aisles, or the tour traverse one of the empty pick aisles completely. If there is a U-turn at one of the pick aisles, say l, (the U-turn can be made at a_l without extra turns), or if the tour uses an empty aisle, the case is converted into the first case of this move type, where complete traversal is justified. Figure 6.9 illustrates a case where the U-turn is made on aisle k.

3. When there are odd non-empty aisles and there exists at least one $v_i, i \neq 0$ on at most one side of v_0, then due to the assumption of not incurring extra turns for entering and exiting the depot, complete traversal of k as in (2) is not justified. When the aisle is

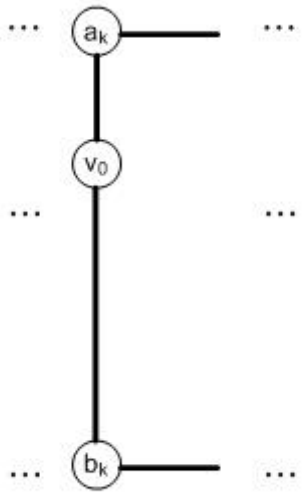

Figure 6.8: When there are even non-empty aisles, complete traversal of aisle k is possible and performs better than partial traversal

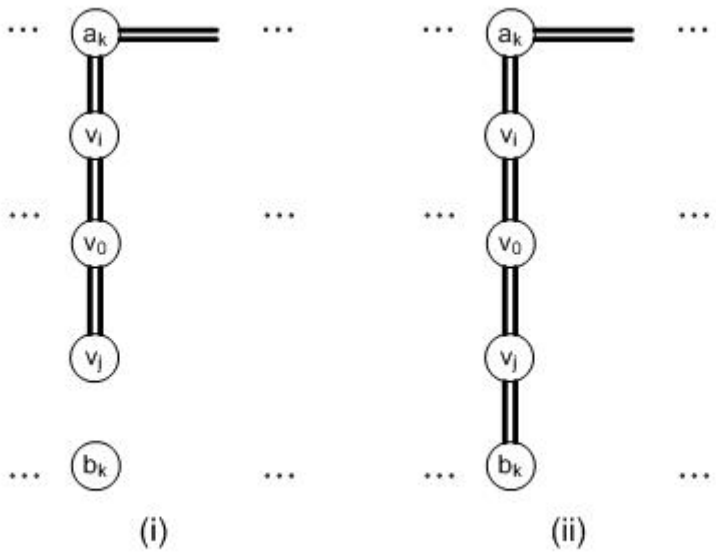

Figure 6.9: When there are odd non-empty aisles and items on both sides of v_0, a U-turn at aisle k is an alternative optimal, (i) the U-turn with partial traversal, (ii) the U-turn with complete traversal. Both incur 2 extra turns.

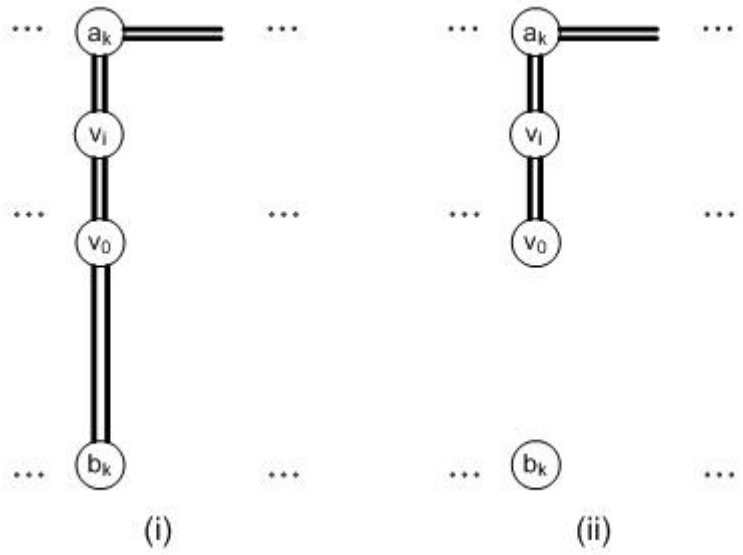

Figure 6.10: There are odd non-empty aisles and an item on only one side of v_0, (i) complete traversal results in 2 extra turns, (ii) partial traversal of k

exited and entered from the same end (the one between which and the depot there are items) incurs 2 less turns than (2). Figure 6.10 illustrates the case. In (i), complete traversal incurs an extra U-turn at b_k, whereas in (ii) partial traversal does not.

■

The Modified Algorithm

The algorithm for turn minimization in 2-OPP graphs can be modified so that the exceptional case in Move Type 4 - Case 3 can be handled. For this, when there are odd non-empty aisles and items on at most one side of v_0, we only need to duplicate aisle k, which contains v_0 partially between the depot until a_k or b_k (between whichever and v_0 there exists at least an item). Figure 6.11 shows the modified version of Algorithm *turn_2-OPP*, called Algorithm *turn_2-OPP_modified*.

An Example Problem

The complexity of the algorithm is again $O(n)$, due to the fact that Algorithm *turn_2-OPP* is called at most once.

In order to illustrate the usage of Algorithm *turn_2-OPP_modified*, we discuss it on a small

begin

\\initialization

- If the depot is to the left of the left-most non-empty aisle or to the right of the right-most non-empty aisle, move it to the nearest corner of the nearest non-empty aisle and add 1 turn to the solution;
- If the exceptional case in Move Type 4 – Case 3 holds, duplicate aisle k partially between v_0 and the corner vertex between which and v_0 there exists at least one item;
- Set all remaining non-empty aisles as required edges. If the number of non-empty aisles is odd and Move Type 4 – Case 3 does not hold, duplicate one of the required edges arbitrarily. Number the required edges from the left-most to the right-most disregarding the duplicated one;
- j = 0;

\\subtour formation

while j ≠ no_of_nonempty_aisles **and** j is odd **do**

- Disregarding the duplicated required edges, make all degrees of the vertices incident to the required edges even by forming a subtour between the jth and $j+1^{st}$ required edge;

endwhile

\\tour formation

while j ≠ no_of_nonempty_aisles **and** j is odd **do**

- Join the adjacent subtours by using double edges from any arbitrary side (front or back cross aisle), with the restriction that the cross aisle containing the depot has to be included;

endwhile

end

Figure 6.11: Algorithm *turn_2-OPP_modified*

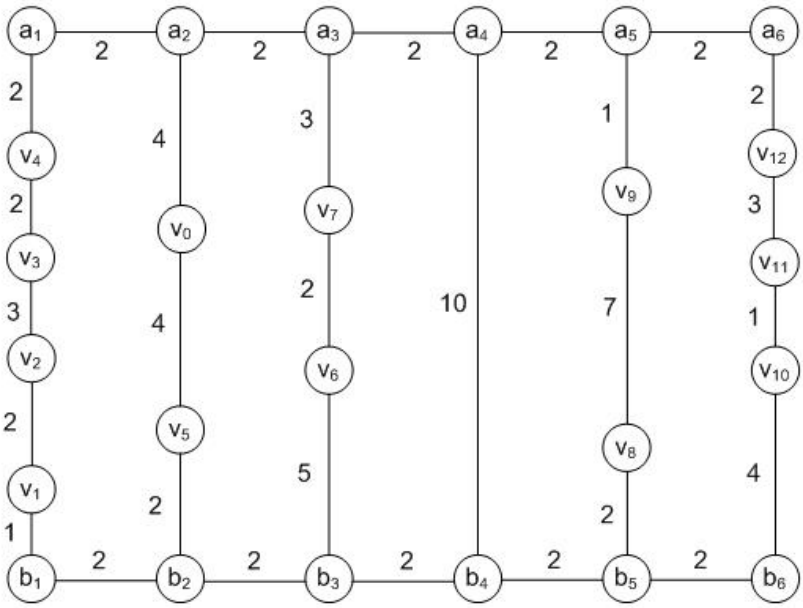

Figure 6.12: An example problem with 6 pick aisles, 5 non-empty aisles and 12 items

example problem consisting of 6 pick aisles, 5 non-empty aisles and 12 items. The example problem is depicted in Figure 6.12.

At the initialization step, we first check the position of the depot node, and see that it is between the left-most and right-most non-empty aisles, therefore no modification is made. Due to the fact that there are 5 non-empty aisles and there is an item (namely v_5) between v_0 and b_2, the edge (v_5 b_2) is a required edge which is duplicated. For all the other non-empty aisles k, the edges (a_k b_k) are set as required edges, as shown in Figure 6.13(i). Then, the edges corresponding to pick aisles 1-3 and 5-6 are joined together to form subtours so that all the degrees of the vertices in the graph are even. The resulting situation is given in Figure 6.13(ii). Lastly, the subtours are joined by double horizontal edges from arbitrary cross aisles. The solution is depicted in 6.13(iii). The resulting optimal solution has 10 turns.

Before closing this section, we propose an observation and a theorem.

Observation 6.7 *An optimal solution to the turn minimization problem on a 2-OPP graph can be found using a "reduced" form of the algorithm proposed by Ratliff and Rosenthal [71] for distance minimization in an order picking tour, where the connection types (1), (2), (3), (5) and (6) in Figure 3.5 are used for within-aisle connections, and the connection types (1),*

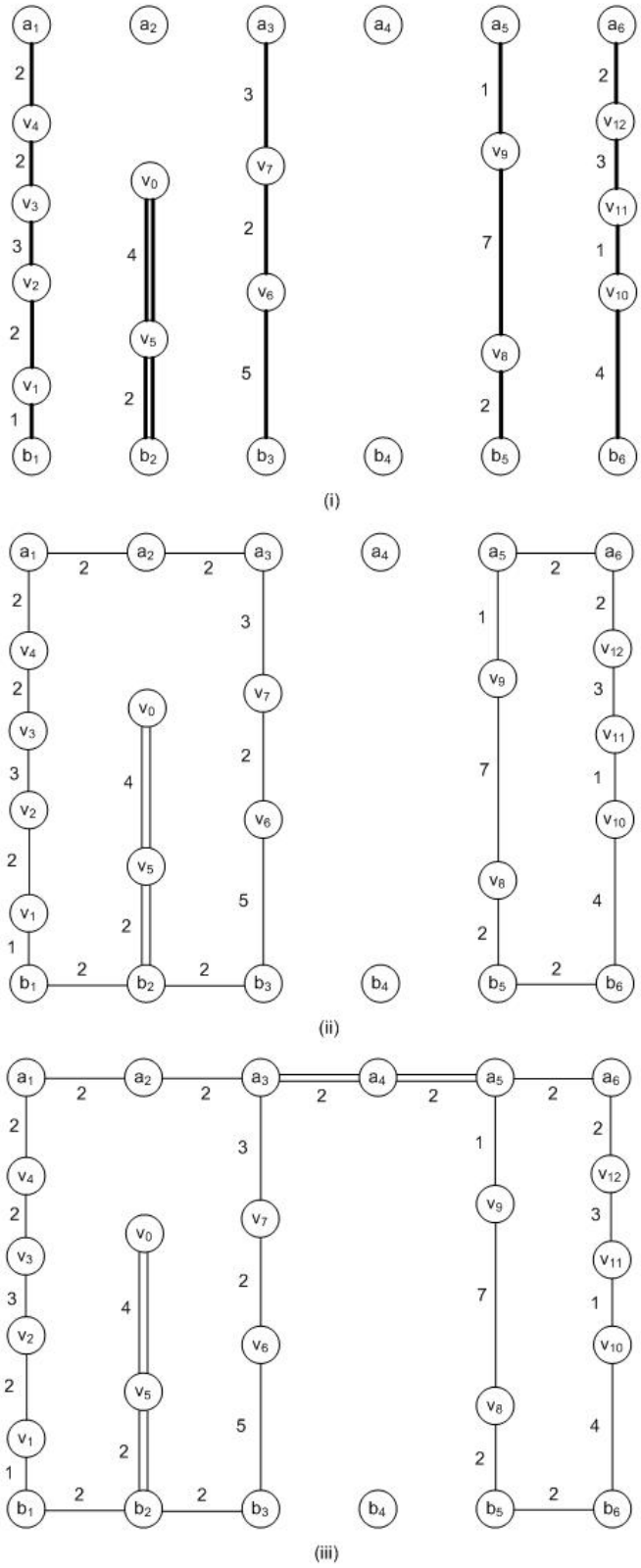

Figure 6.13: Step-by-step solution to the problem in Figure 6.12, (i) required edges, (ii) the subtours, (iii) the optimal solution with 10 turns

(2), (3) and (5) in Figure 3.6 are used for connecting the aisles.

The details of the algorithm and the solution to the example problem are given in Appendix D.

Theorem 6.8 *When the depot is on one end of the front or back cross aisle, the optimal solution to the turn minimization problem can be found by the S-shape heuristic for the distance minimization version of the 2-OPP.*

Proof. In tour formation step of Algorithm *turn_2-OPP*, if subtours are all connected from the same cross aisle (one that includes the depot), one obtains a route constructed using the S-shape heuristic. ■

6.3 A Turn Minimizing Algorithm for k-OPP with $k \geq 3$

The main complication when middle aisles are included in the turn minimization problem on a k-OPP graph is the fact that it may become plausible to use the middle aisles in certain cases. In the cases in which using the middle aisles is not plausible, the problem becomes equivalent to solving the turn minimization problem on a 2-OPP graph. In this section, we first show that when the depot is on one of the front or back cross aisles, the problem can be converted into the Rural Postman Problem, with the required edges being the non-empty subaisles. Then, we propose a linear-time tour construction algorithm that finds the turn minimizing route. Lastly, we consider the extensions where the depot is either on a pick aisle or on one of the middle aisles. We extend the algorithm to include these extensions as well.

6.3.1 Conversion to the Rural Postman Problem

Throughout this section, we make use of the following observation.

Observation 6.9 *For the cases where the depot is on the front or back cross aisle and lies on the corner of one of the non-empty pick aisles, the optimal number of turns is at least* $2j-1$*, where* j *corresponds to the number of non-empty pick aisles.*

Proof. The proof is straightforward: For each non-empty pick aisle, we need one turn to enter the aisle, and one turn to exit it. Since the depot is adjacent to one of the non-empty pick aisles, that specific aisle does not require any turns for entering. Since there are j non-empty pick aisles, the total number of required turns is at least $2j - 1$. ■

It follows directly from Observation 6.9 that when the depot is not on the corner of a non-empty aisle, we need at least $2j$ turns to complete the tour. The following theorem is directly proven by the observation.

Theorem 6.10 *For the turn minimization problem on a k-OPP graph with $k \geq 3$, when the depot is on the front or back cross aisle and there are even non-empty pick aisles, Algorithm turn_2-OPP finds the optimal solution.*

Proof. The algorithm ends up with $2j-1$ turns when the depot is on the corner of a non-empty aisle, and with $2j$ turns when it is not. By Observation 6.9, the solution is optimal. ■

Since the problem can be solved by Algorithm *turn_2-OPP*, the following observation directly follows.

Observation 6.11 *When there are even non-empty pick aisles, the turn minimizing tour on a k-OPP graph with $k \geq 3$ traverses all the non-empty pick aisles completely, and the problem of minimizing turns on the graph is equivalent to solving the Rural Postman Problem on the same graph with the objective of distance minimization, and with the required edges as the non-empty pick aisles.*

For the case with odd non-empty pick aisles, the situation is a little more complicated, as it might become plausible to traverse the middle aisles to end up with the optimal solution. In the following theorem, we present the only case where such a traversal is plausible.

Theorem 6.12 *Suppose we number the blocks, starting with the front-most one, from 1 up to $k-1$; while denoting the cross aisles in the same manner from 1 to k. Suppose we also number the pick aisles, starting with the left-most one, from 1 to $j-1$, and we denote the subaisles as (a, b), where a refers to the block and b refers to the pick aisle it belongs to. If there exists a cross aisle, say n, such that for some pick aisle p_1, all the subaisles $(1, p_1), \ldots, (n-1, p_1)$ are*

all empty, and for some other pick aisle p_2, all the subaisles $(n, p_2), \ldots, (k, p_2)$ are all empty; then it becomes plausible to traverse cross aisle n between the pick aisles p_1 and p_2.

Proof. It is easy to see that when we apply Algorithm *turn_2-OPP* with the required edges as all the non-empty pick aisles other than p_1 and p_2, as well as the edge that is formed by joining the edges corresponding to subaisles $(n, p_1), \ldots, (k, p_1)$ and $(1, p_2), \ldots, (n-1, p_2)$ by the cross aisle n between pick aisles p_1 and p_2, then the algorithm comes up with a solution of $2j-1$ turns when the depot is on the corner of a non-empy pick aisle, whereas it finds a solution of $2j$ when such a case does not hold. ■

We will see an example that illustrates the situation described in Theorem 6.12 in the next section. It should be noted here that the algorithm joins two pairs of unions of subaisles, and treats them as a single pick aisle. When this situation does not hold, crossing of middle aisles never improves the solution. Due to this, we have the following observation.

Observation 6.13 *When the special case in Theorem 6.12 does not hold, the turn minimizing tour on a k-OPP graph with $k \geq 3$ traverses each non-empty pick aisle completely, and the problem becomes equivalent to solving the RPP on the same graph with distance minimization objective, and with the required edges as the non-empty pick aisles. When the special case does hold, then the turn minimizing tour traverses each non-empty subaisle completely, and the problem becomes equivalent to solving the RPP on the same graph with distance minimization objective, and with the required edges as the non-empty subaisles.*

6.3.2 The Algorithm

Again, despite the fact that the RPP is NP-hard on general graphs, we take into account the special structures of the OPP graph to propose a polynomial-time algorithm, which is linear in terms of the number of pick aisles, and which is a simple extension of Algorithm *turn_2-OPP*. The following facts are useful in developing the algorithm.

- When there are even non-empty pick aisles, or there are odd non-empty pick aisles and the special case in Theorem 6.12 does not hold, then Algorithm *turn_2-OPP* finds the optimal solution.

- When there are odd non-empty pick aisles and the special case in Theorem 6.12, the algorithm can be modified to include the union of non-empty subaisles of the pick aisles p_1 and p_2 by cross aisle n so that the union is assumed as a single required edge in the algorithm.

Based on these facts, we propose the algorithm in Figure 6.14, given as Algorithm *turn_k-OPP*.

```
begin

\\initialization
-   Number the blocks, starting with the front-most one, from 1 up to k-1;
-   Number the cross aisles in the same manner from 1 to k;
-   Number the pick-aisles, starting with the left-most one, from 1 to j-1;
-   Let (a,b) denote each subaisle such that a refers to the block and b refers to the pick-aisle it belongs
    to;

if j is odd and there exists a cross aisle n, such that for some pick-aisle p1, all the subaisles (1,p1),..., (n-
1,p1) are all empty, and for some other pick-aisle p2, all the subaisles (n,p2),..., (k,p2) are all empty do

-   Join the edges corresponding to subaisles (n,p1),...,(k,p1) and (1,p2),...,(n-1,p2) by the cross aisle n
    between pick-aisles p1 and p2 and call the new edge E';
-   Apply Algorithm turn_2-OPP to the problem with the required edges as all the non-empty aisles
    other than p1 and p2 as well as E';

endif

else do

-   Apply Algorithm turn_2-OPP to the problem;

endif

end
```

Figure 6.14: Algorithm *turn_k-OPP*

The *if* statement checks whether the special case in Theorem 6.12 holds. If it does, it modifies the required edges and applies Algorithm *turn_k-OPP*; otherwise, it applies Algorithm *turn_k-OPP* directly. The complexity is O(n), as Algorithm *turn_2-OPP* is called once.

An Example Problem

The application of the algorithm to the cases where the special case does not hold is more or less straightforward. Therefore, the algorithm is illustrated on an example problem that includes the special case given in Theorem 6.12. The graphical representation of the example problem, which consists of 2 blocks, 5 pick aisles and 14 items, is given in Figure 6.15. The

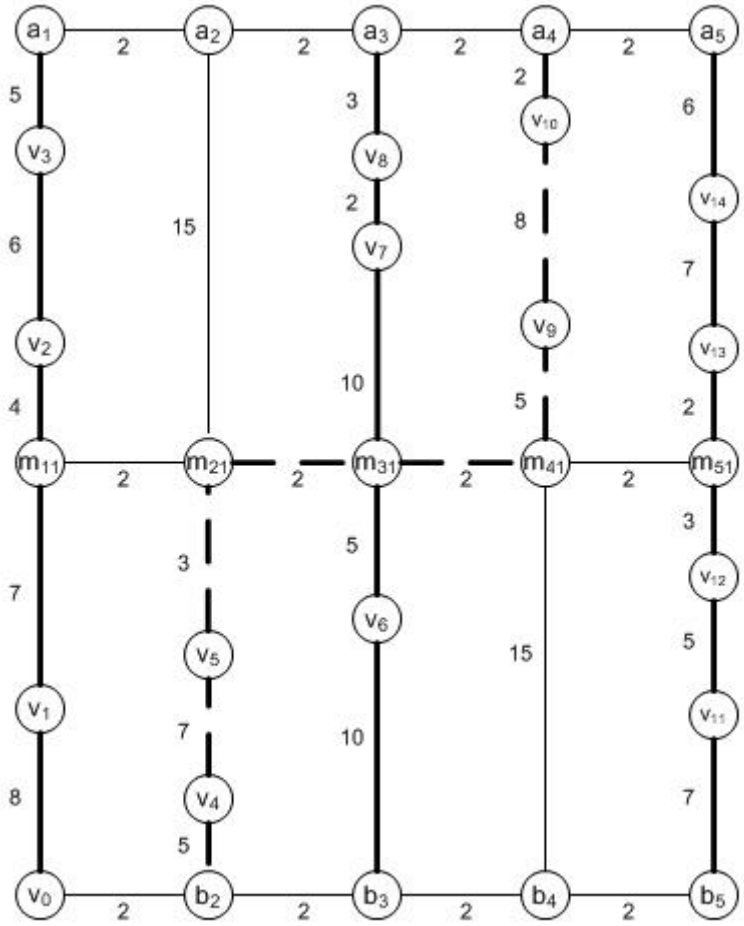

Figure 6.15: An example problem with 2 blocks, 5 non-empty pick aisles and 14 items, with the required edges shown in bold and the unified edge shown in dashed lines

required edges are shown in bold, and the unified edge is shown in dashed lines.

From Figure 6.15, one can observe that for $n = 2$, we have subaisles $(1, 2)$ and $(2, 4)$ empty, which satisfies the condition in Theorem 6.12. Thus we unify the subaisles $(2, 2)$ and $(1, 4)$ using the middle aisle between pick aisles 2 and 4, and set it as a required edge along with pick aisles 1, 3 and 5, as also shown in Figure 6.15. In Figure 6.16, the solution of the problem in Figure 6.15 is depicted. Figure 6.16 shows the subtour formation step, where the first required edge is joined with the dashed edge, and the remaining two edges are joined to form the second subtour. Since the subtour formation step ends up with a Eulerian tour, there is no need for the tour formation step. The optimal solution has 9 turns.

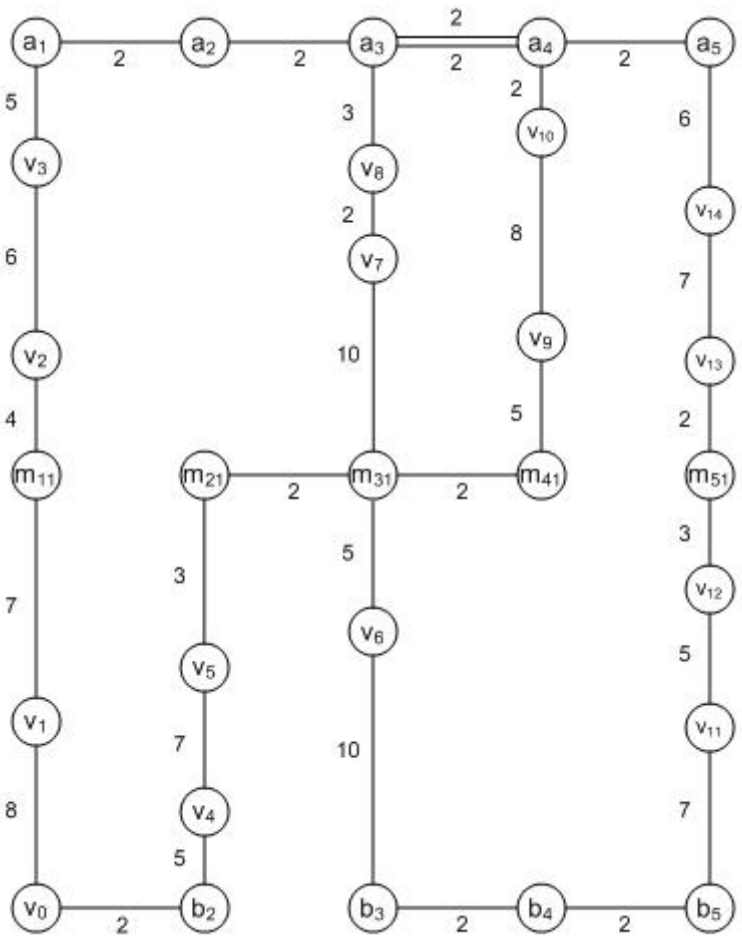

Figure 6.16: The optimal solution to the turn minimization problem given in Figure 6.15

6.3.3 Extensions

As in the case without middle aisles, it is useful to extend the solution approach to the cases where the depot lies on a pick aisle or a middle aisle. Again, we modify our algorithm to include these extensions as well. We show that when the depot is on a pick aisle, Algorithm *turn_2-OPP_modified* is still applicable, whereas for the depot on the middle aisle, we show that Algorithm *turn_k-OPP* is applicable except one special case, and we modify the algorithm to include that special case as well.

First, consider the case where the depot is on a pick aisle. As in the previous part, we make use of the following theorem.

Theorem 6.14 *When the depot is on a pick aisle, Algorithm turn_2-OPP_modified solves the turn minimization problem to optimality.*

Proof. For the case with even non-empty pick aisles, this holds due to the fact that Algorithm *turn_2-OPP_modified* ends up with $2j$ turns. For each non-empty aisle (including the one that includes the depot), we need one turn for entering the aisle and one for exitting it. Hence $2j$ turns are optimal.

For the case with odd non-empty pick aisles, when the special case in Theorem 6.12 does not hold, due to the fact that crossing the middle aisles does not improve the solution, we need at least $2j+2$ turns. Since Algorithm *turn_2-OPP_modified* finds exactly that many turns in this case, the optimal solution can be found using the algorithm.

For the case with odd non-empty pick aisles, suppose that the special case in Theorem 6.12 holds, and the depot is on a subaisle $(1, p)$ or $(k-1, p)$ for some p, then applying Algorithm *turn_2-OPP_modified* comes up with $2j$ turns, hence is optimal. ■

The following observation directly follows.

Observation 6.15 *When the depot is on a pick aisle, complete traversal of subaisles is still justified with the exception of the case with odd pick aisles, the depot on a subaisle* $(1, p)$ *or* $(k-1, p)$ *for some p, and there are items on at most one side of the depot. Disregarding the exception, the problem is equivalent to solving the RPP on the corresponding OPP graph with the required edges as non-empty subaisles.*

Next, consider the case where the depot is on a middle aisle. Since we need additional two turns, one to enter and one to exit the middle aisle, the lower bound on the minimum number of turns becomes $2j + 2$ for j non-empty pick aisles. Based on this, we have the following theorem.

As in the case with the depot on the front or back cross aisle, the complication of the problem is the fact that it may be desirable to use the middle aisles on the optimal route. The following theorem presents the case where such crossings are desirable.

Theorem 6.16 *Suppose that the numbering of blocks, pick aisles and subaisles as in Theorem 6.12. If there are even (odd) cross aisles, and there exists a cross aisle, say n, such that for a pair of pick aisles p_1 and p_2 (a pick aisle p_1), all the subaisles $(1, p_1), \ldots, (n-1, p_1)$ and $(1, p_2), \ldots, (n-1, p_2)$ (only $(1, p_1), \ldots, (n-1, p_1)$ for odd) are empty (case 1), or all the subaisles $(n, p_1), \ldots, (k-1, p_1)$ and $(n, p_2), \ldots, (k-1, p_2)$ (only $(n, p_1), \ldots, (k-1, p_1)$ for odd) are empty (case 2). For case 1, setting subaisles $(n, p_1), \ldots, (k-1, p_1)$ and $(n, p_2), \ldots, (k-1, p_2)$ (only $(n, p_1), \ldots, (k-1, p_1)$ for odd) as required edges along with the remaining non-empty pick aisles; for case 2, setting subaisles $(1, p_1), \ldots, (n-1, p_1)$ and $(1, p_2), \ldots, (n-1, p_2)$ (only $(1, p_1), \ldots, (n-1, p_1)$ for odd) as required edges along with the remaining non-empty pick aisles, followed by application of Algorithm turn_k-OPP (with joining the subaisles to form a subtour), joined to the depot node, finds the minimum number of turns.*

Proof. Again, it is easy to see that the number of turns is equal to $2j$ for even non-empty pick aisles and $2j + 2$ for odd non-empty pick aisles, equal to the lower bound on the minimum number, indicating optimality. ■

Observation 6.17 *When the depot is on a middle aisle, and the special case in Theorem 6.16 does not hold, Algorithm turn_2-OPP solves the problem to optimality.*

In the next section, two examples, each including the special cases discussed in this section, will be given. As a consequence to this section, we have the following observation.

Observation 6.18 *With the exception of the special case in Observation 6.15, the optimal solution to the turn minimization problem traverses each non-empty subaisle completely. Additionally, the problem is equivalent to solving the RPP on the corresponding OPP graph, with the required edges as the non-empty subaisles.*

The Modified Algorithm

In order to handle the special cases discussed here, the Algorithm *turn_k-OPP* can be modified so that optimal solutions are found for all settings. There are two exceptions that we need to consider. First, when the depot is on a pick aisle and the special case in Observation 6.15 occurs, then we need to include the pick aisle until the depot node and duplicate the edge corresponding to this specific aisle, before solving the RPP. Secondly, when the depot is on a middle aisle and the special case in Observation 6.16 occurs, we need to form a subtour between the remaining subaisles of the pick aisles p_1 and p_2. Figure 6.17 summarizes the modified version of Algorithm *turn_k-OPP*, called Algorithm *turn_k-OPP_modified*.

The first if statement checks whether the special case in Theorem 6.12 occurs. If it does, a required edge E' is formed and Algorithm *turn_2-OPP* is called. The second if statement checks if Move Type 4 - Case 3 (discussed in Observation 6.15) occurs, in which case Algorithm *turn_2-OPP_modified* is called. Lastly, the third if statement checks whether the special case in Theorem 6.16 occurs. In such a case, necessary modifications are made and Algorithm *turn_k-OPP* is called. If none of the special cases occurs, the algorithm calls *turn_2-OPP* and connects the depot node to the resulting tour if necessary.

The complexity is O(n), as either Algorithm *turn_2-OPP*, or Algorithm *turn_2-OPP_modified*, or Algorithm *turn_k-OPP* is called once.

Two Example Problems

We now illustrate Algorithm *turn_k-OPP_modified* on two example problems. In the first example, the special case in Observation 6.15 is examplified. The second example illustrates the special case in Theorem 6.16.

The first example problem is depicted in Figure 6.18, with the required edges shown in bold. The problem has 2 blocks, 5 non-empty pick aisles, and 14 items. Due to the fact that there are 5 (odd) non-empty pick aisles, the depot node is on the up-most block and there are no items between v_0 and a_2, the special case in Observation 6.15 is evident here. Therefore we partially select the edge between v_0 and b_2 as a required edge and duplicate it, whereas the remaining non-empty pick aisles are also required edges.

After forming to subtours between non-duplicated required edges and forming the tours by

begin

initialization

- Number the blocks, starting with the front-most one, from 1 up to k-1;
- Number the cross aisles in the same manner from 1 to k;
- Number the pick-aisles, starting with the left-most one, from 1 to j-1;
- Let (a,b) denote each subaisle such that a refers to the block and b refers to the pick-aisle it belongs to;

if the depot is on the front or back cross aisle **and** j is odd **and** there exists a cross aisle n, such that for some pick-aisle p_1, all the subaisles $(1,p_1),\ldots,(n-1,p_1)$ are all empty, and for some other pick-aisle p_2, all the subaisles $(n,p_2),\ldots,(k,p_2)$ are all empty **do**

- Join the edges corresponding to subaisles $(n,p_1),\ldots,(k,p_1)$ and $(1,p_2),\ldots,(n-1,p_2)$ by the cross aisle n between pick-aisles p_1 and p_2 and call the new edge E';
- Apply Algorithm *turn_2-OPP* to the problem with the required edges as all the non-empty aisles other than p_1 and p_2 as well as E';

endif

if the depot is on a pick-aisle **and** j is odd **and** the special case of Move Type 4 – Case 3 occurs **do**

- Apply Algorithm *turn_2-OPP_modified* to the problem;

endif

if the depot is on a middle aisle **and** j is even (odd) **and** there exists a cross aisle n, such that for some pick-aisle p_1 and p_2 (p_1 for odd), $(1,p_1),\ldots,(n-1,p_1)$ and $(1,p_2),\ldots,(n-1,p_2)$ ($(1,p_1),\ldots,(n-1,p_1)$ for odd) are empty (case 1), or all the subaisles $(n,p_1),\ldots,(k-1,p_1)$ and $(n,p_2),\ldots,(k-1,p_2)$ ($(n,p_1),\ldots,(k-1,p_1)$ for odd) are empty (case 2) **do**

- For case 1, set subaisles $(n,p_1),\ldots,(k-1,p_1)$ and $(n,p_2),\ldots,(k-1,p_2)$ ($(n,p_1),\ldots,(k-1,p_1)$ for odd) as required edges and form a subtour between the subaisle sets. Apply Algorithm *turn_k-OPP* with the remaining non-empty pick-aisles;
- For case 2, set subaisles $(1,p_1),\ldots,(n-1,p_1)$ and $(1,p_2),\ldots,(n-1,p_2)$ ($(1,p_1),\ldots,(n-1,p_1)$for odd) as required edges and form a subtour between the subaisle sets. Apply Algorithm *turn_k-OPP* with the remaining non-empty pick-aisles;
- Join the depot node to the tour;

endif

else do

- Apply Algorithm *turn_2-OPP* to the problem;
- If the depot is on a middle aisle, join the depot to the tour;

endif

end

Figure 6.17: Algorithm *turn_k – OPP_modified*

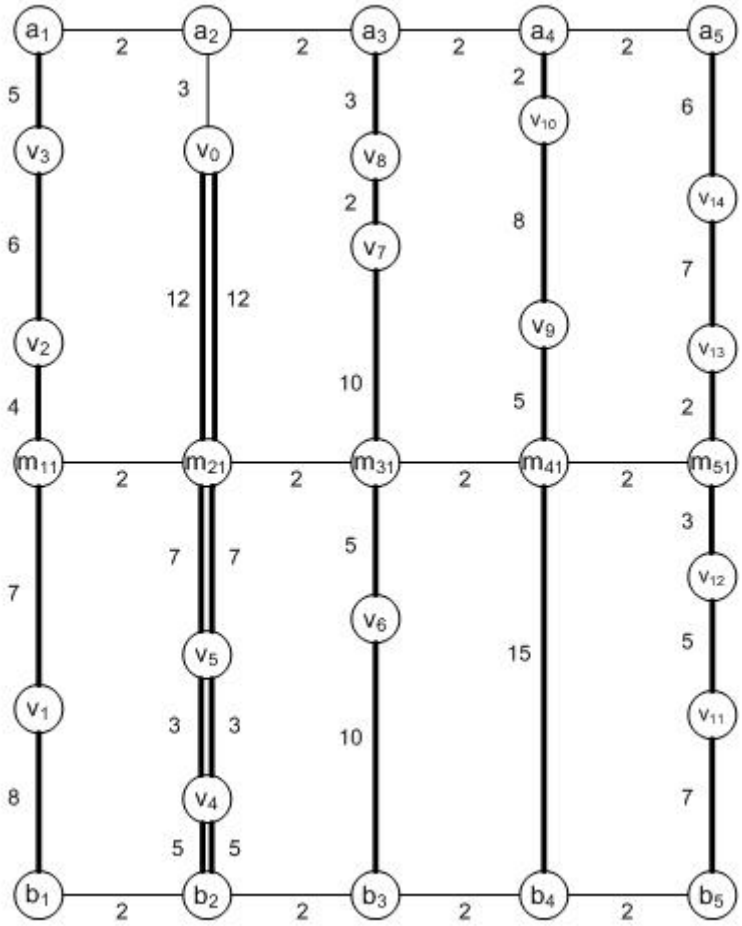

Figure 6.18: An example problem with 2 blocks, 5 non-empty pick aisles and 14 items, with required edges shown in bold

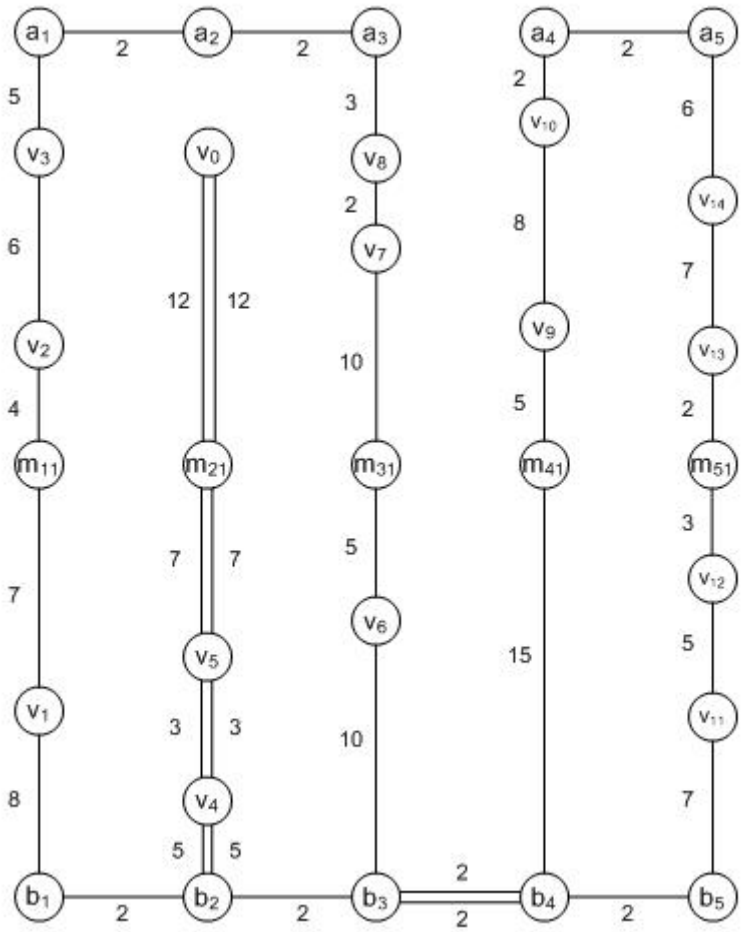

Figure 6.19: Optimal solution to the example problem in Figure 6.18, which incurs 10 turns

joining the subtours using double horizontal edges from arbitrary sides, we obtain the optimal solution in Figure 6.19. The solution has 10 turns.

The second example is illustrated in Figure 6.20, with the required edges shown in bold. As in the previous example, there are 2 blocks, 5 non-empty pick aisles, and 14 items. Since the depot is on a middle aisle, there are 5 (odd) non-empty pick aisles and subaisle (1,4) is empty, the special case in Theorem 6.16 holds. Hence we set subaisle (2,4) as a required edge along with the remaining non-empty pick aisles.

When Algorithm *turn_k-OPP* is applied to the problem, the optimal solution in Figure 6.21 is obtained. The optimal solution has 12 turns.

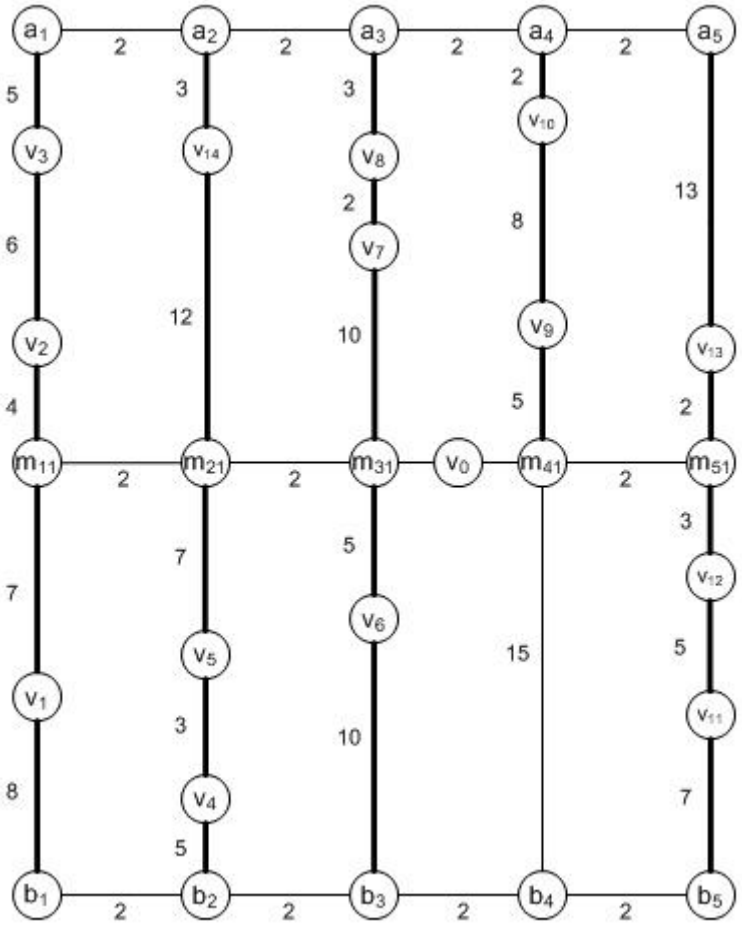

Figure 6.20: An example problem with 2 blocks, 5 non-empty pick aisles and 14 items, the required edges shown in bold

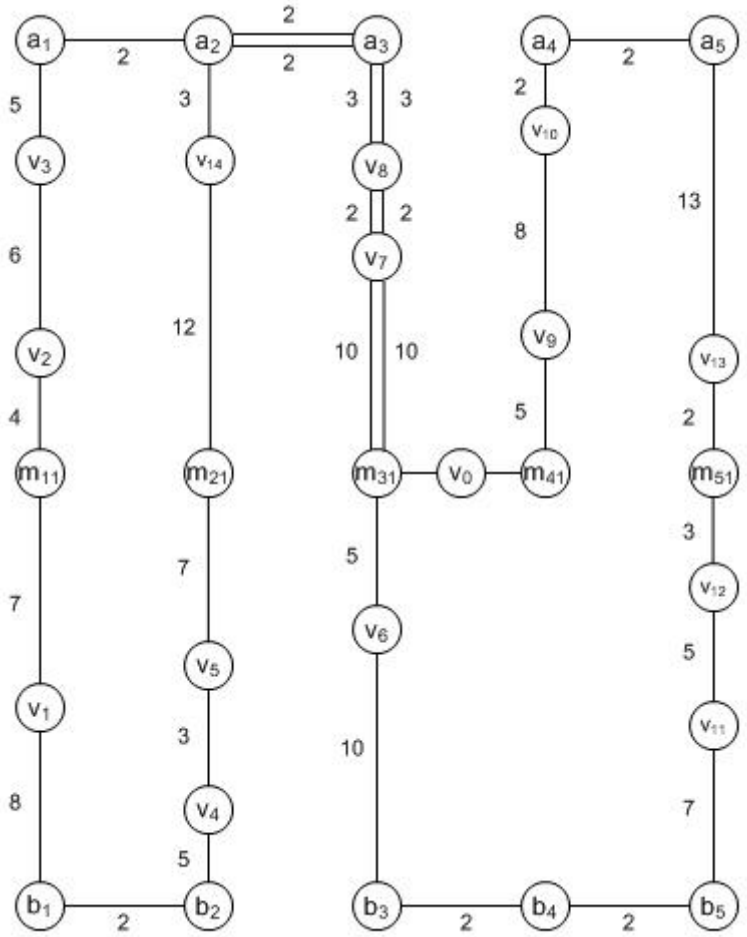

Figure 6.21: Optimal solution to the problem in figure 6.20 with 12 turns

CHAPTER 7

CONCLUSION AND FURTHER RESEARCH DIRECTIONS

The order-picking problem, due to the importance of its contribution to the overall warehousing costs, is one of the critical problems of the supply chain in the sense that providing efficient and effective solutions to it not only decreases the costs, but also the response times to the customer demands. In this study, we have considered various aspects of this problem with different assumptions, objective functions and constraints in each case.

First, we have given an overview of the problems encountered in warehouse design. We have divided these problems into three groups as strategic, tactical and operational level problems, depending on the decision level that they affect. In doing this, our aim is to provide a framework on where the order-picking problem lies among these problems. We have observed that there exists a high level of interdependency between the problems, and the order-picking problem takes up a very important part in creating the interdependencies. We have also seen that despite this strong connection among the problems, the solution approaches usually take a hierarchical approach, aiming to solve the problems in a sequential manner, and we have emphasized the need to take a more overall approach by considering the problems in a more integrated manner. The problems that are studied in the literature have also been observed to be isolated from their environment. Additionally, the strategic problems were seen to lack the level of attention they receive in the literature that they deserve.

To the best of our knowledge, there exists no discussion on the complexity of the OPP in the literature so far. Motivated by this, we have provided a detailed discussion regarding the complexity of the problem. We have surveyed the literature on the complexity of relevant problems such as the Traveling Salesman Problem, its special cases on grid graphs and series-parallel graphs as well as its different versions such as the Graphical Traveling Sales-

man Problem, Steiner Traveling Salesman Problem, the Graphical Steiner Traveling Salesman Problem, and the Steiner Tree Problem. We have distinguished the polynomially solvable cases and also given the cases where conclusions exist on their NP-completeness. For the OPP, we have given the polynomial algorithms in detail for the cases for which they exist, and we have also reviewed the heuristic procedures for the cases for which no conclusion on their complexity exists. We have conjectured that although the problem is polynomial for a fixed number of cross aisles, it is NP-hard in terms of the number of cross aisles.

The conjecture of NP-hardness implies that no polynomial algorithm solves the OPP to optimality unless $P = NP$, therefore the only way to find near-optimal solutions this problem in polynomial time is to use heuristic procedures. In the literature, heuristic procedures lack the property of robustness. In other words, although they perform well for a set of problems, their performances may worsen significantly for certain problem sets. We propose the merge-and-reach heuristic procedure in order to provide a more robust approach. The heuristic depends on using random cuts to break the OPP into subproblems, then solve the OPP corresponding to each block using the algorithm by Ratliff and Rosenthal [71]. We combine the solutions to these blocks together using the merge or reach procedures to obtain a solution for each subproblem, and combine the solutions to the subproblems, again using the merge or reach procedures, to come up with a solution for the problem. We have also proposed a 3-opt improvement over the merge-and-reach procedure and called this improvement procedure merge-and-reach$^+$.

To test the performance of the merge-and-reach and merge-and-reach$^+$ procedures, and to compare the results of the procedures with the ones in the literature, we have generated a set of random problems in line with those of Roodbergen and De Koster [73]. The procedures have come up with good results: the merge-and-reach heuristic deviates no more than 5.4% on average from the optimal in the worst case, whereas for the merge-and-reach$^+$ heuristic the maximum deviation is 2.4% on average in the worst case. When the performances of these heuristics are compared with those in the literature, the merge-and-reach heuristic has been observed to dominate the others in 64 of the 80 cases. The merge-and-reach$^+$ procedure dominates the best results in the literature, to the best of our knowledge.

For the case of the OPP with multiple pickers, we have assumed that there exist load capacities and time limits on each picker. For this problem, we have proposed an evolutionary approach

that makes use of the cluster-first, route-second and route-first, cluster-second approaches for the routing problem. In each generation, the algorithm applies the former approach α percent of the time and the latter otherwise. We have made preliminary runs on a subset of the random problems generated for the single-picker case for the two extreme cases of $\alpha = 0\%$ and $\alpha = 100\%$ to fine-tune the algorithm parameters, and compared the results. We have observed that the first setting performs better than the latter in almost all problem settings. Following this, we have combined the two approaches and tested the combined evolutionary algorithm with values of $\alpha = 25\%$ and $\alpha = 75\%$. In this case, the algorithms have come up with comparable results to each other, but the main thing to note here is that they both perform better than the extreme approaches, justifying the usage of the combined approach.

The literature on the OPP mainly focuses on minimizing the travel distance to minimize the total order-picking time. We have noted that there are other factors than travel time that contribute to the total picking time, and stated that among these, the turns (right, left or U-turns) are the ones that depend on the route on the picker. We have considered the objective of minimizing the total number of turns in an OPP, and have provided a polynomial algorithm for the case with no middle aisle and with middle aisles respectively. We have also considered the cases where the depot can be on a pick aisle or on one of the middle aisles. We have extended the algorithm to these to cases as well.

There are a number of future research directions on the problems discussed in this study. First of all, the complexity of the order-picking problem in terms of the number of cross aisles is still "open". Either a polynomial algorithm that is polynomial in terms of the number of cross aisles is to be found, or a proof of NP-completeness is required.

The merge-and-reach and the merge-and-reach$^+$ come up with near-optimal solutions for the single-picker case. The reasons of their suboptimality can be explored, which might lead to a polynomial algorithm for the OPP, or an explanation of why it is NP-complete, if it is. Additionally, the robustness of the algorithms can be tested for various positions of the depot in the warehouse, as well as non-uniform item distribution.

Due to the lack of benchmark problems for the evolutionary approach proposed for the multiple picker case, the optimal solutions or at least good lower bounds are yet to be found out to compare the performance of the algorithm with the optimal solutions or lower bounds. The computational experiments were made on instances with tight capacities and uniform item

distribution. How the results change under loose capacities and non-uniform item distribution is another research question. In addition to this, the pickers were obliged to start and end at the same depot. It is possible to find better solutions when the pickers are allowed to start and end at different depots, as detours will be avoided.

For the turn minimization problem, the effect of travel times caused by the turns can be incorporated into the problem and a solution procedure can be found for the travel time minimization problem that takes into account both the travel time and time lost due to turns.

REFERENCES

[1] Abdou, G. and El-Masry, M., *Three-dimensional random stacking of weakly heterogeneous palletization with demand requirements and stability measures*, International Journal of Production Research **38-14** (2000), 3149-3164.

[2] Aghezzaf, E., *Capacity planning and warehouse location in supply chains with uncertain demands*, Journal of the Operational Research Society **56** (2005), 453-462.

[3] Aho, A. V., Garey, M. R. and Hwang, P.K., *Rectilinear Steiner trees: Efficient special-case algorithm*, Networks **7** (1977), 37-58.

[4] Arkin, E. M., Bender, M. A., Demaine, E. D., Fekete, S. P., Mitchell, J. S. B. and Sethia, S., *Optimal covering tours with turn costs*, Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (2001), 138-147.

[5] Baiou, M. and Mahjoub, A. R., *The Steiner traveling salesman polytope and related polyhedra*, SIAM Journal of Optimization **13-2** (2002), 498-507.

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

[7] Bartholdi III, J. J. and Hackman, S. T., *Warehouse & Distribution Science*, available online at http://www2.isye.gatech.edu/˜jjb/wh/ book/editions/wh-sci-0.89.pdf (accessed January 2009).

[8] Beasley, J. A., *Route-first cluster second methods for vehicle routing*, Omega **11** (1983), 403-408.

[9] Benavent, E. and Soler, D., *The directed rural postman problem with turn penalties*, Transportation Science, **33-4** (1999), 408-418.

[10] Berger, J. and Barkaoui, M. A., *New hybrid genetic algorithm for the capacitated vehicle routing problem*, Journal of the Operational Research Society **54** (2003), 1254-1262.

[11] Bodin, L. and Kursh, S., *A detailed description of a computer system for the routing and scheduling of street sweepers*, Computers & Operations Research **6** (1978), 181-198.

[12] Bodner, D. A., Govindaraj, T., Karathur K. N., Zerangue, N. F. and McGinnis, L. F., *A process model and support tools for warehouse design*, Proceedings of the 2002 Industrial Engineering Reseach Conference (2002).

[13] Burkard, R. E., Deineko, V. G., Van Dal, R., Van Der Veen, J. A. A. and Woeginger, G. J., *Well-solvable special cases of the traveling salesman problem: a survey*, Society of Industrial and Applied Mathematics Rev. **40-3** (1998), 496-546.

[14] Caldwell, T., *On finding minimal routes in a network with turn penalties*, Communications of the ACM **4-2** (1961), 107-108.

[15] Chopra, S. and Rao, M. R., *The Steiner tree problem I: Formulations, compositions and extension of facets*, Mathematical Programming **64** (1994), 209-229.

[16] Chopra, S. and Rao, M. R., *The Steiner tree problem II: Properties and classes of facets*, Mathematical Programming **64** (1994), 231-246.

[17] Clossey, J., Laporte, G. and Soriano, P., *Solving arc routing problems with turn penalties*, Journal of the Operational Research Society **52** (2001), 433-439.

[18] Corberan, A., Marti, R., Martinez, E. and Soler, D., *The rural postman problem on mixed graphs with turn penalties*, Computers & Operations Research **29** (2002), 887-903.

[19] Cormier, G., *Operational Research Methods for Efficient Warehousing*, in *Logistics Systems* (ed.s Langevin, A. and Riopel, D.) (2005), Springer Science + Business Media, Inc., 93-122.

[20] Cormier, G. and Gunn, E. A., *Modelling and analysis of capacity expansion planning in warehousing*, Journal of the Operational Research Society **50-1** (1999), 52-59.

[21] Cornuejols, G., Fonlupt, J. and Naddef, D., *The traveling salesman on a graph and some integer related polyhedra*, Mathematical Programming **33** (1985), 1-27.

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

[23] De Boer, J. W., *Approximate models and solution approaches for the vehicle routing problem with multiple use of vehicles and time windows*, MSc. Thesis, 2008, METU - Ankara.

[24] De Koster, R., Le-Duc, T. and Roodbergen, K. J., *Design and control of warehouse order picking: a literature review*, European Journal of Operational Research **182** (2007), 481-501.

[25] De Koster, R., Le-Duc, T. and Yugang, Y., *Optimal storage rack design for a 3-dimensional compact AS/RS*, International Journal of Production Research **46-6** (2008), 1495-1514.

[26] De Koster, R. and Van Der Poort, E., *Routing orderpickers in a warehouse: a comparison between optimal and heuristic solutions*, IIE Transactions **30** (1998), 469-480.

[27] Demir, E., *Analysis of evolutionary algorithms for constrained routing problems*, MSc. Thesis, 2004, METU - Ankara.

[28] Edmonds, J. and Johnson, E. L., *Matching, Euler tours and the Chinese postman problem*, Mathematical Programming **5** (1973), 88-124.

[29] Esbensen, H., *Computing near-optimal solutions to the Steiner problem in a graph using a genetic algorithm*, Networks **26-4** (1995), 173-186.

[30] Fonlupt, J. and Nachef, A., *Dynamic programming and the graphical traveling salesman problem*, Journal of the Association for Computing Machinery **40-5** (1993), 1165-1187.

[31] Foulds, L. R., Hamacher, H. W., Schöbel, A. and Yamaguchi, T., *On center cycles in grid graphs*, Annals of Operations Research **122** (2003), 163-175.

[32] Gademann, N. and Van De Velde, S., *Order batching to minimize total travel time in a parallel-aisle warehouse*, IIE Transactions **37** (2005), 63-75.

[33] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, (1979), W. H. Freeman and Company.

[34] Garey, M. R. and Johnson, D. S., *The rectilinear Steiner tree problem is NP-complete*, SIAM Journal of Applied Mathematics **32-4** (1977), 826-834.

[35] Gendreau M., Potvin, J. Y., Bräysy, O., Hasle, G. and Lokketangen, A., *Metaheuristics for the vehicle routing problem and its extensions: A categorized bibliography*, in *The Vehicle Routing Problem*, (eds. B. Golden et al.) (2008), Springer Science + Business Media.

[36] Gendreau, M., Laporte, G. and Yelle, S., *Efficient routing of service vehicles*, Engineering Optimization **28** (1997), 263-271.

[37] Geng, Y., Li, Y. and Lim, A., *A very large-scale neighborhood search approach to the capacitated warehouse routing problem*, Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence (2005).

[38] Georgia Institute of Technology, Concorde TSP Solver (2009). http://www.tsp.gatech.edu/concorde.html (last accessed June 12, 2009).

[39] Goemans, M. X., *The Steiner tree polytope and related polyhedra*, Mathematical Programming **63** (1994), 157-182.

[40] Goetschalckx, M. and Ratliff, H. D., *Optimal lane depths for single and multiple products in block stacking storage systems*, IIE Transactions **23** (1991), 245-258.

[41] Gray, A. E., Karmakar, U. S. and Seidmann, A., *Design and operation of an order-consolidation warehouse: models and application*, European Journal of Operational Research **58** (1992), 14-36.

[42] Grötschel, M., Martin, A. and Weismantel, R., *Routing in grid graphs by cutting planes*, Mathematical Methods of Operations Research **41** (1995), 255-275.

[43] Gu, J., Goetschalckx M. and McGinnis, L. F., *Research on warehouse operation: a comprehensive review*, European Journal of Operational Research **177** (2007), 1-21.

[44] Haghighat, A. T., Faez, K., Dehghan, M., Mowlaei. and Ghahremani, Y., *A genetic algorithm for Steiner tree optimization with multiple constraints using Prüfer number*, EurAsia-ICT 2002 (2002), 272-280.

[45] Hall, R. W. H., *Distance approximations for routing manual pickers in a warehouse*, IIE Transactions **25** (1993), 76-87.

[46] Hanan, M., *On Steiner's problem with rectilinear distance*, SIAM Journal on Applied Mathematics **14** (1966), 255-265.

[47] Ho, J. M., Vijayan, G. and Wong, C. K., *New algorithms for the rectilinear Steiner tree problem*, IEEE Transactions on Computer-Aided Design **9-2** (1990), 185-193.

[48] Holland, J. H., *Adaptation in natural and artificial systems*, (1975), The University of Michigan Press, Ann Arbor, MI.

[49] Itai, A., Papadimitriou, C. H. and Szwarcfiter, J. L., *Hamilton paths in grid graphs*, SIAM Journal of Computing **11-4** (1982), 676-686.

[50] Jaszkiewicz, A. and Kominek, P., *Genetic local search with distance preserving recombination operator for a vehicle routing problem*, European Journal of Operational Research **151** (2003), 352-364.

[51] Johnson, M. E., *The impact of sorting strategies on automated sortation system performance*, IIE Transactions **30** (1998), 67-77.

[52] Johnson, D. S. and Papadimitriou, C. H., *Computational Complexity*, in *The Traveling Salesman Problem* (ed.s Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G. and Shmoys, D. B.) (1985), John Wiley & Sons Ltd., 37-85.

[53] Kabadi, S. N., *Polynomially Solvable Cases of the TSP* in *The Traveling Salesman Problem and Its Variations*, (ed.s Gutin, G. and Punnen A. P.) (2002), Kluwer Academic Publishers, 489-583.

[54] Kapsalis, A., Rayward-Smith, V. J. and Smith, G. D., *Solving the graphical Steiner tree problem using genetic algorithms*, Journal of the Operational Research Society **44-4** (1993), 397-406.

[55] Karp, R. M., *Reducibility among combinatorial problems*, in *Complexity of Computer Calculations* (ed.s Miller, R. E. and Thatcher, J. W.) (1972), Plenum Press, 85-103.

[56] Larson, T. N., March, H. and Kusiak, A., *A heuristic approach to warehouse layout with class-based storage*, IIE Transactions **29** (1997), 337-348.

[57] Le-Duc, T. and De Koster, R., *Travel distance estimation and storage zone optimization in a 2-block class-based storage strategy warehouse*, International Journal of Production Research **43-17** (2005), 3561-3581.

[58] Liu, C. M., *Clustering techniques for stock location and order-picking in a distribution center*, Computers & Operations Research **26** (1999), 989-1002.

[59] Makris, P. A. and Giakoumakis, I. G., *k-Interchange heuristic as an optimization procedure for material handling applications*, Applied Mathematical Modelling **27** (2003), 345-358.

[60] Malmborg, C. J. and Krishnakumar, B., *Optimal storage assignment policies for multi-address warehousing systems*, IEEE Transactions on Systems Management and Cybernetics **19-1** (1989), 197-204.

[61] Margot, F., Prodon, A. and Liebling, T. M., *Tree polytope on 2-trees*, Mathematical Programming **63** (1994), 183-191.

[62] McGinnis, L. F., Goetschalckx, M., Sharp, G., Bodner, D. and Govindaraj, T., *Rethinking warehouse design research*, Proceedings of the 2000 International Material Handling Research Colloquium (2000).

[63] Meller, R. D., *Optimal order-to-lane assignments in an order accumulation/sortation system*, IIE Transactions **29**, 293-301.

[64] Mester, D. and Bräysy, O., *Active guided evolution strategies for the large-scale capacitated vehicle routing problems*, Computers & Operations Research **34** (2007), 508-517.

[65] Monma, C. L., Munson, B. S. and Pulleyblank, W. R., *Minimum-weight two-connected spanning networks*, Mathematical Programming **46** (1990), 153-171.

[66] Petersen, C. G., *Considerations in order picking zone configuration*, International Journal of Operations & Production Management **22-7** (2002), 793-805.

[67] Petersen, C. G. and Aase, G., *A comparison of picking, storage and routing policies in manual order picking*, International Journal of Production Economics **92** (2004), 11-19.

[68] Prim, R. C., *Shortest connection networks and some generalizations*, The Bell System Technical Journal **36** (1957), 1389-1401.

[69] Prins, C., *A simple and effective evolutionary algorithm for the vehicle routing problem*, Computers & Operations Research **31** (2004), 1985-2002.

[70] Queirolo, F., Tonelli, F., Schenone, M., Nan, P. and Zunino, I., *Warehouse layout design: minimizing travel time with a genetic and simulative approach - methodology and case study*, Proceedings of the 14th European Simulation Symposium (2002).

[71] Ratliff, H. D. and Rosenthal, A. S., *Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem*, Operations Research **31** (1983), 507-521.

[72] Roodbergen, K. J. and De Koster, R., *Routing order-pickers in a warehouse with a middle aisle*, European Journal of Operations Research **133** (2001), 32-43.

[73] Roodbergen, K. J. and De Koster, R., *Routing methods for warehouses with multiple cross aisles*, International Journal of Production Research **39-9** (2001), 1865-1883.

[74] Rouwenhorst, B., Reuter, B., Stockrahm, V., Van Houtum, G. J., Mantel, R. J. and Zijm, W. H. M., *Warehouse design and control: framework and literature review*, European Journal of Operational Research **122** (2000), 515-533.

[75] Roy, S. and Rousseau, J. M., *The capacitated Canadian postman problem*, Information Systems and Operational Research **27** (1989), 58-73.

[76] Salman, A. N. M., Baskoro, E. T. and Broersma, H. J., *A note concerning Hamilton cycles in some classes of grid graphs*, Proceedings ITB Sains dan Teknologi **35A-1** (2003), 65-70.

[77] Sönmez, M., *An evolutionary approach to TSP: Crossover with conventional heuristics*, MSc. Thesis, 2003, METU - Ankara.

[78] Takamizawa, K., Nishizeki, T. and Saito, N., *Linear-Time Computability of Combinatorial Problems Series-Parallel Graphs*, Journal of the Association for Computing Machinery **29-3** (1982), 623-641.

[79] Tompkins J. A., White, J. A., Bozer, Y. A. and Tanchoco, J. M. A., *Facilities Planning* (2003), John Wiley & Sons, NJ.

[80] Umans, C. and Lenhart, W., *Hamiltonian cycles in solid grid graphs*, Proceedings of the Annual Symposium on Foundations of Computer Science (1997), 496-505.

[81] Van Den Berg, J. P., Sharp, G. P., Gademann, A. J. R. M. and Pochet, Y., *Forward-reserve allocation in a warehouse with unit-load replenishments*, European Journal of Operational Research **111** (1998), 98-113.

[82] Vaughan, T. S. and Petersen, C. G., *The effect of warehouse cross aisles on order picking efficiency*, International Journal of Production Research **37-4** (1999), 881-897.

[83] Winter, P., *Steiner problem in networks: a survey*, Networks **17** (1987), 129-167.

[84] Winter, P. and Smith, J. M., *Path-distance heuristics for the Steiner problem in undirected networks*, Algorithmica **7** (1992), 309-327.

[85] Won, J. and Olafsson, S., *Joint order batching and order picking in warehouse operations*, International Journal of Production Research **43-7** (2005), 1427-1442.

[86] Yang, B., *An evolution algorithm for the rectilinear Steiner tree problem*, ICCSA 2005 (2005), 241-249.

APPENDIX A

ALTERNATIVE HEURISTIC SOLUTIONS TO THE PROBLEM IN FIGURE 3.1

A.1 Optimal Solution

Figure A.1 illustrates the optimal route to the problem given in Figure 3.1. The total length of the optimal solution is 164 units.

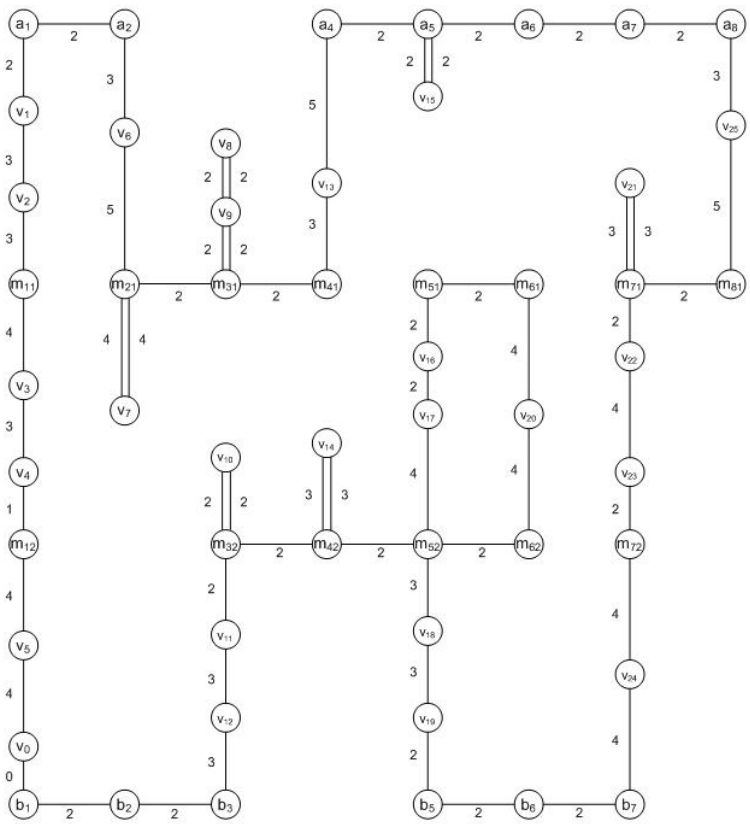

Figure A.1: Optimal solution of the example problem in Figure 3.1, with a total travel distance of 164 units

A.2 S-shape Heuristic Solution

Figure A.2 shows the resulting route from the S-shape heuristic to the problem given in Figure 3.1. The total length of the optimal solution is 192 units.

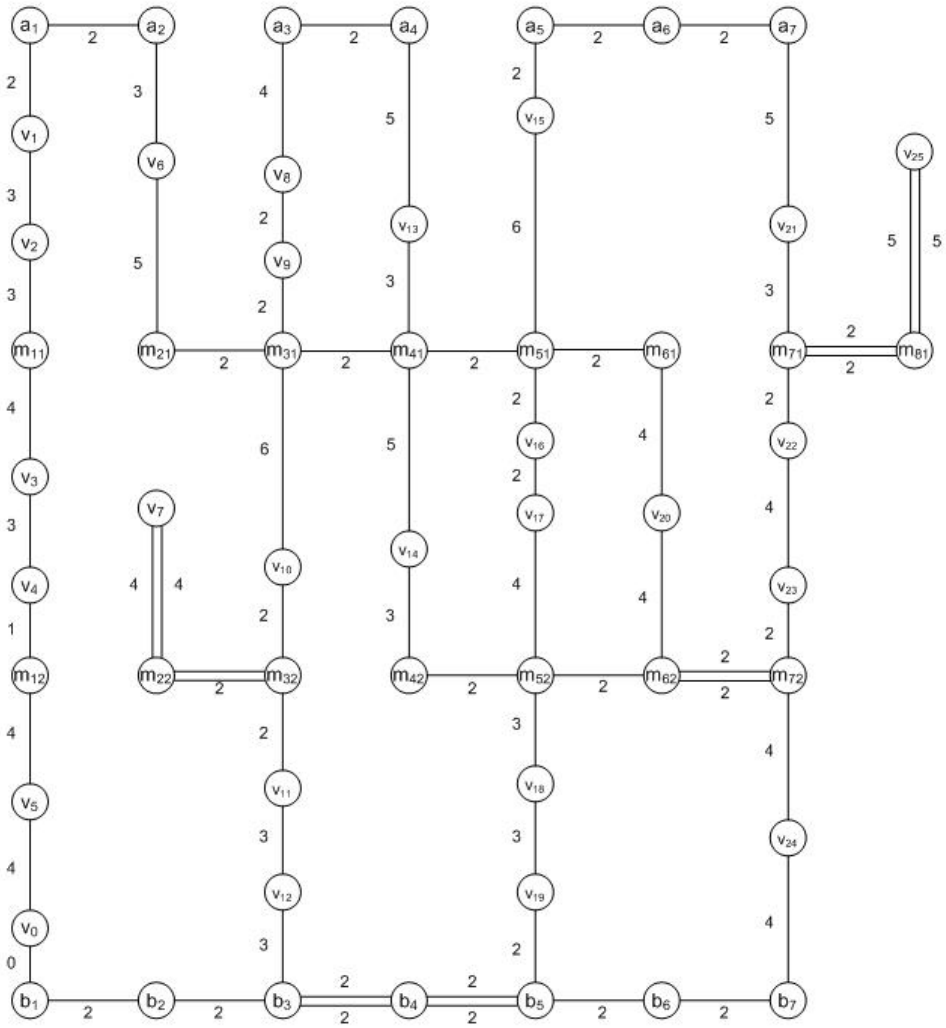

Figure A.2: S-shape heuristic solution to the example problem in Figure 3.1, with a total travel distance of 192 units

A.3 Largest Gap Heuristic Solution

Figure A.3 illustrates the resulting route from the largest gap heuristic to the problem given in Figure 3.1. The total length of the optimal solution is 190 units.

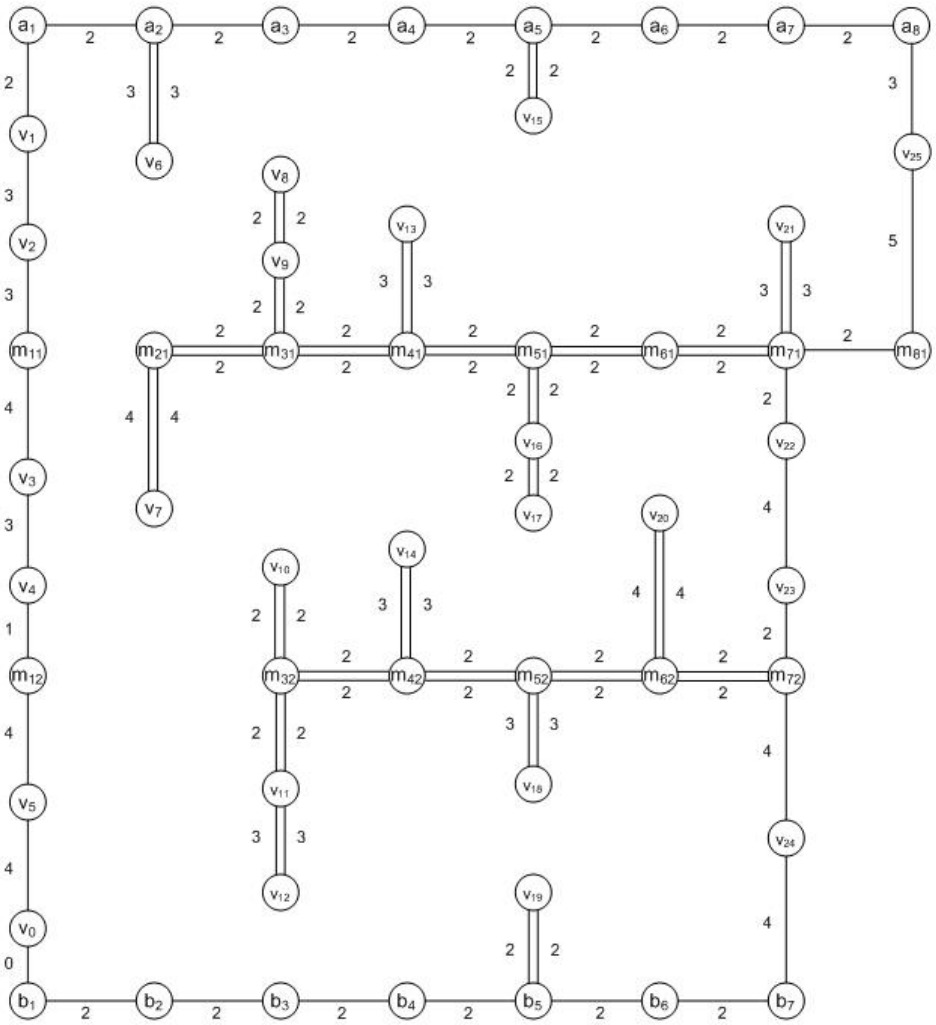

Figure A.3: Largest gap heuristic solution to the example problem in Figure 3.1, with a total travel distance of 190 units

A.4 Aisle-by-Aisle Heuristic Solution

Figure A.4 illustrates the resulting route from the aisle-by-aisle heuristic (due to Vaughan and Petersen [82]) to the problem given in Figure 3.1. The total length of the optimal solution is 190 units.

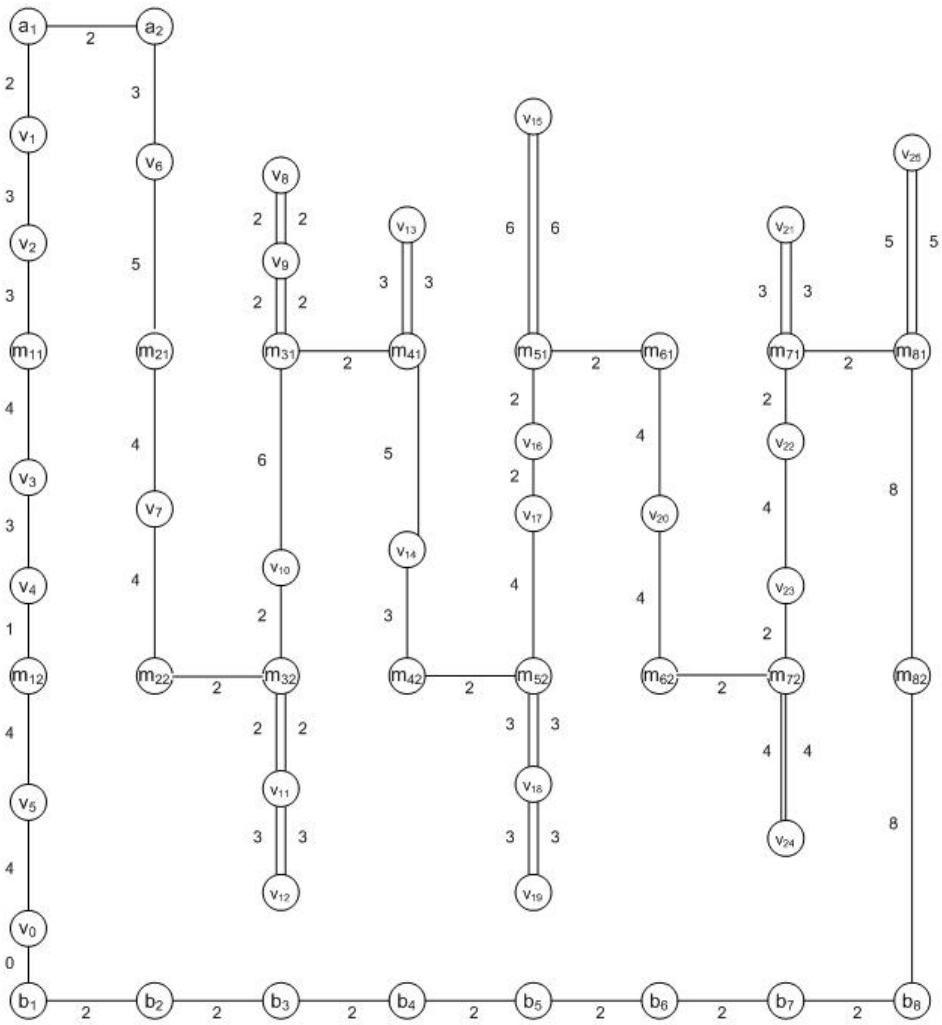

Figure A.4: Aisle-by-aisle heuristic solution for the problem in Figure 3.1, with a total travel distance of 190 units

A.5 Combined Heuristic Solution

Figure A.5 depicts the resulting route from the combined heuristic (due to Roodbergen and De Koster [73]) to the problem given in Figure 3.1. The total length of the optimal solution is 182 units.

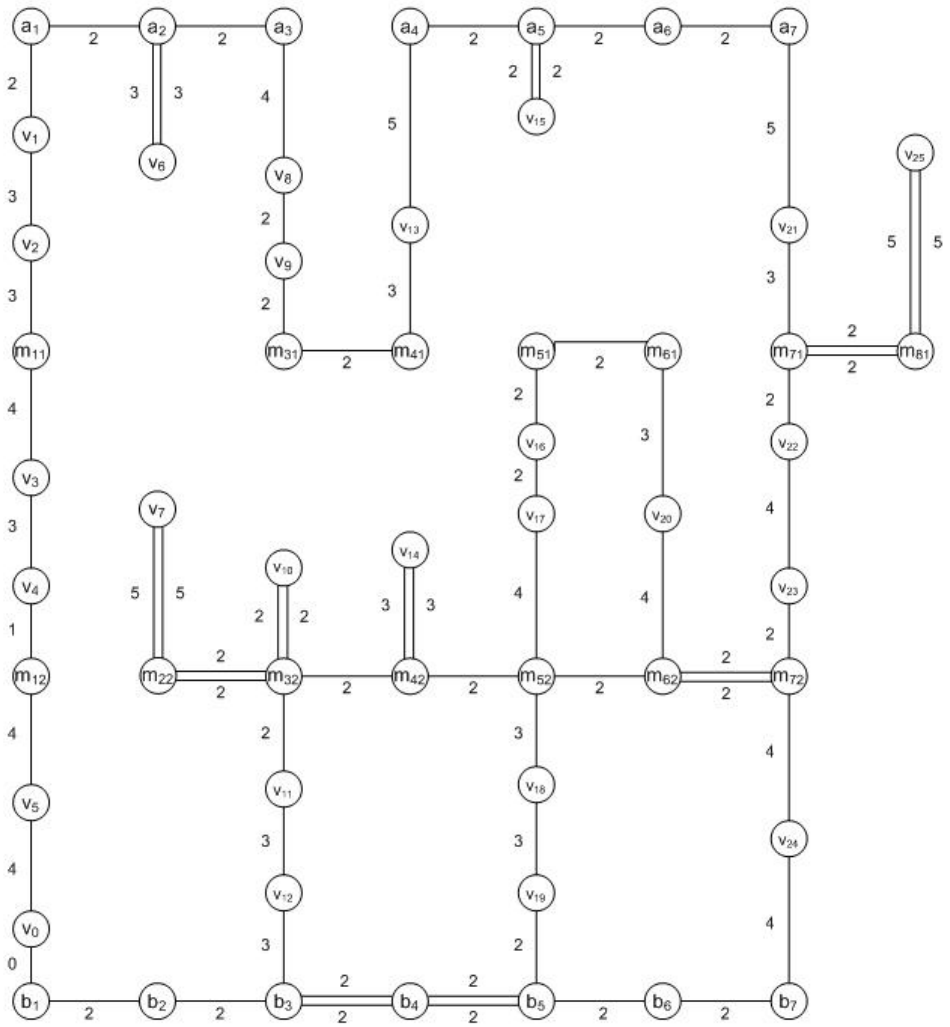

Figure A.5: Combined heuristic solution for the example problem in Figure 3.1, with a total travel distance of 182 units

APPENDIX B

NP-COMPLETENESS OF THE TRAVELING SALESMAN PROBLEM

Johnson and Papadimitriou [52] start their proof of NP-completeness of the TSP with the following fact: In order for the TSP to be polynomially solvable, its corresponding decision problem TSP DECISION should be polynomially solvable as well. The converse is also true, that is, if the decision problem is NP-complete, so is the original problem. TSP DECISION can be defined as follows.

PROBLEM: TSP DECISION
INSTANCE: Integer $n \geq 3$ and $n \times n$ integer matrix $C = (c_{ij})$, where each $c_{ij} \geq 0$, and an integer $B \geq 0$.
QUESTION: Is there a cyclic permutation Π of the integers from 1 to n such that

$$\sum_{i=1}^{n} c_{i\pi(i)} \leq B?$$

The proof begins with a polynomial transformation of the NP-complete INTEGER PROGRAMMING to the problem of 0-1 PROGRAMMING. The following definitions of both problems are from [52].

PROBLEM: INTEGER PROGRAMMING
INSTANCE: An $m \times n$ integer matrix $A = (a_{ij})$, an m-vector $b = (b_1, \ldots, b_m)$ of integers.
QUESTION: Is there an n-vector x with $x_j \geq 0$ such that $Ax = b$, i.e. $\sum_{j=1}^{n} a_{ij}x_j = b_i$, $1 \leq i \leq m$?

PROBLEM: 0-1 PROGRAMMING
INSTANCE: An $m \times n$ integer matrix $A = (a_{ij})$, an m-vector $b = (b_1, \ldots, b_m)$ of integers.
QUESTION: Is there an n-vector x with $x_j \in \{0, 1\}$ such that $Ax = b$, i.e. $\sum_{j=1}^{n} a_{ij}x_j = b_i$, $1 \leq$

$i \leq m$?

The transformation by [52] of the instance I of INTEGER PROGRAMMING to an instance *Z(I)* of 0-1 PROGRAMMING is is as follows: Each variable x of I is transformed into p_j 0-1 variables $x_0, x_1, \ldots, x_{p_j-1}$, where $x = \sum_{k=0}^{p_j-1} x_k$. Since every integer can be written in this form, I and *Z(I)* are equivalent. That is, INTEGER PROGRAMMING has a solution only if 0-1 PROGRAMMING does. Since the first problem is NP-complete, so is the second one. The second transformation by [52] is from 0-1 PROGRAMMING to EXACT COVER. Before proceeding, the problem of EXACT COVER can be stated as follows:

PROBLEM: EXACT COVER
INSTANCE: A family $F = \{S_1, S_2, \ldots, S_n\}$ of subsets of a set $U = \{u_1, u_2, \ldots, u_m\}$ corresponding to the rows of matrix A and the columns of A are the characteristic vectors of S_j.
QUESTION: Is there a subfamily $C \subseteq F$ such that each $u_i \in U$ is in exactly one of the subsets $S_j \in C$? C corresponds to a 0-1 solution x for $Ax = b$.

The latter problem is a special case of the first in that A is a 0-1 matrix and the vector b is all ones. By transformation of the A matrix, variable re-definition and transformation of the b values, Johnson and Papadimitriou show that 0-1 PROGRAMMING is a special case of EXACT COVER, implying that EXACT COVER is also NP-complete.

The problem of HAMILTONIAN CYCLE asks whether the graph G is Hamiltonian. By constructing an instance of this problem that has a solution only if the corresponding EXACT COVER has, one transforms EXACT COVER into HAMILTONIAN CYCLE. This implies that answering the question of whether a general graph contains a Hamiltonian cycle is NP-complete. The last problem, HAMILTONIAN CYCLE FOR GRID GRAPHS, tries to find a Hamiltonian cycle on a general grid graph. It is shown in [52] that every general graph can be transformed into a grid graph by an embedding procedure, meaning that HAMILTONIAN CYCLE FOR GRID GRAPHS generalizes the problem HAMILTONIAN CYCLE and hence is NP-complete. This last problem is a special case of TSP, leading to the conclusion that the TSP is NP-complete.

The path of transformation from HAMILTONIAN CYCLE FOR GRID GRAPHS to TSP is shown in Figure B.1, borrowed from [52]. In the figure, each arrow denotes that the first problem generalizes the second. The fact that HAMILTONIAN CYCLE FOR GRID GRAPHS

is NP-complete in general graphs implies that all the problems in Figure B.1 are also NP-complete for general graphs. The figure can be traced as follows: The problem of Hamiltonian Cycle in Grid Graphs is generalized by Rectilinear TSP and Euclidean TSP, which are special cases of the TSP in the plane, using rectilinear and Euclidean distance metrics between the nodes of the graph G respectively. These two problems are planar, therefore satisfy the triangle inequality property and the distance metrics are symmetric. In other words, for each pair of nodes i and j, $d_{ij} = d_{ji}$. This implies that the more generalized problem of Symmetric Triangle Inequality TSP, where the symmetric distance metric and triangle inequality hold regardless of the distance metric. This leads us to the more general problem of Symmetric TSP, where the triangle inequality does not necessarily hold, which generalizes Symmetric Triangle Inequality TSP, and is NP-complete. The General Asymmetric TSP generalizes Symmetric TSP in the sense that the symmetric distance property may not hold. This concludes the proof, as General Asymmetric TSP is also NP-complete.

Another path in Figure B.1 that proves NP-completeness of the problem TSP starts from the problem of Hamiltonian Cycle in Bipartite Planar Graphs, which seeks whether a planar and bipartite graph G contains a Hamiltonian cycle. As grid graphs are bipartite and planar, the problem generalizes Hamiltonian Cycle in Grid Graphs, and is NP-complete. The more general problem of Hamiltonian Cycle, which was defined previously, generalizes Hamiltonian Cycle in Bipartite Planar Graphs and is also NP-complete. The directed version of this problem, Directed Hamiltonian Cycle, which tries to find whether a directed graph has a Hamiltonian cycle, is a more generalized version, implying that Directed Hamiltonian Cycle is NP-complete. The Asymmetric Triangle Inequality TSP tries to find whether a Hamiltonian cycle exists in a directed graph satisfying triangle inequality. This problem generalizes the problems of Directed Hamiltonian Cycle and Symmetric Triangular Inequality TSP, leading to the conclusion that the Asymmetric Triangle Inequality TSP is also NP-complete. If the triangle inequality does not necessarily hold for this problem, we end up with the general TSP, providing another proof that it is NP-complete.

Garey and Johnson [33], on the other hand, prove the NP-completeness of the problem TSP starting with the problem SATISFIABILITY, stated below.

PROBLEM: SATISFIABILITY

INSTANCE: A set U of variables and a collection C of clauses over U.

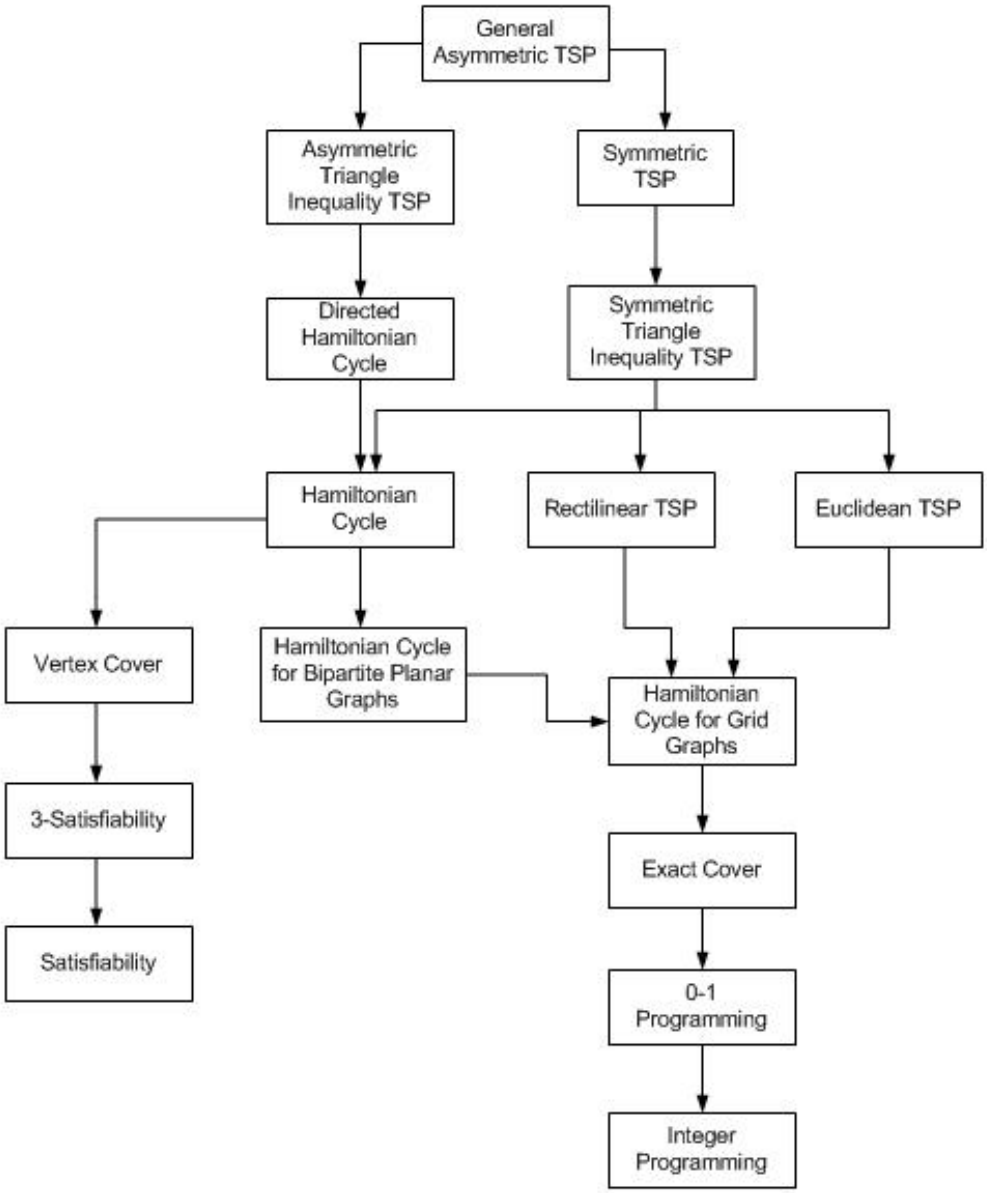

Figure B.1: Transformation from Hamiltonian Cycle for Grid Graphs to the General TSP [52]

QUESTION: Is there a satisfying truth assignment for C?

This problem is the basic NP-complete problem, and Garey and Johnson [33] transform it to the following 3-SATISFIABILITY problem.

PROBLEM: 3-SATISFIABILITY
INSTANCE: Collection $C = \{c_1, \ldots, c_m\}$ of clauses on a set U of variables such that $|c_i| = 3$ for $1 \leq i \leq m$.
QUESTION: Is there a truth assignment for U that satisfies all the clauses in C?

The transformation in [33] is by expanding the variable set U, defining literals from the variables in U and redefining the clauses. A further transformation in [33] is made from 3-SATISFIABILITY to VERTEX COVER. The latter problem can be stated verbally as finding a subset of vertices on a graph so that each vertex not in the subset can be reached from those in the subset by using at most a single edge. More formally,

PROBLEM: VERTEX COVER
INSTANCE: A graph $G=(V,E)$ and a positive integer $K \leq |V|$.
QUESTION: Is there a vertex cover of size K or less for G, that is, a subset $V' \subseteq V$ such that $V' \leq |K|$ and, for each edge $(uv) \in E$, at least one of u and v belongs to V'?

The vertex cover instance is given in [33] with variables and clauses as nodes and edges between corresponding variables and clauses. It is shown that there is a vertex cover of this graph only if the 3-SATISFIABILITY problem has a solution. The transformation to the HAMILTONIAN CIRCUIT problem is by using so-called *cover-testing components* in the graph. since HAMILTONIAN CIRCUIT can be polynomially transformed into the TSP, which implies that the TSP is NP-complete. The left part of Figure B.1 shows the transformation provided by [33], from Satisfiability to the General Asymmetric TSP.

APPENDIX C

COMPARISON OF HEURISTIC PROCEDURES FOR *k*-OPP

Table C.1, borrowed from [73] gives the resulting average travel times by the largest gap, S-shape and aisle-by-aisle heuristic procedures, whereas Table C.2, also borrowed from [73], presents the average travel times by the combined and combined$^+$ heuristics as well as the average optimal travel times. The optimal travel times are obtained using a branch-and-bound algorithm for the Traveling Salesman Problem. It is reported in [73] that for each instance, average calculation time is less than 0.1 seconds on a 350 MHz computer.

It can be observed from Tables C.1 and C.2 that in 74 of the 80 problem settings, the combined$^+$ heuristic gives the best average results. It has already been observed by Roodbergen and De Koster [73] that the combined heuristic performs better than the S-shape heuristic, as the set of solutions for the S-shape heuristic is a subset of those of the combined heuristic. Additionally, the percentage difference between the S-shape and combined solutions decreases as the cross aisles or the number of items are increased, owing to the fact that in both cases, complete traversal of the subaisles becomes desirable compared to partial traversal, which increases the performance of the S-shape heuristic.

Roodbergen and De Koster [73] also indicate that in warehouses with a single block, the aisle-by-aisle, combined and combined$^+$ heuristics give identical results, as the main idea in each of the combined and combined$^+$ heuristics is that the aisle-by-aisle heuristic is applied to each block in a problem. Therefore, with a single block, the same solutions are obtained. As the number of blocks are increased, the total length of the route decreases to some extent, but after some point, the total travel distance starts to increase. This is because traversal of the cross aisles starts to contribute significantly to the overall length of the tour, and is in line with the findings of Vaughan and Petersen [82].

Table C.1: Resulting average travel times (in seconds) by the largest gap, S-shape and aisle-by-aisle heuristic procedures for the problem set of Roodbergen and De Koster [73], with the best heuristic results indicated in bold (2,000 instances for each setting)

				Number of blocks									
Method	No. of aisles	Length	No. of items	1	2	3	4	5	6	7	8	9	10
Largest gap	7	10	10	**146.6**	156.9	164.6	169.3	176.3	181.9	187.4	194.7	201.9	209.3
	7	10	30	208.6	240.9	273.9	303.5	330.3	348.6	363.4	379.6	394.4	408.4
	15	10	10	**227.3**	265.2	287.2	296.2	305.7	312.2	317.1	324.6	331.4	338.4
	15	10	30	357.5	413.5	484.5	552.1	614.6	660.1	692.3	724.7	750.1	776.3
	7	30	10	**295.1**	259.9	250.7	246.3	247.4	250.3	254.5	259.7	264.9	270.6
	7	30	30	451.7	424.7	425.7	435.9	446.9	456.1	464.1	472.2	478.4	488.7
	15	30	10	**401.0**	377.6	379.1	377.2	379.5	382.4	386.6	390.6	395.2	400.6
	15	30	30	**715.6**	646.0	665.4	705.2	746.3	779.9	805.0	826.5	842.8	863.7
S-shape	7	10	10	165.1	145.7	152.6	155.7	161.4	167.7	174.6	181.8	188.6	196.4
	7	10	30	203.5	210.3	250.1	253.4	278.3	287.8	301.0	311.9	322.3	332.4
	15	10	10	266.2	224.6	245.0	252.6	261.2	270.2	278.9	288.3	295.5	304.6
	15	10	30	391.3	359.5	431.1	422.8	478.3	491.1	518.4	539.0	557.8	576.0
	7	30	10	353.1	276.2	256.5	245.0	242.5	243.4	247.1	251.7	257.1	262.5
	7	30	30	452.0	426.8	438.2	420.1	427.7	423.4	427.0	429.7	432.9	437.1
	15	30	10	517.6	376.9	361.0	349.4	347.6	350.1	354.7	360.4	366.4	372.7
	15	30	30	833.3	686.0	688.2	636.4	663.4	653.6	666.5	675.2	684.4	695.1
Aisle-by-aisle	7	10	10	148.5	144.3	153.7	164.6	177.5	189.9	205.4	216.4	230.3	245.6
	7	10	30	**192.1**	207.2	227.0	246.7	268.5	289.5	315.7	333.6	357.6	383.8
	15	10	10	235.2	220.7	229.4	241.0	255.1	268.9	286.3	298.4	313.6	330.9
	15	10	30	**356.7**	349.3	**369.1**	392.5	421.3	449.8	486.3	509.8	542.1	578.6
	7	30	10	304.7	268.0	268.4	276.5	287.5	299.6	313.0	325.7	339.0	351.8
	7	30	30	**418.8**	413.9	422.8	438.5	457.2	477.7	500.4	522.5	544.1	565.8
	15	30	10	427.2	362.0	358.6	366.8	378.3	391.7	406.6	420.6	435.3	449.7
	15	30	30	732.7	648.8	642.8	658.6	682.0	709.1	739.7	769.4	798.9	828.9

Table C.2: Resulting average travel times (in seconds) by the combined, and combined$^+$ heuristic procedures and average optimal times for the problem set of Roodbergen and De Koster [73], with the best heuristic results indicated in bold (2,000 instances for each setting)

				Number of blocks									
Method	No. of aisles	Length	No. of items	1	2	3	4	5	6	7	8	9	10
Combined	7	10	10	148.5	134.6	145.4	151.2	158.2	165.5	172.9	180.4	187.5	195.6
	7	10	30	**192.1**	196.5	236.1	240.4	267.0	277.7	292.2	304.2	315.4	326.3
	15	10	10	235.2	208.6	235.3	246.7	257.1	267.2	276.6	286.6	294.1	303.6
	15	10	30	**356.7**	324.6	402.5	399.5	459.0	474.8	504.2	526.8	547.1	566.7
	7	30	10	304.7	243.4	235.1	231.2	232.5	236.1	241.5	247.4	253.5	259.7
	7	30	30	**418.8**	386.2	397.4	382.2	394.9	394.7	401.7	407.5	413.8	420.7
	15	30	10	427.2	330.1	332.5	331.8	334.8	340.8	347.6	355.0	362.0	369.2
	15	30	30	732.7	584.6	605.6	569.7	609.2	608.6	628.2	642.5	656.5	671.3
Combined$^+$	7	10	10	148.5	**133.2**	**136.2**	**140.2**	**146.0**	**152.1**	**159.4**	**165.9**	**173.3**	**180.8**
	7	10	30	**192.1**	**196.0**	**224.7**	**232.0**	**244.2**	**253.2**	**260.7**	**268.9**	**276.6**	**283.9**
	15	10	10	235.2	**207.0**	**214.5**	**220.8**	**227.6**	**234.3**	**242.0**	**249.3**	**257.0**	**264.7**
	15	10	30	**356.7**	**323.6**	**373.3**	**379.6**	**401.4**	**415.3**	**425.5**	**437.2**	**447.6**	**457.2**
	7	30	10	304.7	**235.4**	**219.3**	**214.9**	**216.2**	**219.8**	**225.2**	**230.8**	**237.2**	**243.4**
	7	30	30	**418.8**	**381.7**	**379.3**	**365.8**	**362.7**	**360.6**	**361.6**	**363.6**	**366.6**	**370.6**
	15	30	10	427.2	**323.1**	**305.3**	**300.5**	**301.0**	**304.3**	**310.1**	**315.7**	**322.0**	**328.5**
	15	30	30	732.7	**577.6**	**567.6**	**539.8**	**538.5**	**536.7**	**537.8**	**541.6**	**545.2**	**550.7**
Optimal	7	10	10	138.7	129.7	131.5	135.7	141.7	148.0	155.5	162.0	169.6	177.4
	7	10	30	186.6	191.4	198.6	207.1	216.3	224.9	235.0	243.2	252.8	261.6
	15	10	10	219.6	202.0	201.4	205.2	211.4	218.2	226.7	233.8	242.2	251.0
	15	10	30	337.5	314.3	311.6	315.7	324.5	333.4	346.1	355.6	368.7	381.4
	7	30	10	269.6	222.9	211.1	209.0	211.4	215.8	221.3	227.4	233.9	240.2
	7	30	30	398.3	361.1	342.9	336.5	333.8	334.2	337.0	340.8	345.0	349.7
	15	30	10	377.3	308.0	290.9	287.7	289.3	293.5	299.2	305.4	312.0	318.5
	15	30	30	665.5	540.6	495.9	479.8	473.3	472.8	475.9	480.7	486.0	491.7

APPENDIX D

APPLICATION OF THE 2-OPP ALGORITHM TO FIND THE TURN MINIMIZING TOUR

In this part, we use a "reduced" form of the distance minimization algorithm proposed by Ratliff and Rosenthal [71] in order to find a turn minimizing tour in a 2-OPP graph. In coming up with the following observation, we are assuming that only the aisles between the left-most and right-most non-empty aisles are considered.

Observation D.1 *The only equivalence classes that are applicable are (U,U,1C), (0,E,1C), (E,0,1C), (E,E,1C) and (0,0,1C).*

(E,E,2C) is not applicable because partial traversal of an aisle from both sides is suboptimal, whereas (0,0,0C) is not applicable because we are assuming that only the aisles between the left-most and right-most non-empty aisles are considered.

Observation D.2 *Possible arc configurations within the aisles in an optimal tour subgraph are the ones labeled as (1), (2), (3), (5) and (6) in Figure 3.5.*

However, the arc configurations in Figure 3.5 will be used by guiding some insight. We know that (2) and (3) are applicable only if the depot is on that aisle, the number of nonempty aisles is odd and there is no item on one side of the depot on the aisle but items on the other side. If there are no items on the aisle, then any of (2) and (3) can be arbitrarily used. Furthermore, (5) is used when the number of nonempty aisles is odd and when (2) and (3) are not needed. It is also known that (5) is used exactly once when needed, so there is no need to consider it more than once. Lastly, (6) is used for empty aisles.

We have the following observation for the arc configurations between the aisles.

Table D.1: Resulting equivalence classes after adding the possible connection types in Figure 3.6 to each L_j^+ equivalence class, dashed lines indicating suboptimality or infeasibility

L_j^+ equivalence classes	Connection types in Figure 3.6		
	(1)	(2)	(3)
(U,U,1C)	(U,U,1C)	-	-
(E,0,1C)	-	(E,0,1C)	-
(0,E,1C)	-	-	(0,E,1C)
(E,E,1C)	-	(E,0,1C)	(0,E,1C)

Table D.2: Resulting equivalence classes after adding the possible connection types in Figure 3.5 to each L_j^- equivalence class, dashed lines indicating suboptimality

L_j^- equivalence classes	Connection types in Figure 3.5				
	(1)	(2)	(3)	(5)	(6)
(U,U,1C)	(E,E,1C)	(U,U,1C)	(U,U,1C)	(U,U,1C)	(U,U,1C)
(E,0,1C)	(U,U,1C)	(E,0,1C)	-	(E,E,1C)	(E,0,1C)
(0,E,1C)	(U,U,1C)	-	(0,E,1C)	(E,E,1C)	(0,E,1C)
(E,E,1C)	(U,U,1C)	(E,E,1C)	(E,E,1C)	(E,E,1C)	(E,E,1C)
(0,0,1C)	(U,U,1C)	(E,0,1C)	(0,E,1C)	(E,E,1C)	(0,0,1C)

Observation D.3 *Possible arc configurations between the aisles in an optimal tour subgraph are the ones labeled as (1), (2) and (3) in Figure 3.6.*

Table D.1 summarizes the resulting equivalence classes when the between-aisle connections in Figure 3.6 are added to each equivalence class, whereas Table D.2 gives the resulting equivalence classes when the within-aisle connections in Figure 3.5 are added to each equivalence class.