Methodologies and Applications for Scheduling, Routing & Related Problems

by

HAKAN YILDIZ

Submitted to the Tepper School of Business

in Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

at the

CARNEGIE MELLON UNIVERSITY

August 2008

Advisor: Michael A. Trick

Abstract

Scheduling and routing problems are two similar problem classes and distinction between them is not exactly clear as variants of them can be shown to be identical or similar. In this thesis, we develop methodologies, including large neighborhood search, simulated annealing, genetic algorithms, Bender's cuts, mixed-integer programming and constraint programming, to solve scheduling, routing and related problems. The applications we considered include scheduling of Major League Baseball (MLB) umpires, home-delivered meals provision and logistics planning at a manufacturer.

The scheduling needs of umpires differ from the needs of sports teams. For some sports, since umpires travel throughout the season, it is important to balance distance traveled with a wish to not handle the games of any particular teams too many times. In the first part of this thesis, we attempt to model these conflicts by introducing the Traveling Umpire Problem (TUP) as a multi-objective sports scheduling problem and we present results using several methodologies to solve it.

In the second part, we develop a large neighborhood search heuristic that uses network heuristics to solve the seemingly unrelated Graph Coloring problem. We provide results on benchmark instances.

In the third part, we present a Genetic Algorithm to solve the Home Delivered Meals Location-Routing Problem (HDM-LRP), which addresses facility location and design of delivery routes, while balancing efficiency and effectiveness considerations. We present results on benchmark LRP instances and also a case study on HDM data for Allegheny

County of Pennsylvania.

In the last part, we undertake an integrated study of inbound and outbound logistics at a leading automotive parts manufacturer. We identify the opportunity for significant cost savings over the current system by matching opposite flows of material from and to the customers and suppliers.

to my wife Nur

and my parents Ilknur & Cemil Yildiz

Acknowledgements

It has been an enjoyable journey, but it was certainly not easy. I thank everyone who has supported me. If I forgot to mention anyone's name here, I ask for their forgiveness, it was certainly not intentional.

I first would like to express my gratitude to my advisor, Michael A. Trick, for his guidance and support. It has been an honor being advised by and working with such a prominent figure in the Operations Research community. I especially thank him for giving me the freedom to work on a variety of problems and projects.

I would like to thank Michael P. Johnson for being a member of my dissertation committee and guiding me in my research. I also thank him for providing financial support during my second summer at Tepper and also providing travel support for conferences.

I also thank R. Ravi for being a member of my dissertation committee and getting me involved in a very interesting project during my last year at Tepper, which resulted in the fifth chapter of this thesis. He has been an excellent mentor during this project.

I am also indebted to the rest of the members of my dissertation committee, Steve Roehrig and Tallys Yunes for their valuable suggestions, insights and guidance. I also would like to thank Anuj Mehrotra for being a reader for my first summer paper, Willem-Jan Van Hoeve for his suggestions, and Anne Jackson for editing my papers and for her friendship.

I taught three courses as an instructor at Tepper. I am very thankful to Deputy Dean Ilker Baybars for his faith in me and allowing me to teach and also for his morale support

throughout the years.

I appreciate the assistance and collaboration given me by all my fellow graduate students and very much appreciate their friendship. Thank you Borga, Nihat, Latife, Erkut, Ozgun, Ismail, Anureet, Vineet, John, Vijay, Teresa, Prasad and Miroslav. I also would like to thank Lawrence Rapp for his promptness and professionalism in helping the PhD students.

I want to thank all my friends who made Pittsburgh a wonderful place to live. You are so many that I can't list you all here, but I want you to know that I love you! I would like to especially thank Oguzhan Alagoz, Mahmut Demir, Ensar Hatipoglu, Ahmet Kaya, Caner Kazanci, Galip Kiyakli, Umit Ogras, Osman Ozaltin, Kursat Ozenc, and Burhaneddin Sandikci.

I also thank my long time friend Altan Kalayci who has been there for me whenever I needed. His friendship is invaluable.

And, my wonderful family...I am forever indebted to my parents, Ilknur and Cemil for their love and support. Although they would much rather me being in Turkey and close to them, they supported me from the beginning and endured the uneasiness of being far away. Thanks also to my sisters Handan and Setenay and my aunt Makbule for their constant love.

I have added another wonderful family to my life by marrying my wife: my parents-in-law Taciser and Yasar, and my brother-in-law Fatih. Thank you for bearing with being thousands of miles away from your beloved Nur, my wonderful wife, my best friend and to whom I owe so much that I can not list here. Words can never express my thanks for the support she has given me over the years. None of this would have been possible without her endless love, inspiration, encouragement and unconditional support.

Table of Contents

1 Introduction **1**

2 Scheduling Baseball Umpires & the Traveling Umpire Problem **5**

2.1 Problem Description 6

2.1.1 TUP Instances and Implementation of Models & Algorithms 9

2.2 Exact Solution Approaches: Initial Models and Results 10

2.3 Bender's Cuts Guided Large Neighborhood Search for TUP 11

2.3.1 Greedy Matching Heuristic 11

2.3.2 A Bender's Based Modification 12

2.3.3 Bender's Cuts and Large Neighborhood Search 13

2.3.4 Computational Results 18

2.4 Exact Solution Approaches Revisited 23

2.4.1 Integer Programming 23

2.4.2 Constraint Programming 24

2.4.3 Computational Results Using Best Strategies 28

2.5 Simulated Annealing for TUP 28

2.5.1 Greedy Matching Heuristic with Backtracking 28

2.5.2 Simulated Annealing 29

2.5.3 Computational Results 31

2.6 Locally Optimized Crossover for TUP 31

2.6.1 Solution Strategy 31

2.6.2 Computational Results 37

2.7 Summary and Future Research 40

2.7.1 Strengthening the Bender's Cuts 41

2.7.2 Solving the multi-objective TUP 41

2.7.3 Bridging Local Search and Genetic Algorithms 42

2.7.4 Optimized Crossovers 42

2.7.5 Graph Coloring and TUP 42

3 A Large Neighborhood Search Heuristic for Graph Coloring 45
3.1 Local Search for Graph Coloring 46
3.1.1 Moves and Neighborhoods 47
3.1.2 Size of the Neighborhoods 48
3.1.3 Graph Cuts 48
3.1.4 Finding the Optimal Swap Move 48
3.1.5 Finding the Optimal Expansion Move 50
3.2 Algorithms 55
3.2.1 Swap-Move Algorithm 55
3.2.2 Expansion-Move Algorithm 55
3.2.3 Check-Bipartite Algorithm 56
3.2.4 Expansion-Swap Algorithm 56
3.2.5 Finding a Maximum Cut 58
3.3 Experimental Results 59
3.3.1 Implementation Details 59
3.3.2 Instance Description 60
3.3.3 Summary of Results 60
3.4 Summary and Future Research 64

4 Home-Delivered Meals Location-Routing Problem 67
4.1 Problem Definition and Related Literature 68
4.2 Solution Strategy 71
4.2.1 Representation and Genetic Operators 73
4.2.2 Initial Population of Solutions 81
4.2.3 Evaluation Function 82
4.2.4 Additional Features 82
4.2.5 The Algorithm 84
4.3 Experimental Results 85
4.3.1 Implementation Details 85
4.3.2 Instance Description 86
4.3.3 Summary of Results 86
4.4 Case Study: HDM-LRP in Allegheny County, PA 91
4.4.1 Data 92
4.4.2 Computational Results 95
4.5 Summary and Future Research 97

5 Optimization of Customer and Supplier Logistics at a Large Automotive Parts Manufacturer 99
5.1 Matching Opposite Flows: A Unique Opportunity . . . 101
5.1.1 Crossdocking . . . 102
5.2 Motivation . . . 103
5.3 Problem Description . . . 103
5.4 Literature Review . . . 104
5.5 Modeling Assumptions . . . 106
5.6 Problem Formulation . . . 106
5.6.1 Customer Sensitive Constraints . . . 107
5.7 Data & Preprocessing . . . 108
5.7.1 Cost Structure . . . 109
5.7.2 Initial Analysis . . . 109
5.8 Solution & Results . . . 109
5.8.1 Flexibility Analysis . . . 111
5.8.2 Utilizing the Crossdock . . . 113
5.8.3 Inventory Considerations . . . 114
5.8.4 Other Considerations . . . 115
5.9 Summary and Future Research . . . 116
5.9.1 Methodological Extensions . . . 116

6 Appendix 119
6.1 Appendix for Chapter 2 . . . 120
6.1.1 Structural Properties of TUP . . . 120
6.1.2 Initial IP Formulation for the TUP . . . 126
6.1.3 Improved IP Formulation for the TUP . . . 128
6.1.4 Initial CP Model for the TUP in OPL . . . 130
6.1.5 Improved CP Model for TUP in OPL . . . 132
6.2 Appendix for Chapter 4 . . . 134
6.3 Appendix for Chapter 5 . . . 137

List of Figures

2.1 Illustration of forming a Bender's Cut: Part 1 15
2.2 Illustration of forming a Bender's Cut: Part 2 16
2.3 Illustration of the optimization in the crossover operator: Part 1 34
2.4 Illustration of the optimization in the crossover operator: Part 2 35

3.1 Illustration of a Swap Graph. 49
3.2 Illustration of an Expansion Graph. 51
3.3 Illustration of Property 1 on an Expansion Graph. 53
3.4 Illustration of Property 2 on an Expansion Graph. 54
3.5 Relationship of the solution time with the density of the graphs. 64

4.1 Network representation of Solution 1 for LRP 74
4.2 Network representation of Solution 2 for LRP 74
4.3 Illustration of a Binary Representation. 79
4.4 Sample trade-off curve for a bi-objective LRP instance. 90
4.5 Client and kitchen locations for Allegheny County, PA 93
4.6 Client and kitchen locations for the portion of Allegheny County, PA studied in the case study. 94
4.7 First tradeoff curve for HDM-LRP on Allegheny County Data 96
4.8 Second tradeoff curve for HDM-LRP on Allegheny County Data 96

5.1 Logistics in the auto industry. 100
5.2 Matching uneven flows from and to the suppliers and customers. 101
5.3 Use of a crossdock to better match the uneven flows. 102
5.4 Illustration of current and future states. 110
5.5 Flexibility analysis over the weekdays 111
5.6 Flexibility analysis over the weeks of the quarter. 112
5.7 The effect pushing all flow through the crossdock. 114

5.8 Effect of shipment frequency on the inventory. 115

List of Tables

2.1 An example round robin tournament and a feasible umpire schedule. 8
2.2 Results of the initial IP and CP models. 11
2.3 Partial schedule for 8 teams and 4 umpires. 15
2.4 Illustration of forming a Bender's Cut. 17
2.5 Comparison of the IP and the Greedy Matching Heuristic with Bender's Cuts Guided Neighborhood Search (GBNS) in finding feasible solutions for instances with 12 teams and 14 teams. 20
2.6 Comparison of the 3-Umpire Neighborhood search when started from the initial solutions found by IP and GBNS for instances with 12 teams and 14 teams. 22
2.7 Effect of valid cuts on the IP Model. 24
2.8 Effect of symmetry breaking for the IP Model. 24
2.9 Comparison of atmost and alldifferent formulations of Constraint 4 in the CP Model. 25
2.10 Comparison of Implications and alldifferent formulations of Constraint 5 in the CP Model. 26
2.11 Effect of the Search Command in the CP Model. 27
2.12 Effect of symmetry breaking for the CP Model: 8 team instance 27
2.13 Effect of symmetry breaking for the CP Model: 10 team instance 27
2.14 Results for the improved IP and CP models. 28
2.15 Parameters used in the Simulated Annealing Algorithm. 30
2.16 Illustration of two feasible parent solutions. 32
2.17 Illustration of the crossover operator. 33
2.18 Results for the Simulated Annealing and Genetic Algorithms: Part 1 38
2.19 Results for the Simulated Annealing and Genetic Algorithms: Part 2 38
2.20 Results for the TUP instance defined on the MLB's 2006 game schedule with 30 teams. 39

3.1 List of the edge weights on the example Expansion Graph. 52
3.2 Comparison of the results of the expansion-swap(ES) algorithm to the results of other heuristics: Part 1 62
3.3 Comparison of the results of the expansion-swap(ES) algorithm to the results of other heuristics: Part 2 63

4.1 Results of the MIP Model on small LRP instances. 87
4.2 Results of the GA on the first set of LRP benchmark instances. 89
4.3 Results of the GA on the second set of LRP benchmark instances. 90

5.1 Results for different scenarios. 109
5.2 Max truck volume utilization on average and 90th percentile data. 113

6.1 A supporting example for Conjecture 1. 125

Chapter 1

Introduction

Scheduling, in very general terms, is the process of optimally allocating scarce resources to activities over time. Routing, on the other hand, is the optimal selection of paths in a network along which to send physical traffic. The distinction between scheduling and routing problems is not exactly clear as variants of these problems can be shown to be identical or similar: a scheduling problem may be represented as a routing problem and vice versa [8]. In this thesis we present solution methods and applications for different variants of these two related problem classes. Namely, we solve umpire scheduling, graph coloring and location-routing problems. This thesis is composed of four chapters and each of these chapters is self contained.

Scheduling the umpires of Major League Baseball(MLB) is a complex and difficult problem as it consists of dozens of pages of constraints. Moreover, the schedule aims to optimize a number of conflicting goals. In Chapter 2 of this thesis, we introduce the Traveling Umpire Problem (TUP) as a multi-objective sports scheduling problem. Like the Traveling Tournament Problem for league scheduling, which was introduced by Easton et al. [27], the TUP is based on the most important features of a real sports scheduling problem, the MLB Umpire Scheduling Problem. We model this problem both as a mathematical program and as a constraint program. We then improve these formulations using several techniques. We show

that exact solution methods are ineffective in solving large instances of TUP. Even finding a feasible solution is very challenging for many instances. Given the difficulties we face when trying to find optimal solutions, we explore heuristic approaches to find good solutions in a reasonable amount of time. We begin with developing a simple Greedy Matching Based Heuristic (GMH). We, then, introduce the hybrid use of GMH with Bender's Cuts and Large Neighborhood Search. Then, we modify the initial GMH to allow backtracking and improve the solution with a simple local search in a Simulated Annealing framework. And finally, we develop a Genetic Algorithm that uses a locally optimized crossover operator to generate offspring from parent solutions.

TUP is much related to the Graph Coloring Problem (GCP) and the Vehicle Routing Problem (VRP). In fact, we show that the feasibility version of the TUP, after relaxing one constraint, can be represented as a GCP on a special graph. On that same graph, TUP is equivalent to solving the Weighted Graph Coloring Problem, where weights are assigned to edges of the graph and the objective is to minimize to total sum of edges within each color class. Similarly, it is not hard to see TUP as a VRP with side constraints. With these observations, we move to the next two chapters of this thesis that investigate solution methods for the GCP and the variants of the VRP.

GCP is one of the central problems in graph theory. Many scheduling problems can, either exactly or partially, be represented as a coloring problem on a special graph, where nodes usually represent the activities and edges between nodes represent problem specific conflicts. Exam and class scheduling[65], crew scheduling [33], job-shop scheduling[1] are among the scheduling problems that have been represented and solved as a type of GCP. In Chapter 3, we develop a large neighborhood search heuristic that uses network heuristics to solve the seemingly unrelated GCP. This study is one of the first attempts to solve the GCP using local search in very large neighborhoods.

In Chapters 4 and 5 we look into two problems that both have routing and location aspects. The Location-Routing Problem(LRP) is a generalization of two well-known difficult

problems: the Fixed Charge Facility Location Problem and the Multi Depot Vehicle Routing Problem. The general planning problem that we address in Chapter 4 is the location-routing problem for home-delivered meals, referred to as HDM-LRP. In this problem, the goal is to simultaneously choose "facility" (kitchen) locations that provide "products" (meals) to spatially dispersed customers via routes driven by multiple vehicles. This planning model addresses facility location, allocation of demand to facilities, and design of delivery routes, while balancing efficiency and effectiveness considerations. We develop a Genetic Algorithm to solve this problem.

In Chapter 5, we study the integrated logistics planning at a leading automotive parts manufacturer. The problem involves selecting a subset of customers and suppliers to consolidate their shipments on the same routes. We consider the use of a crossdock located close to the customer and supplier base to use as a consolidation center. We identify the opportunity for significant cost savings over the current system by matching opposite flows of material from & to the customers and suppliers. We consider the problem from a supply chain coordination perspective, where the automotive parts manufacturer makes all the transportation arrangements for its customers & suppliers based on the results of the centralized optimum solution.

Chapter 2

Scheduling Baseball Umpires & the Traveling Umpire Problem

The Major League Baseball (MLB) is composed of 30 teams who play 780 series of 2-, 3-, or 4-game series with each other. This adds up to 2430 games in a season. At any given time, at most 15 series are scheduled and need umpire crews (other sports refer to these as referees), who go as a group of four, officiate an entire series. The job of an umpire is not an easy task. Being an umpire is a full time job for six months and a typical umpire handles about 130 games over a 182 day season (players play 162 games in same time). Interestingly, an umpire travels from one city to another through out the season instead of staying in one city. There are two reasons for this: First, we would need to almost double the number of crews if umpires stayed in one city. Second, it is undesirable to have umpires handle too many games with one team.

The MLB Umpire Scheduling Problem (MLB-USP) is complex and difficult to solve as it consists of dozens of pages of constraints, including such idiosyncratic constraints as an umpire's preferred vacation dates. Moreover, the schedule must satisfy league-imposed travel rules and restrictions and it aims to optimize a number of conflicting goals. The MLB-USP is formally defined in [93]. In this chapter we introduce the Traveling Umpire

Problem (TUP), which extracts the most critical aspects of MLB-USP but not the extraneous features, as a multi-objective sports scheduling problem and we develop several methodologies to solve it.

The literature contains many papers concerned with the scheduling of sports leagues (see the following surveys and books for more references [28, 76, 13]). However, papers concerned with the scheduling of sports officials are very few. A general referee assignment problem is considered in [25]. There are a few studies related to scheduling umpires for Major League Baseball. These papers are due to Evans et al.[44, 45] and Ordonez and Knowles [70]. There is another study for scheduling umpires in the US Open, which is due to Farmer and Smith [29], and there are two papers written by Wright on scheduling umpires for cricket leagues [89, 90].

The rest of this chapter is organized as follows. In Section 2.1 we describe the TUP. In Section 2.2 we discuss the initial math programming and constraint programming models. We present a greedy heuristic and Bender's Cuts Guided Large Neighborhood Search in Section 2.3. Section 2.4 revisits the exact solution approached to improve the initial models. In Section 2.5 we present the Simulated Annealing Algorithm we developed and in Section 2.6 we present a Genetic Algorithm that uses locally optimized crossover operators. The summary is given in Section 2.7.

2.1. Problem Description

In MLB each umpire crew consists of four officially trained and licensed major league umpires. These crews are put together at the beginning of the season and work together all season long. Thus, we do not worry about forming these crews, but rather assume that they are already predetermined. Because of that we will consider and refer to "umpire crews" as a single "umpire" in this chapter.

In contrast to MLB-USP, TUP limits the constraints to the key issues: no umpire should

be assigned to a team too often in short succession, and every umpire should be assigned to every team some time in a season. Given these constraints, the primary objective is to minimize the travel of the umpires.

Given a double round robin tournament, where every team plays against all other teams twice, on $2n$ teams ($4n - 2$ slots), we want to assign one of n umpires to each game.

There are several objectives that need to be minimized:

1) Total umpire travel.
2) Number of teams seen once or twice (home or away).
3) Number of teams seen 5 times or more.
4) Number of teams not seen away.
5) The difference between max travel and min travel.

The main objective is the first one and the rest can be considered as side objectives. The objectives (2-4) can be aggregated into "Number of defects". Although TUP is defined as a multi-objective problem, in the rest of this chapter, we focus on solving the single objective version of TUP, where the objective is minimizing the total umpire travel. We will use *TUP* to represent the single objective version of the Traveling Umpire Problem with only the first objective and we will use *TUP-M* for the multi-objective version of the problem with all five objectives.

The constraints that need to be satisfied are:

1) Every game gets an umpire.
2) Every umpire works exactly one game per slot.
3) Every umpire sees every team at least once at the team's home.
4) No umpire is in a home site more than once in any $n - d_1$ consecutive slots.

Table 2.1: Round robin tournament for 4 teams and a feasible schedule for 2 umpires.

Slots	**1**	**2**	**3**	**4**	**5**	**6**
Umpire1	(1,3)	(3,4)	(1,4)	(3,1)	(4,3)	(2,3)
Umpire2	(2,4)	(1,2)	(3,2)	(4,2)	(2,1)	(4,1)

5) No umpire sees a team more than once in any $\lfloor \frac{n}{2} \rfloor - d_2$ consecutive slots.

We present an example of a round robin tournament for 4 teams and a feasible umpire schedule for $d_1 = d_2 = 0$ in Table 2.1. A game is represented as a pair (i, j) where i is the home team and j is the away team. Rows correspond to umpire schedules and columns correspond to games that are played in the corresponding time slots.

The parameters for the objective function and the constraints are not chosen arbitrarily. These selections are explained next.

Properties of the Parameters in the Objectives

Since there are $4n - 2$ time slots and n umpires, an umpire sees a team $4 - \frac{2}{n}$ times on average. So in a balanced umpire schedule, an umpire sees a team 3 or 4 times. Thus we want minimize the number of teams seen too few (1 or 2) or too many (5 or more) times. This is the rationale behind objectives 2 and 3.

Properties of the Parameters in the Constraints

Let P represent the TUP and let $P_{(R)}$ be a relaxation of P with constraint set R, where $R \subset \{1, 2, 3, 4, 5\}$.

Theorem 1 *For $P_{(1,2,4)}$, for any tournament and any game there exists a feasible schedule for a single umpire that covers that game. Moreover, in constraint 4 we have the following two properties: when n is even, n is an upper bound on $n - d_1$ for the existence of*

this feasibility. When n is odd, $n+1$ is an upper bound on $n - d_1$ for the existence of this feasibility.

Theorem 2 *For $P_{(1,2,5)}$ and $d_2 = 0$, for any tournament and any game there exists a feasible schedule for a single umpire that covers that game.*

Conjecture 1 *For $P_{(1,2,5)}$ and when $\lfloor \frac{n}{2} \rfloor$ is replaced with $\lceil \frac{n}{2} \rceil + 1$ in constraint 5 there exists a tournament and a game such that no umpire schedule can cover that game.*

We present the proofs of both theorems and a discussion of Conjecture 1 in Appendix 6.1.1.

Theorem 1 implies that, even though we do not want an umpire to see the same team at home frequently, there is a limit on the enforcement of this constraint even for the relaxation $P_{(1,2,4)}$ of TUP. This limit is at most n consecutive games, in which case $d_1 = 0$. Theorem 2, on the other hand, does not imply any restrictions on the number of consecutive slots during which an umpire can see the same team more than once. It just shows that the relaxation $P_{(1,2,5)}$ of TUP is always feasible when $d_2 = 0$.

The reason d_1 and d_2 have been included in the definition is the computational observation that, given a game schedule, the problem may become infeasible when the constraints are enforced with $d_1 = 0$ and $d_2 = 0$.

2.1.1 TUP Instances and Implementation of Models & Algorithms

We have tested the solution approaches presented in this chapter on a set of TUP instances. An instance of the TUP has two matrices: The Distance Matrix, which stores the pairwise distances between cities and the Opponents Matrix, which stores the tournament information. We have used instances with 4 teams to 32 teams. The instances with 14 teams or

less use the TTP Tournaments as given at [82]. The instances with more than 14 teams use the distance matrix for the National Football League given at [82], and the game schedule is generated using a Constraint Program [84] that creates a round robin tournament. All of the instances used in this study and additional ones are available at [92].

Depending on the choice of d_1 and d_2, the difficulty of the problem changes. Decreasing these parameters makes the problem harder to solve. Choosing a $d_1 = n - 1$, which makes $n - d_1 = 1$, or a $d_2 = \lfloor \frac{n}{2} \rfloor - 1$, which makes $\lfloor \frac{n}{2} \rfloor$ - $d_2 = 1$, simply means that Constraint 4 or Constraint 5 is not in effect.

All the models and algorithms developed to solve the TUP instances are implemented using the modelling and script language in ILOG OPL Studio 3.7 [45]. This version of the software uses ILOG CPLEX 9.0 to solve the math programming models and ILOG Solver 6.0 to solve constraint programming models.

2.2. Exact Solution Approaches: Initial Models and Results

TUP has many characteristics in common with the Vehicle Routing Problem with Time Windows (VRPTW) due to the emphasis on minimizing total travel cost of multiple routes. In fact, if we ignore Constraints 3,4 and 5, the problem becomes a special case of VRPTW. It is well known that VRP and almost all of its variants, including VRPTW, are NP-Hard [58] and exact solution approaches are ineffective in solving large instances. Since TUP has side constraints in addition to the routing constraints, it is even a more challenging problem to solve than the VRP and its variants.

We formulated TUP as an Integer Program (IP) and also as a Constraint Program (CP). The formulations are given in Appendix 6.1.2 and Appendix 6.1.4, respectively. The computational results for the IP and CP models are given in Table 2.2 for the smallest four instances with $d_1 = d_2 = 0$, which means Constraints 4 and 5 are most restricting. We allowed a maximum of 24 CPU hours for both approaches. Both of the approaches were

Table 2.2: Results of the initial IP and CP models for $d_1 = d_2 = 0$.

	IP		**CP**	
No of Teams	**Distance**	**Time**	**Distance**	**Time**
4	5176	0	5176	0
6	14077	1.5 secs	14077	1.3 secs
8	34311	145 secs	34311	525 secs
10	51128	24 hrs	53599	24 hrs

able to solve the instances with 4,6 and 8 teams to optimality very fast. Both of the models were able to find a feasible solution for the 10-team instance in the allowed 24 hours time limit but they could not find the optimum solution. Overall, the IP model performed better than the CP model for these instances. We present these results to demonstrate that the problem becomes very difficult to solve as we increase the number of teams. Given the difficulty of the problem, even finding a feasible solution to the TUP is challenging, making it difficult to find good solutions in a reasonable amount of time. Because of this we start exploring heuristic approaches to find good solutions.

2.3. Bender's Cuts Guided Large Neighborhood Search for TUP

In this section, we consider the hybrid use of Bender's Cuts, Large Neighborhood Search and a Greedy Matching Based Heuristic to find good, feasible solutions to the TUP.

2.3.1 Greedy Matching Heuristic

Greedy Matching Heuristic (GMH) is a constructive heuristic, which builds the umpire schedules starting from Slot 1 and ending at Slot $4n - 2$. For every slot t, the heuristic assigns umpires to games such that all constraints, except Constraint 3, are satisfied and the best possible assignment is made to minimize the total umpire travel at Slot t. To do that,

GMH solves a Perfect Matching Problem on a Bipartite Graph in every slot t by solving an integer program. The partitions in this Bipartite Matching Problem are the Umpires and the Games in slot t. An edge is placed between an umpire u and a game (i, j) if Constraints 4&5 are not violated by assigning u to game (i, j) in slot t, given the assignments from slot 1 to $t-1$. Cost of an edge $(u, (i, j)) = Distance(k, i) - Incentive(u, i)$. In this cost function, k is the venue that u is assigned in slot $t-1$ and $Distance(k, i)$ is the distance between cities k and i. $Incentive(u, i)$ takes a positive value if umpire u has never visited venue i in the previous slots. This reduction in distance guides the GMH towards assigning umpires to cities that they have not visited yet, thus reducing the possibility of having a solution that violates Constraint 3 at the end of the execution of GMH.

2.3.2 A Bender's Based Modification

In practice, the GMH often gets stuck at some time slot because there may be no feasible perfect matching. In this case, we can identify a set of previous assignments that are causing this lack of perfect matching. At least one of those previous assignments must be changed in order to create a perfect matching. This set of assignments leads to a *Logic-Based Bender's Cut* in the terminology of [44].

While we could add the Bender's cuts to an integer programming formulation of this problem, in a standard master/subproblem approach to this problem, instead we use these cuts to guide a large neighborhood search heuristic. Violation of Bender's Cuts is penalized in the objective function with a large cost. Thus, any changes in the schedule that reduces the the number of violated Bender's Cuts or reduces total umpire travel is accepted. When a solution that satisfies all the Bender's Cuts is found, we stop the neighborhood search and solve the Perfect Matching Problem for slot t again. If there is no feasible matching, we repeat the process. A high level pseudo code is presented in Algorithm 1. We explain the generation of Bender's Cuts and the Large Neighborhood Search Algorithm in the next section.

Algorithm 1 Greedy Matching Heuristic with Bender's Cuts Guided Neighborhood Search (GBNS)

```
1: Arbitrarily assign umpires to games in slot 1
2: for all 1 < t ≤ 4n-2 do
3:    Construct a Bipartite Graph G = (V, E) for the Perfect Matching Problem
4:    Find a Minimum Cost Perfect Matching on G
5:    if there is no feasible perfect matching then
6:       Find all Bender's Cuts
7:       Set Objective = (no of violated cuts) * (violation cost) + (total umpire travel)
8:       Set improvement = 1
9:       while at least one of the cuts is violated & improvement > 0 do
10:         Do neighborhood search using 3-Ump and 3-Slot Neighborhoods
11:         if Objective is not improved then
12:            improvement = 0
13:         end if
14:      end while
15:   end if
16: end for
```

2.3.3 Bender's Cuts and Large Neighborhood Search

Generating Bender's Cuts

Bender's Cuts are generated when there is no feasible matching at a slot t during the course of GMH. Such an infeasibility implies that the partial schedule built until slot t can not be completed to obtain a feasible solution. We, then, examine the reasons for this infeasibility and this examination leads to constraints that exclude a number of partial solutions. Clearly, the Bender's Cuts generated in this method are not classical Bender's Cuts, which are obtained from a linear subproblem as in traditional Bender's Decomposition. Instead these cuts are *Logic-Based Bender's Cuts* as defined by Hooker and Ottosson[44], where the Bender's Cuts are obtained from *inference duals*. The difference between the Logic-Based Bender's Cuts and classical Bender's Cuts is that no standard form exists for the Logic Based Bender's Cuts and such cuts are generated by logical inference on the solution of the subproblem. When the subproblem is a feasibility problem, the inference dual is a condition which, when satisfied, implies that the master problem is infeasible. This condition can be

used to obtain a Bender's Cut for cutting off infeasible solutions. Logic Based Bender's Cuts are a special case of "nogoods," a well-known known idea in constraint programming literature, but they exploit problem structure in a way that nogoods generally do not[23]. Logic-Based Bender's Cuts have been successfully used in several studies [43, 41, 39, 75]. In this subsection we describe the way we generate Logic Based Bender's Cuts and how we use these cuts.

The Bender's Cuts are generated for any set of Umpires (or Games) whose adjacency neighborhood has a cardinality less than the cardinality of the set itself. The *adjacency neighborhood* $N(A)$ of a set A is the set of nodes that are adjacent to a node in A. This condition is known as Hall's Theorem:

Hall's Theorem: *Let G = (V, E) be a bipartite graph with bipartitions X and Y. Then G has a perfect matching if and only if* $|N(A)| \geq |A|$ *for all* $A \subseteq X$

Because of Hall's theorem, if there is no perfect matching in a time slot, there is a subset of umpires A whose neighborhood $N(A)$ has cardinality smaller than $|A|$. We will generate a constraint that creates at least one edge between A and $Y \setminus N(A)$ as follows. For each pair x, y such that $x \in A$ and $y \in Y \setminus N(A)$ and $(x, y) \notin E$, there exists an already made game-umpire assignment in previous slots that prevents the edge to be in the matching problem. We find all such game-umpire assignments for all the missing edges between $x \in A$ and $y \in Y \setminus N(A)$. Then the corresponding Bender's Cut requires that at least one of these game-umpire assignments should be changed. To obtain all possible Bender's Cuts, we identify Hall sets by checking all subsets of Umpires and their neighborhoods. For small instances, the number of cuts identified when an infeasibility occurs is not too many, though for larger problems it would be necessary to generate only a limited number of cuts.

We present a partial schedule for an example instance for 8 teams and 4 umpires in Table 2.3. For this instance we assume that $d_1 = d_2 = 0$. Thus, the fourth constraint imposes

Table 2.3: Partial schedule for 8 teams and 4 umpires. The first three slots are scheduled and the games for the fourth slot are in consideration for assignment.

Slots	1	2	3	4
Umpire1	(7,5)	(2,4)	(5,7)	(2,1)
Umpire2	(1,8)	(3,6)	(4,1)	(4,5)
Umpire3	(2,6)	(1,7)	(6,8)	(6,3)
Umpire4	(4,3)	(5,8)	(3,2)	(8,7)

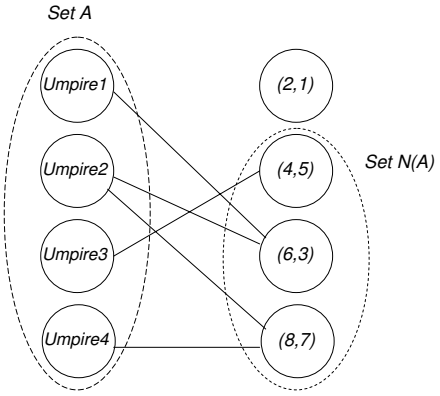

Figure 2.1: Bipartite matching problem for slot 4. The partitions are Umpires and the Games in slot 4.

that an umpire can not visit the same home venue more than once in any 4 consecutive games. The fifth constraint, on the other hand, imposes that an umpire can not see a team more than once in any two consecutive games. For this example the games for the first three slots are scheduled and we are considering the games in the fourth slot.

The matching problem that corresponds to slot 4 is given in Fig. 2.1. In this figure, set A and $N(A)$, the adjacency neighborhood of A, are circled with dashed lines. The cardinality of A is 4, whereas the cardinality of $N(A)$ is 3. Thus, there is no feasible perfect matching for this graph.

The way we obtain the Bender's Cut from set A is best demonstrated with the help of Fig. 2.2 and Table 2.4. The Bender's Cut states that at least one of the edges between the nodes in A, the umpires, and the complement of $N(A)$, Game $(2, 1)$, has to be in the graph.

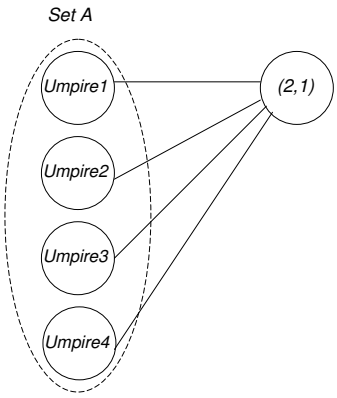

Figure 2.2: The missing edges between set A and the complement of N(A). Bender's Cut requires at least one of these to be present in the bipartite perfect matching problem.

To write this cut in terms of the game-umpire assignments, we identify the game-umpire assignments in slots $1, 2$ and 3, which prevents the edges being in the graph. For Umpire1, Game $(2, 4)$ in slot 2 is preventing the edge between Umpire1 and Game $(2, 1)$, because Umpire1 can not visit venue 2 twice in four consecutive games. Similarly, for Umpire2, Game $(4, 1)$ in slot 3 is preventing the edge between Umpire2 and Game $(2, 1)$; for Umpire 3, Game $(2, 6)$ in slot 1 is preventing the edge between Umpire3 and Game $(2, 1)$; for Umpire4, Game $(3, 2)$ in slot 3 is preventing the edge between Umpire4 and Game $(2, 1)$. Thus the cut generated is

$$assigned[1, 2, 2] + assigned[2, 4, 3] + assigned[3, 2, 1] + assigned[4, 3, 3] \leq 3$$

$$\text{where, } assigned[u, i, t] = \begin{cases} 1, & \text{if umpire u is at venue i in slot t} \\ 0, & \text{otherwise.} \end{cases}$$

Very Large Neighborhood Search

A *neighborhood* of a solution S is a set of solutions that are in some sense close to S, i.e., they can be easily computed from S or they share a significant amount of structure with S.

Table 2.4: Games that are in conflict with game (2,1) in slot 4 are highlighted.

Slots	1	2	3
Umpire1	(7,5)	**(2,4)**	(5,7)
Umpire2	(1,8)	(3,6)	**(4,1)**
Umpire3	**(2,6)**	(1,7)	(6,8)
Umpire4	(4,3)	(5,8)	**(3,2)**

An algorithm that starts at some initial solution and iteratively moves to solutions in the neighborhood of the current solution is called a *Neighborhood Search Algorithm* or a *Local Search Algorithm*

The Very Large Neighborhood Search(VLNS) for the TUP tries to improve the solution quality at each iteration. It is clear that the larger the neighborhood, the better is the quality of the solutions that can be reached in one single move. At the same time, the larger the neighborhood, the longer it takes to search the neighborhood at each iteration.

To make the notion of *very large neighborhood* clear, we'll first introduce the well known *2-exchange neighborhood*, which is a small polynomially sized neighborhood. Given an umpire schedule, a *2-exchange move* swaps the umpires assigned to two games played in the same slot t. The neighborhood for this move is the set of schedules that can be obtained by performing a single move. We introduce two very large neighborhoods in the following subsections. We use both neighborhoods in our search for a feasible solution with the use of Bender's Cuts, when the Greedy Matching Heuristic is unable to assign the games to umpires at a slot. Once an initial solution is found, we further use the K-Umpire Neighborhood to improve that solution. We stop this search when we reach a local optimum.

K-Umpire Neighborhood: We take the schedules of $K \leq n$ umpires and allow exchanges of game assignments within these schedules. The exchanges of game assignments are allowed only within the same slot, not across slots, as we need to make sure that Constraints 1&2 are always satisfied. In each slot, there are $K!$ different ways of assigning games to umpires. Since there are $4n - 2$ slots, the possible solutions that can be reached by one K-Umpire

move is $(K!)^{4n-2}$.

The problem of finding the best move is done using the Restricted IP for K-Umpires(RIP-U) for the TUP with only the games for the K umpires in consideration. Since this RIP-U is solved many times during the execution of the algorithm, the solution time should be very low to have the algorithm terminate in a reasonable amount of time. Since the RIP-U becomes a very hard problem for $K \geq 4$ for even small instances, we take $K = 3$ and we look at all possible 3 combinations of the n umpires.

K-Slot Neighborhood: We take the games at $K \leq 4n - 2$ slots and allow exchanges of umpire assignments within these slots. The exchanges of umpire assignments are allowed only within the same slot, not across slots, as we need to make sure that Constraints 1&2 are always satisfied. In each slot, there are $n!$ different ways of assigning games to umpires. Since only K slots are considered at a time, the possible solutions that can be reached by one K-Slot move is $(n!)^K$.

The problem of finding the best move is done using the Restricted IP for K-Slots(RIP-S) for the TUP with only the games for the K slots in consideration. Since this RIP-S is solved many times during the execution of the algorithm, the solution time should be very low to have the algorithm terminate in a reasonable amount of time. Since the RIP-S becomes a hard problem as K grows we take $K = 3$ and we look at all possible 3 combinations of the $4n - 2$ slots.

2.3.4 Computational Results

We report the computational results in this subsection.

Finding a Feasible Solution

As the problem size increases and as the constraints 4 and 5 become more restricting, even finding a feasible solution to the TUP becomes difficult. To find a feasible solution we used the IP and the GMH with the usage of Bender's Cut's Guided Large Neighborhood Search,

which we will refer as GBNS.

Table 2.5 summarizes the results for IP and GBNS for the instances with 12 and 14 teams. The GBNS approach finds much better solutions much faster in both instances and all the combinations of the parameters for Constraints 4&5. For 12 teams instance the GBNS obtains improvements ranging from 3.1% to 23.6% and for 14 teams instance ranging from 4.5% to 38.6% over the IP.

For 14 teams and for the combination of parameters $(n - d_1, \lfloor \frac{n}{2} \rfloor$ - $d_2) = (6, 3)$ and $(n - d_1, \lfloor \frac{n}{2} \rfloor$ - $d_2) = (7, 3)$ the IP could not find a feasible solution even after more than 30 hours of execution, whereas GBNS found a solution for each case. Notice that it takes GBNS only 0.7 seconds to solve this instance for $(n - d_1, \lfloor \frac{n}{2} \rfloor$ - $d_2) = (6, 3)$, whereas for $(n - d_1, \lfloor \frac{n}{2} \rfloor$ - $d_2) = (7, 3)$, it takes GBNS 7322.9 seconds. This drastic difference in time is due to the fact that the problem becomes extremely difficult to solve when $d_1 = d_2 = 0$.

For 12 teams and for the combination of parameters $(n - d_1, \lfloor \frac{n}{2} \rfloor$ - $d_2) = (5, 3)$ and $(n - d_1, \lfloor \frac{n}{2} \rfloor$ - $d_2) = (6, 3)$ neither the IP nor the GBNS could find a feasible solution even after several hours of running. Since the IP was terminated before concluding to the infeasibility of these cases, we do not currently know if these two instances are actually feasible or not.

Improving the solution with VLNS

We run 3-Umpire Neighborhood Search on the initial solutions obtained by the IP and the GBNS. While GBNS used this neighborhood in creating the solution, this was done with costs associated with the Bender's Cuts. Once a feasible solution is found, those costs are no longer required. Furthermore, the local search aspect of GBNS is only applied when there is an infeasibility. So it is entirely possible that further improvements are possible once GBNS terminates, and our computational tests bear that out.

Table 2.6 summarizes the results for IP and GBNS for the instances with 12 and 14 teams. The quality of the initial solution does not make any significant difference in terms

Table 2.5: Comparison of the IP and the Greedy Matching Heuristic with Bender's Cuts Guided Neighborhood Search (GBNS) in finding feasible solutions for instances with 12 teams and 14 teams. The columns consist of the parameter for Constraint 4 ($n - d_1$), the parameter for Constraint 5 ($\lfloor \frac{n}{2} \rfloor$ - d_2), IP solution cost (IP), time in seconds, GBNS solution cost (GBNS) and time in seconds, and % cost improvement of GBNS over IP (% Impr.).

12 Teams

$\mathbf{n - d_1}$	$\lfloor \frac{\mathbf{n}}{\mathbf{2}} \rfloor$ - d_2	IP	time(sec)	GBNS	time(sec)	% Gap
2	1	95024	7.1	72557	0.1	23.6
3	1	97276	10.4	76407	0.1	21.5
4	1	93762	7.4	76756	0.1	18.1
5	1	93030	19.1	76781	0.0	17.5
6	1	99632	67.3	77818	0.1	21.9
2	2	101055	20.8	88277	0.1	12.6
3	2	102399	46.7	88637	0.1	13.4
4	2	101978	36.8	90231	0.1	11.5
5	2	100641	93.4	91951	0.1	8.6
6	2	100372	134.1	91131	0.1	9.2
2	3	100089	7136.3	95072	359.3	5.0
3	3	100797	1025.3	95072	359.3	5.7
4	3	101063	2194.4	97945	28.6	3.1
5	3	–		–		
6	3	–		–		

14 Teams

$\mathbf{n - d_1}$	$\lfloor \frac{\mathbf{n}}{\mathbf{2}} \rfloor$ - d_2	IP	time(sec)	GBNS	time(sec)	% Gap
2	1	182520	56.5	112142	0.1	38.6
3	1	184069	70.0	117618	0.1	36.1
4	1	180170	42.8	118647	0.1	34.1
5	1	184496	46.6	120781	0.1	34.5
6	1	187357	182.9	121998	0.1	34.9
7	1	182435	196.7	126360	0.0	30.7
2	2	196977	88.4	152658	0.1	22.5
3	2	202383	133.9	153536	0.1	24.1
4	2	199001	194.2	155073	0.1	22.1
5	2	201533	198.7	155525	0.1	22.8
6	2	194975	370.6	157761	0.1	19.1
7	2	206335	621.7	161428	0.1	21.8
2	3	196003	401.4	175884	0.1	10.3
3	3	196394	10085.8	175884	0.1	10.4
4	3	190640	1989.1	182054	0.1	4.5
5	3	205230	2442.9	182746	0.1	11.0
6	3	–		183331	0.7	
7	3	–		186979	7322.9	

of the quality of the final solution obtained by the neighborhood search. On the other hand, the total times spent on finding the initial solution and then running the neighborhood search is significantly less for GBNS, which is due to the very fast execution of GBNS. In short, the real value of GBNS is the quick generation of a feasible solution.

Table 2.6: Comparison of the 3-Umpire Neighborhood search when started from the initial solutions found by IP and GBNS for instances with 12 teams and 14 teams. The columns consist of the parameter for Constraint 4 ($n-d_1$), the parameter for Constraint 5 ($\lfloor \frac{n}{2} \rfloor$ - d_2), solution cost when initial solution is found by IP (IP & 3-Ump), time in seconds, solution cost when initial solution is found by GBNS (GBNS & 3-Ump) and time in seconds, and % cost improvement obtained when started with GBNS over when started with IP (% Impr.).

12 Teams

$\mathbf{n-d_1}$	$\lfloor \frac{\mathbf{n}}{\mathbf{2}} \rfloor$ - d_2	IP & 3-Ump	time(sec)	GBNS & 3-Ump	time(sec)	% Gap
2	1	62515	242.2	62374	192.2	0.2
3	1	66159	408.2	66090	325.4	0.1
4	1	66608	352.5	66897	378.5	-0.4
5	1	68312	466.2	69103	351.5	-1.2
6	1	69445	722.2	69775	759.7	-0.5
2	2	82288	256.9	82188	349.0	0.1
3	2	83095	459.3	83504	245.9	-0.5
4	2	83529	407.5	83974	444.5	-0.5
5	2	85192	541.1	85804	495.4	-0.7
6	2	91865	464.2	91018	279.6	0.9
2	3	95924	7367.3	94080	666.7	1.9
3	3	92672	1578.3	94080	666.7	-1.5
4	3	101047	2573.2	97945	288.5	3.1
5	3	–		–		
6	3	–		–		

14 Teams

$\mathbf{n-d_1}$	$\lfloor \frac{\mathbf{n}}{\mathbf{2}} \rfloor$ - d_2	IP & 3-Ump	time(sec)	GBNS & 3-Ump	time(sec)	% Gap
2	1	95075	894.6	94748	1053.1	0.3
3	1	103854	1451.3	102021	1427.9	1.8
4	1	106243	1874.4	106751	1508.8	-0.5
5	1	109901	1645.8	110896	1307.5	-0.9
6	1	112385	2920.7	111184	1792.6	1.1
7	1	114077	2283.0	117332	1671.9	-2.9
2	2	140855	1789.8	140570	1221.0	0.2
3	2	142334	1157.1	141936	813.9	0.3
4	2	143746	1435.9	144301	910.9	-0.4
5	2	146129	850.9	146964	1400.2	-0.6
6	2	148860	2122.6	148389	2723.9	0.3
7	2	157402	1969.5	161413	950.8	-2.5
2	3	157444	5678.1	157111	3235.5	0.2
3	3	155776	5793.6	157111	3235.5	-0.9
4	3	166345	4754.3	164263	3625.9	1.3
5	3	177480	5762.4	168995	2469.4	4.8
6	3	–		181052	714.0	
7	3	–		186979	8005.6	

2.4. Exact Solution Approaches Revisited

The initial integer and constraint programming formulations discussed in Section 2.2 were ineffective in solving larger instances. In this section we try different methods of improving these formulations and present improved results.

2.4.1 Integer Programming

In addition to the five constraints that define TUP, we use some additional valid cuts to strengthen the initial formulation. We present these next.

Adding Valid Cuts

We strengthened the formulation with the following additional valid inequalities:

- If team i plays away in slot t, no umpire can be assigned to the game at venue i;
- If umpire c moves from venue i to venue j in slot t, it must be assigned to a game at venue i in t;
- If umpire c moves from venue i to venue j in slot t, it must be assigned to a game at venue i in $t+1$;
- Number of umpires moving to venue j at slot t should be equal to the number of umpires moving from venue j at $t+1$;
- Every umpire must move in every slot.

The computational results before and after adding these cuts to the IP model is given in Table 2.7, which indicates that adding these cuts improved the IP model significantly.

Symmetry Breaking

Since umpires are essentially identical, this creates symmetry in the formulations of the TUP. Breaking this symmetry is easy by arbitrarily assigning the games in one time slot. When

Table 2.7: Results for the IP with and without valid cuts.

		IP without cuts		IP with cuts	
No of Teams	**OPT Distance**	**Distance**	**Time**	**Distance**	**Time**
4	5176	5176	0	5176	0
6	14077	14077	1.5 secs	14077	0.3 secs
8	34311	34311	145 secs	34311	1.6 secs
10	48942	51128	$>$ 24 hrs	48942	$>$ 13 hrs

Table 2.8: Results for the IP (with cuts) and with Symmetry Breaking on the 10 Team Instance.

Break Symmetry	**Assign Games in Slot No**	**Time to prove OPT**
No	–	$>$ 13 hrs
Yes	1 (first)	538 secs
Yes	9 (middle)	72 secs
Yes	18 (last)	1804 secs

we tried this approach, the solution times for the IP model are reduced very significantly. Regardless of which slot we pick, this reduction is very significant. Interestingly, we see that the slot that we pick also makes a difference. We compare fixing the games in the first, middle and last slots and we see that we get a much more reduction in time for the middle time slot. The results for the 10 team instance is summarized in Table 2.8.

2.4.2 Constraint Programming

We tried several methods to improve the initial formulation and we discuss these next.

atmost vs. alldifferent for Constraint 4

We can formulate Constraint 4 of TUP by using the `atmost` constraint as shown below:

```
forall(u in Umps, t in Slots: t<= (4n-2)-n+1 )
   atmost(all(i in Teams) 1, all(j in Teams) j, all(r in [0..n-1]) assigned[u,t+r]);
```

Table 2.9: Comparison of atmost and alldifferent formulations of Constraint 4 on the Instance with 8 Teams.

Constraint 4	**Time to find OPT (sec)**	**Time to prove OPT (sec)**
atmost	21	525
alldifferent	15	415

We can also use the `alldifferent` constraint, which we demonstrate below:

```
forall(u in Umps, t in Slots: t<= (4n-2)-n+1 )
   alldifferent(all(r in [0..n-1]) assigned[u,t+r]);
```

The comparison of the results for these two formulations is shown in Table 2.9.

Implications vs. alldifferent for Constraint 5

In the initial CP formulation, we modelled Constraint 5 using several implications. We can replace these implications with an `alldifferent` constraint if we introduce new variables, `assigned_prime`, to the model, where `assigned_prime[u,t]`$= i$ when umpire u sees team i away in slot t. We directly link these new variables to the original `assigned` variables using the following relationship:

```
forall(t in Slots)
   forall(u in Umps)
        opponents[t,assigned[u,t]] =  assigned_prime[u,t];
```

Constraint 5 is now modelled using the `alldifferent` constraint as follows:

```
forall(u in Umps, t in Slots: t<= (4n-2)- floor(n/2))+1)
    alldifferent(all(r in [0..no_rep/2-1])assigned_prime[u,t+r]);
```

Table 2.10: Comparison of Implications and alldifferent formulations of Constraint 5 on 8 Team Instance.

Constraint 5	Time to find OPT (sec)	Time to prove OPT (sec)
Implications	21	525
alldifferent	49	1191

The results with this new formulation is given in Table 2.10. Interestingly, the `alldifferent` constraint performs worse than the long implications. One possible explanation for this is the fact that `alldifferent` constraints usually works better if the domain size is proportional to the variable set size. But in TUP, this is not the case. In fact, a variable can potentially take all possible assignments, and this may be preventing the `alldifferent` constraint to make more deductions than the implications.

Search Strategy

When we instruct the solver to search for all potential assignments where an umpire is assigned to one of the home venues, the solution times decrease very significantly, as seen in Table 2.11. The instruction we used is as follows:

```
search {
    forall(t in Slots)
      forall(u in Umps)
         tryall (i in Teams: opponents[t,i] > 0)
            assigned[u,t] = i; };
```

Symmetry Breaking

We used the same approach, as we did with the IP formulation, for breaking the symmetry of umpires: We pick a slot and arbitrarily assign the games in that slot to the umpires.

Table 2.11: Effect of Search Command on 8 Team Instance.

Search Command	**Time to find OPT (sec)**	**Time to prove OPT (sec)**
no	21	525
yes	0	2

Table 2.12: Results of Symmetry Breaking in CP for 8 Team Instance.

Break Symmetry	**Assign Games in Slot No**	**Time to find OPT (secs)**	**Time to prove OPT(secs)**
No	–	21	525
Yes	1 (first)	0	2
Yes	7 (middle)	1	2
Yes	14 (last)	8	10

Regardless of which slot we pick there is a significant reduction in the solution time. However, we do not see a clear difference in solution times between the different slots. The results for the 8 team and 10 team instances are summarized in Table 2.12 and Table 2.13, respectively. Based on the results, we pick the first slot to fix the games as it seems to be slightly superior than fixing the middle slot.

Table 2.13: Results of Symmetry Breaking in CP for 10 Team Instance.

Break Symmetry	**Assign Games in Slot No**	**Time to find OPT (secs)**	**Time to prove OPT(secs)**
No	–		
Yes	1 (first)	279 hrs	407 hrs
Yes	9 (middle)	300 hrs	388 hrs
Yes	18 (last)	410 hrs	550 hrs

Table 2.14: Improved IP and CP results for TUP with $d_1 = d_2 = 0$.

		BEST Distance		Time for OPT or BEST	
No of Teams	**OPT Distance**	**Imp.IP**	**Imp.CP**	**Imp.IP**	**Imp.CP**
4	5176	5176	5176	0	0
6	14077	14077	14077	0.3 secs	0.2 secs
8	34311	34311	34311	1.6 secs	2 secs
10	48942	48942	50380	72 secs	24 hrs
12	infeasible	–	infeasible	24 hrs	25 secs
14	unknown	187374	177684	24 hrs	4568 secs
16	unknown	–	–	24 hrs	24 hrs

2.4.3 Computational Results Using Best Strategies

In this subsection, we summarize the results using the improved IP and CP models, which were modified based on the findings presented in the previous two subsections.

The improved IP model is presented in Appendix 6.1.3. It includes the valid cuts derived and the symmetry breaking strategy by fixing the games in the middle slot is used.

The improved CP model is presented in Appendix 6.1.5. We use the `alldifferent` constraint instead of the `atmost` constraint to model Constraint 4. We also use the `search` command as described in Section 2.4.2 and the symmetry breaking strategy by fixing the games in the first slot.

The results are summarized in Table 2.14.

2.5. Simulated Annealing for TUP

In this section, we present a Simulated Annealing algorithm (SA) to solve the TUP.

2.5.1 Greedy Matching Heuristic with Backtracking

We use a slightly modified version of the Greedy Matching Heuristic (GMH), which we introduced in Section 2.3.1, to find an initial solution. GMH builds the umpire schedules

starting from the first slot and ending at the last slot. For every slot t, the best possible umpire-game assignment is made to minimize the total umpire travel in slot t and to minimize the constraint violations. If in a slot t, there is no feasible matching available, we backtrack to the previous slot $t-1$ and pick the second best matching and try again. Backtracking is made at most once at each slot. Thus, this heuristic may end up with an infeasible solution.

2.5.2 Simulated Annealing

A pure local search algorithm has the main disadvantage that it terminates in the first local optimum it reaches, which may be far from any global optimum because the algorithm only executes moves that generate a decrease in cost. SA, a powerful stochastic local search method, alternatively attempts to avoid becoming trapped in a local optimum by sometimes (with a non-zero probability that gradually decreases as the algorithm continues its execution) executing a move that generates an increase in cost. This way it is possible to "climb out of" the local minimums.

SA has its origins in the fields of materials science and physics [73]. Kirkpatrick et al. [49] established an analogy between minimizing the cost function of a combinatorial optimization problem and the slow cooling process of a solid by utilizing an optimization process. This algorithm has proven to be a good technique for many applications [86].

The pseudo code for the SA we used is presented in Algorithm 2. Our cost function is total distance traveled by the umpires. We use the simple *2-exchange moves* to do a local search. Given an umpire schedule S, a *2-exchange move* swaps the umpires assigned to two games played in the same slot.

We empirically tested the parameters for the SA and used the values presented in Table 2.15 in our tests.

Algorithm 2 Simulated Annealing Algorithm.

```
1: while time limit and iteration limit not exceeded do
2:   S = initial solution with prob. p or incumbent solution with prob. (1 − p)
3:   t = t_0
4:   while t > TEMP_LIMIT do
5:     for all ITER iterations do
6:       Pick one feasible exchange E at random
7:       d = impact of E in objective function
8:       if d < 0 then
9:         Execute E
10:        if new solution better than incumbent then
11:          Update incumbent
12:        end if
13:      else
14:        x = random number in [0, 1]
15:        if x < exp(−d/t) then
16:          Execute E
17:        end if
18:      end if
19:    end for
20:    t = t ∗ ALPHA
21:  end while
22: end while
```

Table 2.15: Parameters for the SA.

Parameter	**Value**
t_0	2000
TEMP_LIMIT	500
ITER	2500
ALPHA	0.95
p	0.2

2.5.3 Computational Results

We have tested our solution approach on the TUP instances and also on the 2006 MLB Schedule. The results are summarized in Section 2.6.2 along with the results of the Genetic Algorithm presented in the next section.

2.6. Locally Optimized Crossover for TUP

In this section we present a Genetic Algorithm (GA) to solve the TUP. A GA is an adaptive heuristic search method based on population genetics, and it borrows its vocabulary from that domain. The basic concepts of a GA were primarily developed in [42] and described later in [37]. A GA uses a genetic representation for potential solutions. First an initial population of solutions are created. Then offspring solutions are formed from parent solutions using genetic operators called *crossover*. Solutions are evaluated using a function that rates solutions in terms of their "fitness". Then the population is updated by selecting a subset of parent and offspring solutions. This is continued until a termination criterion is met.

The GA we developed uses a special crossover operator that uses local optimization. Thus, instead of obtaining a random solution from two parent solutions, we obtain a partially optimized solution. Since the offspring obtained is not necessarily the best possible solution that can be obtained from the two parents, we refer to this operator as *locally-optimized crossover*.

2.6.1 Solution Strategy

A GA consists of a population of chromosomes; in essence, a set of character strings that are analogous to the base-4 chromosomes that we see in our own DNA. Chromosomes represent potential solutions to the problem being solved and they evolve over a number of generations and are subject to genetic operators at each generation. These solutions are

Table 2.16: Two feasible solutions for 8 teams and 4 umpires.

						Parent1							
Slots	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Umpire 1	(1,5)	(2,8)	(5,6)	(3,7)	(8,1)	(4,6)	(7,5)	(2,4)	(6,8)	(4,5)	(3,1)	(8,6)	(1,3)	(4,2)
Umpire 2	(4,8)	(5,3)	(1,4)	(8,2)	(6,5)	(2,3)	(1,8)	(3,6)	(5,7)	(2,1)	(7,6)	(3,5)	(6,4)	(8,3)
Umpire 3	(6,2)	(4,7)	(3,8)	(5,4)	(7,2)	(8,5)	(2,6)	(5,8)	(4,1)	(6,3)	(8,4)	(1,2)	(7,8)	(5,1)
Umpire 4	(7,3)	(1,6)	(2,7)	(6,1)	(3,4)	(7,1)	(4,3)	(1,7)	(3,2)	(8,7)	(2,5)	(7,4)	(5,2)	(6,7)

						Parent2							
Slots	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Umpire 1	(1,5)	(2,8)	(5,6)	(8,2)	(3,4)	(7,1)	(4,3)	(1,7)	(3,2)	(8,7)	(2,5)	(7,4)	(5,2)	(6,7)
Umpire 2	(4,8)	(5,3)	(1,4)	(3,7)	(6,5)	(2,3)	(1,8)	(3,6)	(5,7)	(2,1)	(7,6)	(1,2)	(6,4)	(8,3)
Umpire 3	(6,2)	(4,7)	(3,8)	(5,4)	(8,1)	(4,6)	(7,5)	(2,4)	(6,8)	(4,5)	(3,1)	(8,6)	(1,3)	(4,2)
Umpire 4	(7,3)	(1,6)	(2,7)	(6,1)	(7,2)	(8,5)	(2,6)	(5,8)	(4,1)	(6,3)	(8,4)	(3,5)	(7,8)	(5,1)

obtained by means of an encoding/decoding mechanism. Most of the developmental work of GA theory was performed using a binary-coded GA. Historically, it is the most widely used representation. In a binary coding each chromosome is a vector comprised of zeroes and ones, where each bit represents a gene. However different encodings, such as the one used in this GA, are also possible.

Initially, a feasible set of chromosomes are needed, which can be generated randomly or by using a heuristic. Each chromosome has an associated *fitness value*; a set of "best fit" chromosomes from each generation survive into the next generation. The genetic operations applied to chromosomes are *crossover* and *mutation*. Typically, crossover is defined such that two individuals (the parents) combine to produce one or two individuals (the children). The primary purpose of the crossover operator is to transmit genetic material from the previous generation to the subsequent generation. Mutation is a genetic operator that alters one or more gene values in a chromosome from its initial state. The mutation operator introduces a certain amount of randomness to the search. It can help identify solutions that crossover alone might not.

Table 2.17: New solution is obtained by crossover at slot 11 and optimized by solving a matching problem at the crossover slot.

					Before Optimization					Crossover at slot=11					
Slots	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	‖	**11**	**12**	**13**	**14**
Umpire 1	(1,5)	(2,8)	(5,6)	(3,7)	(8,1)	(4,6)	(7,5)	(2,4)	(6,8)	(4,5)	‖	(2,5)	(7,4)	(5,2)	(6,7)
Umpire 2	(4,8)	(5,3)	(1,4)	(8,2)	(6,5)	(2,3)	(1,8)	(3,6)	(5,7)	(2,1)	‖	(7,6)	(1,2)	(6,4)	(8,3)
Umpire 3	(6,2)	(4,7)	(3,8)	(5,4)	(7,2)	(8,5)	(2,6)	(5,8)	(4,1)	(6,3)	‖	(3,1)	(8,6)	(1,3)	(4,2)
Umpire 4	(7,3)	(1,6)	(2,7)	(6,1)	(3,4)	(7,1)	(4,3)	(1,7)	(3,2)	(8,7)	‖	(8,4)	(3,5)	(7,8)	(5,1)

					After Optimization										
Slots	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	‖	**11**	**12**	**13**	**14**
Umpire 1	(1,5)	(2,8)	(5,6)	(3,7)	(8,1)	(4,6)	(7,5)	(2,4)	(6,8)	(4,5)	‖	(3,1)	(8,6)	(1,3)	(4,2)
Umpire 2	(4,8)	(5,3)	(1,4)	(8,2)	(6,5)	(2,3)	(1,8)	(3,6)	(5,7)	(2,1)	‖	(7,6)	(1,2)	(6,4)	(8,3)
Umpire 3	(6,2)	(4,7)	(3,8)	(5,4)	(7,2)	(8,5)	(2,6)	(5,8)	(4,1)	(6,3)	‖	(8,4)	(3,5)	(7,8)	(5,1)
Umpire 4	(7,3)	(1,6)	(2,7)	(6,1)	(3,4)	(7,1)	(4,3)	(1,7)	(3,2)	(8,7)	‖	(2,5)	(7,4)	(5,2)	(6,7)

Representation and Genetic Operators

A complete schedule of umpires is a natural representation for a solution. We present two example solutions for the 8 team instance in Table 2.16. A game is represented as a pair (i, j) where i is the home team and j is the away team. Rows correspond to umpire schedules and columns correspond to games that are played in the corresponding time slots.We use a crossover operator and a mutation operator to create new offspring solutions, which we discuss next.

Optimized Crossover: We pick two parent solutions and produce a new solution from them. We randomly pick a crossover slot t and copy the schedule of Parent1 for slots 1 to *t-1* and copy the schedule of Parent2 for slots t to the last slot. Before finalizing the schedule of the offspring, we do the following: We fix the partial schedules copied from Parent1. Thus, an umpire i of the offspring has the same exact schedule as the umpire i of Parent1 for slots 1 through *t-1*. But, the rest of the schedule segments that are copied from Parent2

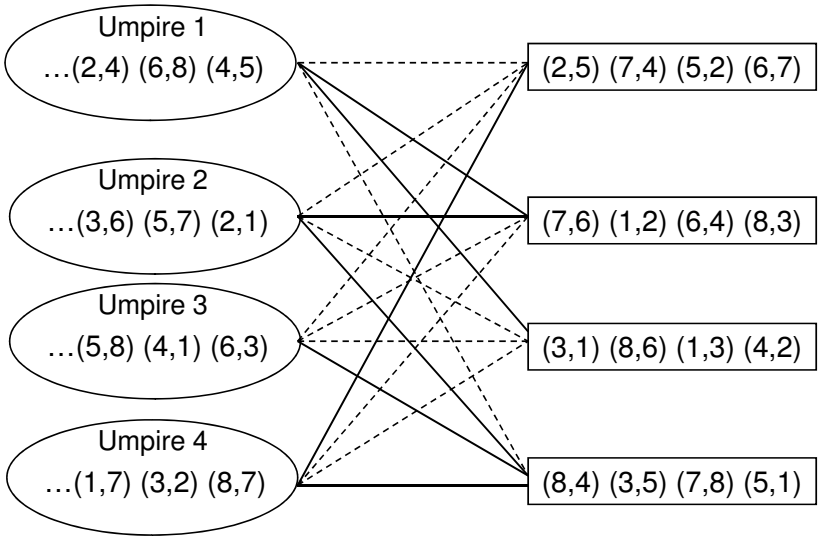

Figure 2.3: The matching problem that is solved. The solid edges are feasible assignments, whereas the dashed lines are penalized for their constraint violations.

(from slots t through the last slot) are allowed to be exchanged among the umpires. To find the best assignment of these schedule segments, we solve a bipartite matching problem. In this matching problem, the partitions are Umpires (with fixed schedules for slots 1 to *t-1*) and partial schedule segments for slots t to the last slot. We place all edges between the partitions, which means all possible assignments are allowed. Cost of an edge between an Umpire u and a Segment s is $Distance(k, i) + Penalty(u, s)$. In this cost function, k is the venue that Umpire u is assigned in slot $t - 1$ and i is the venue of the game in slot t of segment s. Thus, $Distance(k, i)$ is the distance between cities k and i. For every constraint violated by assigning Umpire u to Segment s, the cost of the edge is increased by a high penalty, which makes up the *Penalty(u,s)* term in the cost function.

We illustrate how the crossover operator works in Table 2.17. The crossover operator is applied to the parent solutions in Table 2.16 at slot $t = 11$ to form this new solution. We copy Parent1's schedule for slots 1 to 10 and copy Parent2's schedule for slots 11 to 14. To finalize the new schedule we solve a matching problem on a bipartite graph as

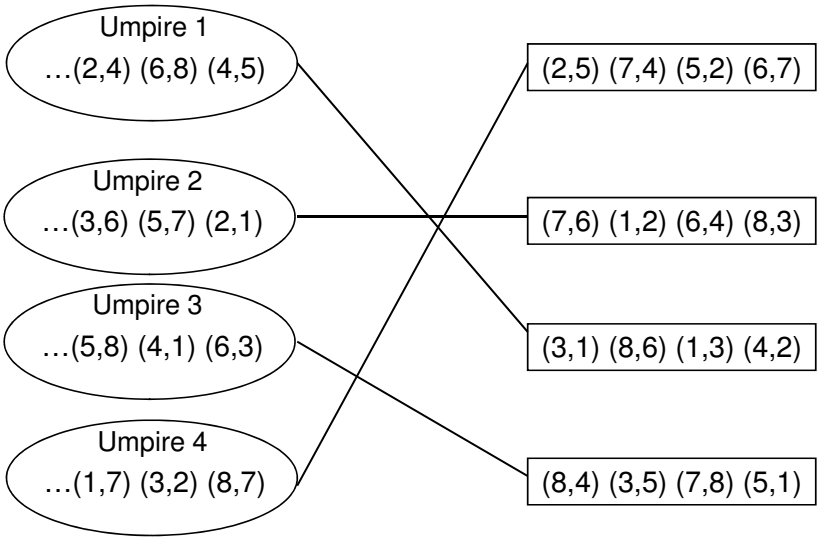

Figure 2.4: The optimum solution to the matching problem.

illustrated in Figure 2.3. For this instance we assume that $d_1 = d_2 = 0$. Thus, *Constraint 4* imposes that an umpire can not visit the same home venue more than once in any four consecutive games. *Constraint 5*, on the other hand, imposes that an umpire can not see a team more than once in any two consecutive games. In this figure, the solid edges are feasible assignments, whereas the dashed edges are penalized for their constraint violations. For instance, the edge between Umpire 1 and the first segment ((2,5),(7,4),(5,2),(6,7)) has a cost equal to $Distance(4,2) + Penalty(Umpire1, Segment1)$, where $Distance(4,2)$ is the distance between cities 4 and 2, which are the venues Umpire 1 would be visiting in slots 10 and 11, respectively. There are two constraint violations in this assignment that are penalized. First, Umpire 1 would visit city 2 in both slots 8 and 11, which violates *Constraint 4*. Second, Umpire 1 would see team 5 in slots 10 and 11, which violates *Constraint 5*. *Constraint 3* is not violated as Umpire 1 will be visiting all the cities in the new solution.

The optimum solution to the matching problem is illustrated in Figure 2.4. Fortunately, we were able obtain a feasible matching that consist of only solid edges. Based on this matching, the new offspring generated is the one illustrated in the lower part of Table 2.17.

The total cost of the parent solutions are 34432 and 35097, respectively and the cost of the new solution after optimization is 34773. Before optimization, the new solution would be an infeasible solution violating the constraints several times.

Mutation: We randomly select two games in a randomly selected slot t and flip the umpires assigned to them.

Initial Population of Solutions

The initial set of solutions are formed by using the GMH with Backtracking described in Section 2.5.1. In each slot, a matching problem is solved and games in that slot are assigned to umpires using either the best or second best matching with equal probability. This probabilistic assignment allows us to form a diverse set of quality solutions.

Evaluation Function

We use an evaluation function that consists of two parts. One is the total travel distance of umpires. The other is the penalties for the constraint violations.

The Algorithm

1. Set $time = 0$
2. Form $N_{initial}$ solutions to form the set $CurrentSolutions(time)$.
3. While the *Stopping Criteria* is not met do:
   (a) Set $time = time + 1$
   (b) Create $N_{initial}/2$ new solutions by:
      i. Randomly select two solutions from the current population of solutions.
      ii. Randomly select a crossover slot.
      iii. Apply the crossover on the two selected solutions to create a new solution.

iv. If the new solution created exists in $TempSolutions(time)$ or in $CurrentSolutions(time-1)$, discard it and goto Step 4(b)i. Otherwise add the new solution into the set $TempSolutions(time)$

(c) Replace the inferior solutions in $CurrentSolutions(time-1)$ with the superior solutions in $TempSolutions(time)$ and create $CurrentSolutions(time)$

(d) For every solution S in $CurrentSolutions(time-1)$ apply Mutation to S with $MutationProbability$.

The parameters of the GA have been set empirically: We used an initial population size $N_{initial} = 100$. At each generation we created 50 new solutions by applying the Crossover Operator to the current solutions. We used $MutationProbability = 0.05$. We allowed the algorithm to run for 3 hours.

2.6.2 Computational Results

We now present and discuss computational results for both the Simulated Annealing described in Section 2.5 and the GA we presented in this section. We have performed our tests on a set of TUP instances and also on the 2006 MLB Schedule. We compare these results with the results obtained by the improved IP and CP models described in Section 2.4.

Table 2.18 summarizes the results on the smallest 7 instances with $d_1 = d_2 = 0$, which means Constraints 4 and 5 are most restricting. Although both the SA and GA ware able to solve the 4,6 and 8 team instances to optimality fast, neither of them was able to solve the 10 team instance to optimality. The quality of the solution found by the GA for the 10 team instance is better than the solution found by the SA. The 12 team instance is already proven to be infeasible. For the 14 instance, the SA was unable to find a feasible solution whereas the GA was able to find one, but it is not better than the current best known solution, which we found by using the improved CP model. We were unable to find a feasible solution for the 16 team instance using any of the methods developed in this chapter.

Table 2.18: SA and GA results for TUP with $d_1 = d_2 = 0$ and comparison to best known solution.

		Simulated Annealing		Genetic Algorithm	
Teams	**OPT Distance**	**Dist.**	**Time (secs)**	**Dist.**	**Time(secs)**
4	5176	5176	0	5176	0
6	14077	14077	0.5	14077	1
8	34311	34311	60	34311	5
10	48942	50196	228	49075	25

	Best Known	**Simulated Annealing**		**Genetic Algorithm**	
Teams	**Distance**	**Dist.**	**Time**	**Dist.**	**Time**
12	infeasible	–	–	–	–
14	177684	–	–	182578	3 hrs
16	–	–	–	–	3 hrs

Table 2.19: Results for SA and GA on the relaxations of 14 and 16 team instances, where with $d_1 + d_2 > 0$, and comparison to results of improved IP.

			Improved IP		**Simulated Annealing**		**Genetic Algorithm**	
Teams	**n - $\mathbf{d_1}$**	$\lfloor \frac{\mathbf{n}}{\mathbf{2}} \rfloor$ - d_2	**Dist.**	**Time**	**Dist.**	**Time**	**Dist.**	**Time**
14	6	3	182531	24 hrs	180697	3 hrs	174919	3 hrs
14	5	3	169012	24 hrs	169173	3 hrs	161607	3 hrs
16	8	2	–	24 hrs	–	3 hrs	–	3 hrs
16	7	3	–	24 hrs	–	3 hrs	184811	3 hrs
16	7	2	–	24 hrs	176527	3 hrs	166077	3 hrs

Table 2.20:]
Improved IP, improved CP, SA and GA results for TUP on the MLB's 2006 game schedule with 30 teams.

			Imp. IP		Imp. CP		SA		GA	
Teams	$\mathbf{n - d_1}$	$\lfloor \frac{\mathbf{n}}{\mathbf{2}} \rfloor$ - d_2	**Dist.**	**Time**	**Dist.**	**Time**	**Dist.**	**Time**	**Dist.**	**Time**
30	5	5	–	out of memory	–	24 hrs	581363	5 hrs	449239	5 hrs

Based on the results we obtained using the different methods presented, it is clear that the 14 and 16 team instances are very difficult instances to solve when we have $d_1 = d_2 = 0$. To further investigate both the GA's and SA's performances on the TUP, we solve the relaxations of these two instances by increasing the values of d_1 and d_2. The results for these relaxations is given in Table 2.19. We also compare the results to the results obtained by the improved IP. The IPs are run for 24 hours, whereas both of the metaheuristics are run for only 3 hours. For the 14 team instance, we see that the GA obtains better results than the IP and SA. For the 16 team instance, neither of the methods were able to find a feasible solution for the relaxation with $\mathbf{n - d_1} = 8$ and $\lfloor \frac{\mathbf{n}}{\mathbf{2}} \rfloor$ - $d_2 = 2$. For the relaxation of the 16 team instance with $\mathbf{n - d_1} = 7$ and $\lfloor \frac{\mathbf{n}}{\mathbf{2}} \rfloor$ - $d_3 = 2$, GA is the only method to find a feasible solution. For the remaining relaxation of 16 team instance with $\mathbf{n - d_1} = 7$ and $\lfloor \frac{\mathbf{n}}{\mathbf{2}} \rfloor$ - $d_2 = 2$, GA obtained a better solution than SA.

As stated before, TUP is an abstraction of the MLB-USP, which is defined on a 30 team league and a 52 week game schedule. To resonate TUP and MLB-USP, we created an instance of TUP on this set of teams and the 2006 MLB game schedule. We tried to solve this instance using the improved IP and CP models and the GA and SA. The results are given in Table 2.20. We see that GA outperformed all others on this instance. We should note that the time for the SA and GA do not include the time for obtaining the initial solution. Interestingly, it takes a lot more time for the GMH with Backtracking to find solutions for this 30 team instance. For the SA, it took about 1 hour to find the initial solution. For the

GA, it took about 19 hours to find 10 initial solutions. A possible explanation for the long running times for the matching heuristic is that we solve the matching problems by solving an math program. Using network algorithms to solve these matching problems will likely decrease these solution times significantly.

2.7. Summary and Future Research

Traveling Umpire Problem is a sports scheduling problem inspired by the real-life needs of the officials of Major League Baseball and it is an abstraction of the real MLB Umpire Scheduling Problem. In this chapter, we defined and extensively studied TUP. We developed exact and approximate solution methods and tested these on benchmark instances. TUP is a highly constrained scheduling problem and we show that even finding feasible solutions is very difficult.

We formulated TUP using IP and CP techniques and our initial formulations are shown to be ineffective as the size of the instances grow. We then introduced a new approach for finding good solutions. We showed how to generate Benders cuts during the execution of a simple Greedy Heuristic. These cuts enforce feasibility requirements that allow the Greedy Heuristic to avoid infeasibilities. This method enables the simple Greedy Matching Heuristic to produce quality feasible solutions very fast. We introduced two Very Large Neighborhoods, 3-Umpire and 3-Slot Neighborhoods, to search the solution space for the Traveling Umpire Problem. We showed that by searching these neighborhoods, guided by the Bender's cuts, we can obtain an initial feasible solution. Moreover, after finding an initial feasible solution, we can improve the quality of the solution by searching the 3-Umpire Neighborhood.

We tried different methods to improve the initial IP and CP formulations. Although the results have been significantly improved by these attempts, we see that both approaches still become ineffective as we increase the size of the instances.

We demonstrated that the Simulated Annealing and Genetic Algorithms we developed are simple to implement and provide very good results much faster than the IP and CP models for most of the instances. Both SA and GA were able to find a solution for the largest instance that has 30 teams and based on the 2006 MLB schedule.

We would like to conclude this chapter by pointing out some future research directions:

2.7.1 Strengthening the Bender's Cuts

The Bender's cuts presented in Section 2.3 can be further strengthened by the following observation: suppose GMH is stuck at slot t and variable $assigned(u, i, s)$ is in a cut, where $assigned(u, i, s)$ equals to 1 if umpire u is at venue i in slot s and equals to zero otherwise. Let also team j be the opponent of i in slot t. Then all variables corresponding to (u, i) assignments in slots $t-(n-d_1)+1$ through t can be added to the same cut as well. Similarly, variables corresponding to all (u, j) assignments in slots $t - (\lfloor \frac{n}{2} \rfloor$ - $d_2) + 1$ through t can be added to the same cut as well. This and similar approaches can strengthen the Bender's Cuts we developed and make the GBNS algorithm more effective.

2.7.2 Solving the multi-objective TUP

In this thesis, we only focused on solving the single objective version of the TUP. One research direction is modifying the current solution methods we presented or develop new methods to solve the multi-objective version TUP-M. One natural way of handling the multi-objective version is using the GA developed. The GA uses a population of solutions in each iteration instead of a single solution. Thus, the outcome of a GA is also a population of solutions, making GAs suitable for solving multi-objective optimization problems, such as TUP-M. Since the ideal strategy for multi-objective optimization requires multiple trade-off solutions to be found, a GA's population-approach can be suitably utilized so that it finds multiple solutions in a single simulation run.

2.7.3 Bridging Local Search and Genetic Algorithms

The crossover operator we used in the GA can also be used as a local search operator on a single solution. This fact is an interesting resemblance between two different solution methods and can be further investigated.

2.7.4 Optimized Crossovers

We developed a locally optimized crossover operator that generates good offspring from two parent solutions by using optimization techniques at each crossover. That operator used a slot t as a crossover point and we copied parts of schedules before and after that crossover point. We can utilize a similar approach to develop another optimized crossover operator that operates in the *umpire* dimension instead of the *slot* dimension. That is, a crossover point will be an umpire u, and full umpire schedules will copied before and after the crossover point from the two parents.

2.7.5 Graph Coloring and TUP

TUP is an interesting problem as we can find resemblances with other problems including Graph Coloring Problem(GCP). In fact, we show that the feasibility version of the TUP, after relaxing Constraint 3, can be represented as a GCP on a special graph G, where each color corresponds to an umpire. The nodes of G are the games. The edge set is constructed as follows:

1) For each slot, an edge is placed between every game, forming a clique on n nodes.

2) A game (i,j) in slot s is connected to a game (i,k) in slot t, where $s \neq t$, if $\mid t-s \mid < n-d_1$.

3) A game (i,j) in slot s is connected to games (i,k), (k,i), (j,k), (k,j) in slot t, where $s \neq t$, if $\mid t-s \mid < \lfloor \frac{n}{2} \rfloor$ - d_2.

It is not difficult to see that a feasible coloring on G corresponds to a feasible umpire schedule. In a feasible coloring, every node is colored, so the first constraint is satisfied. First set of edges ensure that each every umpire works exactly one game per slot, so Constraint

2 is satisfied. Second set of edges ensure that no umpire is in a home site more than once in any $n - d_1$ consecutive slots. Finally, third set of edges ensure that no umpire sees a team more than once in any $\lfloor \frac{n}{2} \rfloor$ - d_2 games.

In this chapter, we clearly showed that even finding a feasible solution becomes very difficult as we increase the size of the instances. Because of that, reformulating TUP as a GCP and using state of the art solution methods developed for GCP might enable us to find feasible solutions for larger instances.

Chapter 3

A Large Neighborhood Search Heuristic for Graph Coloring

Graph coloring is one of the central problems in graph theory, has direct applications in practice [5, 59], and is related to many other problems such as computer register allocation [2], bandwidth allocation [34] and timetabling [57, 78, 24]. Given an undirected graph G=(V, E), a *coloring* f of G is an assignment of a color to each vertex. A *proper (or feasible) coloring* is a coloring such that for each edge $(i, j) \in E$, vertices i and j have different colors. A *conflict* is the situation when two adjacent vertices have the same color assigned to them. We say that a coloring is *improper (or infeasible)* if there exists at least one conflict. The *conflict graph* of a graph G is the graph induced by the vertices that are incident to the conflicts in G.

The decision version of the problem asks whether for a graph G a proper coloring with K colors can be found, where K is given. A *minimum coloring* of G is a feasible coloring with the fewest different colors, which also defines the optimization version of the problem. One approach to solving the optimization problem, which is also the approach used in this study, is to treat it as a sequence of decision problems: Start with an initial number of colors K. If the answer to the decision problem is "*no*", increase K by one and repeat the

process until a feasible coloring exists for some K.

It is well known that graph coloring is a hard combinatorial optimization problem [35], and exact solutions can be obtained for only small instances [64]. Therefore, heuristic algorithms are used to solve large instances. In this paper we introduce a new local search algorithm that searches large neighborhoods, based on ideas introduced by Boykov et al.[12].

The rest of the paper is organized as follows. In Section 1 we shortly review the known neighborhood search methods and introduce our very large neighborhood approach. We describe our algorithm in Section 2 and present the experimental results in Section 3. The conclusion is given in Section 4.

3.1. Local Search for Graph Coloring

Local search is based on the concept of a neighborhood. A *neighborhood* of a solution S is a set of solutions that are in some sense close to S, i.e., they can be easily computed from S or they share a significant amount of structure with S.

Local search for the GCP starts at some initial, improper coloring and iteratively moves to neighboring solutions, trying to reduce the number of conflicts. It is clear that the larger the neighborhood, the better is the quality of the solutions that can be reached in one single move. At the same time, the larger the neighborhood, the longer it takes to search the neighborhood at each iteration.

In this paper, we investigate a new local search method that uses very large scale neighborhoods. This is one of the first attempts to solve the GCP using local search in large neighborhoods. The only other large neighborhood searches we are aware of are due to Chiarandini et al.[17] and Avanthay et al.[3]. Interested reader can see a recent survey of local search methods for graph coloring problem by Galinier and Hertz[31].

To make the notion of *large neighborhood* clear, we'll first introduce the well known *1-exchange* and *2-exchange neighborhoods*, which are small polynomially sized neighborhoods.

Given a coloring, a *1-exchange move* changes the color of exactly one node and a *2-exchange move* swaps the colors of two vertices. The corresponding neighborhoods for these moves are the set of colorings that can be obtained by performing a single move.

We consider the neighborhoods proposed by Boykov et al. [12] for energy minimization problems in computer vision, and use these neighborhoods to solve the GCP. In the following subsections, we formally describe these neighborhoods and the corresponding moves, which are explained best in terms of partitions. Then we describe how to find the optimal moves by using graph cuts. The structures of the graphs, the cuts on these graphs, and the properties of the cuts are also explained in detail.

3.1.1 Moves and Neighborhoods

The first neighborhood is the α-β-swap: for a pair of colors $\{\alpha, \beta\}$, this move exchanges the colors between an arbitrary set of vertices colored α and another arbitrary set colored β. The second neighborhood we consider is α-expansion: for a color α, this move assigns the color α to an arbitrary set of vertices.

The GCP can be represented as a partitioning problem, in which a feasible coloring f corresponds to a partition of the set of vertices into K sets such that no edge exists between two vertices from the same color class. Let $\mathbf{V} = \{V_l | l \in L\}$ be such a partition, where L is the set of colors and $V_l = \{v \in V | f(v) = l\}$ is the subset of vertices assigned color $l \in L$.

Given a pair of colors (α, β), a move from a partition $\mathbf{V}$ (coloring f) to a new partition $\mathbf{V}'$ (coloring f') is called an α-β-swap if $V_l = V_l'$ for any color $l \neq \alpha, \beta$. This means that the only difference between $\mathbf{V}$ and $\mathbf{V}'$ is that some vertices that were colored α in $\mathbf{V}$ are now colored β in $\mathbf{V}'$, and some vertices that were colored β in $\mathbf{V}$ are now colored α in $\mathbf{V}'$.

Given a color α, a move from a partition $\mathbf{V}$ (coloring f) to a new partition $\mathbf{V}'$ (coloring f') is called an α-expansion if $V_\alpha \subset V_\alpha'$ and $V_l' \subseteq V_l$ for any label $l \neq \alpha$. In other words, an α-expansion move allows any set of vertices to change their colors to α.

3.1.2 Size of the Neighborhoods

For an α-β-swap, each vertex either keeps its current color or switches to the other one. Since each partition has $\Omega(n)$ vertices, the possible solutions that can be reached by one swap move is $2^{\Omega(n)}$.

For an α-expansion, each vertex that is not colored α either keeps its old color or acquires the new color α. Since there are $\Omega(n)$ such vertices, the possible solutions that can be reached by one expansion move is $2^{\Omega(n)}$. These imply:

Lemma 1 *The size of both neighborhoods is* $2^{\Omega(n)}$.

3.1.3 Graph Cuts

The important part of the local search algorithms, which are presented in the following sections, is efficiently finding the best neighboring solution to the current solution by using graph cuts. Let $G = (V, E)$ be a connected and undirected edge weighted graph. A *cut* C of G is a minimal subset of E, which increases the number of connected components by exactly one. The *weight* (or *cost*) of a cut C is the sum of the weights of edges in the cut and is represented by $w(C)$. A *maximum cut* (or a *maxcut*) is then defined as a cut of maximum weight.

3.1.4 Finding the Optimal Swap Move

Given a coloring f and a pair of colors $\{\alpha, \beta\}$, we want to find a coloring $\hat{f}$ that minimizes the number of conflicts over all colorings within one α-β-swap of f. Our technique is based on computing a coloring that corresponds to a maximum cut on the subgraph $G^{\alpha\beta} = (V^{\alpha\beta}, E^{\alpha\beta})$, which is a clique over the vertices colored with α or β in f. For all $(i, j) \in E^{\alpha\beta}$, we assign a weight equal to one if $(i, j) \in E$ and a weight equal to zero if $(i, j) \notin E$. The latter ensures that $G^{\alpha\beta}$ is connected. The structure of the graph $G^{\alpha\beta}$ is illustrated in Fig. 3.1.

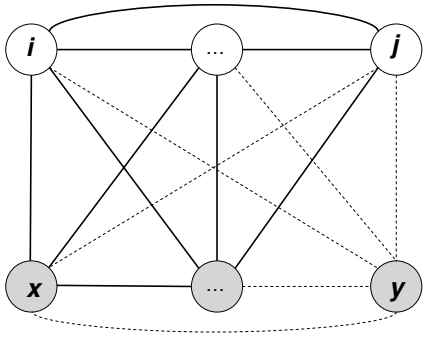

Figure 3.1: An example of $G^{\alpha\beta}$. The set of vertices are $V^{\alpha\beta}= V_\alpha \cup V_\beta$ where $V_\alpha = \{i, ..., j\}$ and $V_\beta = \{x, ..., y\}$. Solid edges are induced edges and have weight 1. Dashed edges are artificial edges and have weight 0, which ensure that $G^{\alpha\beta}$ is connected.

Every edge, with a weight 1, between the vertices of the same color is a conflict. Every swap move defines a new bipartition of the vertex set $V^{\alpha\beta}$, possibly with a different number of conflicts. Notice that every cut in this graph defines a swap move that results a bipartition of the vertex set, thus a new coloring. In order to obtain the optimal swap move that results with minimum number of conflicts, we need to minimize the total weight of edges within the partitions. Notice that this is equivalent to maximizing the total weight of edges between the two partitions, which is equivalent to solving a maxcut problem on $G^{\alpha\beta}$. After finding a maxcut, the vertices in one partition are going to be colored α, and the vertices in the other partition will be colored β. The selection of which partition will be colored α is arbitrary. This implies:

Theorem 1 *Let $G^{\alpha\beta}$ be constructed as described above for a given f and $\{\alpha, \beta\}$ and let T be the total weight of edges in $G^{\alpha\beta}$. A coloring f^C corresponding to a cut C on $G^{\alpha\beta}$ is one α-β-swap away from the initial coloring f. Moreover the optimal α-β-swap move is equivalent to a maxcut C^* in $G^{\alpha\beta}$ and the number of conflicts within $G^{\alpha\beta}$ for the new coloring f^{C^*} is x if $w(C^*) = T - x$.*

3.1.5 Finding the Optimal Expansion Move

Given an input coloring f and a color α, we want to find a coloring $\hat{f}$ that minimizes the number of conflicts over all colorings within one α-expansion of f. Our technique is based on computing a coloring that corresponds to a maximum cut on the graph $G^\alpha = (V^\alpha, E^\alpha)$. The structure of this graph is determined by the current partition $\mathbf{V}$ and by the color α, so the graph dynamically changes after each iteration.

The structure of the graph is illustrated in Fig. 3.2. The set of vertices include all vertices $v \in V$. Moreover it includes two terminals α and $\overline{\alpha}$, which are auxiliary vertices representing the color α in consideration and the rest of the colors, respectively. In addition, we have six types of auxiliary vertices. For each edge that is incident to two vertices with color α, we create an auxiliary vertex of type A_1. For each edge that is incident to exactly one vertex with color α, we create an auxiliary vertex of type A_2. For each adjacent vertex pair such that neither vertex in the pair is colored with α, we create two auxiliary vertices of types B_1 and B_2 if the pair has different colors. If they are colored with the same color, say γ, we create two vertices of types D_1 and D_2.

We now explain the way we connect the vertices by edges with different wights. The weights assigned to these edges are summarized in Table 3.1. The two terminals are connected by an edge with a very high weight M to ensure that the maxcut that is found separates the two terminals α and $\overline{\alpha}$. Each vertex $v \in V$ is connected by an edge to the terminals α and $\overline{\alpha}$. Each pair of adjacent vertices $\{i, j\} \in V$ is connected by edges to the auxiliary vertices corresponding to that pair. Each pair of adjacent vertices such that neither of them are colored with α are connected by an edge. In addition to these, type $A1$, $A2$, B_1, and D_1 vertices are connected by an edge to the terminal α, and type B_2 and D_2 vertices are connected by an edge to the terminal $\overline{\alpha}$. As a result, each adjacent vertex pair and the auxiliary vertices corresponding to the pair and the edges incident to these vertices form four different structures, which we call as *gadgets*. The four different gadgets that correspond to four edge types of the original graph G are illustrated in Fig. 3.2. Formally,

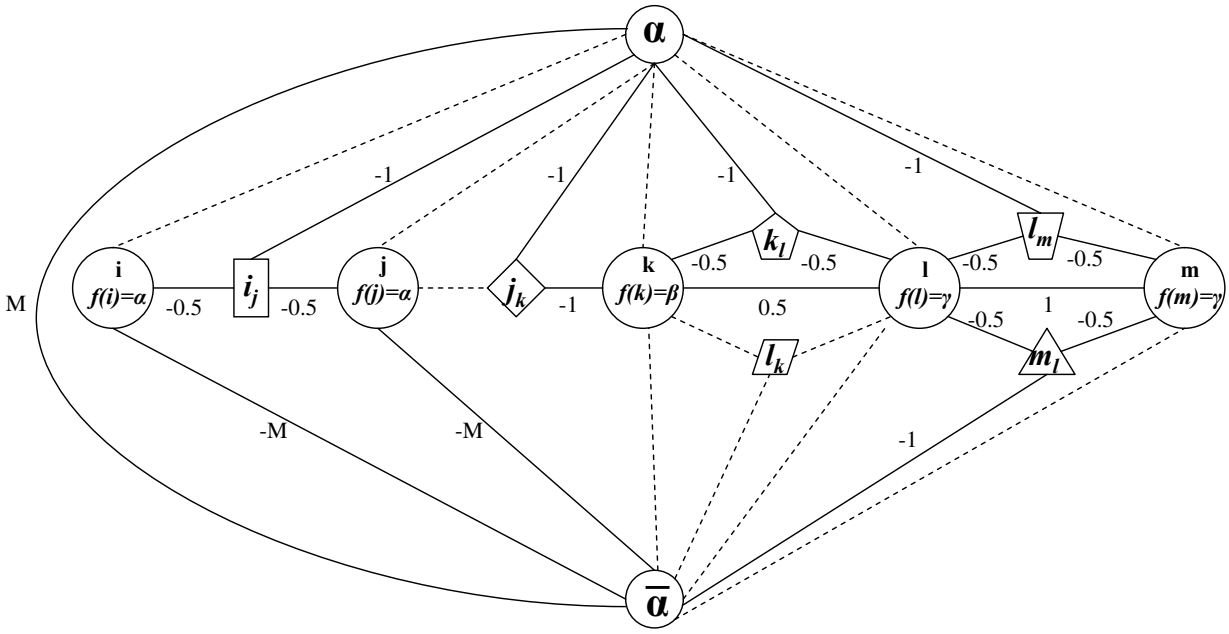

Figure 3.2: An example of G^{α}. Dashed edges have weight 0. The type sets for auxiliary vertices: T=$\{\alpha, \overline{\alpha}\}$, A_1=$\{i_j\}$, A_2=$\{j_k\}$, B_1=$\{k_l\}$, B_2=$\{l_k\}$, D_1=$\{l_m\}$, D_2=$\{m_l\}$

for an edge (i, j), there are four possible situations depending on the colors of the vertices incident to those edges:

1. $f(i) = f(j) = \alpha$
2. $f(i) = \alpha$, $f(j) \neq \alpha$, or $f(i) \neq \alpha$, $f(j) = \alpha$
3. $f(i) \neq \alpha$, $f(j) \neq \alpha$, $f(i) = f(j)$
4. $f(i) \neq \alpha$, $f(j) \neq \alpha$, $f(i) \neq f(j)$

Any cut C on G^{α}, which separates the two terminals α and $\overline{\alpha}$, must include exactly one of the edges that connect $v \in V$ to the terminals. This defines a natural coloring f^C corresponding to a cut C on G^{α}. Formally,

$$f^C(v) = \begin{cases} \alpha, & \text{if } (v, \alpha) \in C \\ f(v), & \text{if } (v, \overline{\alpha}) \in C \end{cases}$$

Table 3.1: The weights assigned to the edges in the graph presented in Fig. 3.2.

edge	weight	for	example(Fig. 3.2)
(v,α)	0	$v \in V$	(i,α),(j,α),(k,α),(l,α),(m,α)
$(v,\overline{\alpha})$	$-M$	$v \in V_\alpha$	$(i,\overline{\alpha})$,$(j,\overline{\alpha})$
$(v,\overline{\alpha})$	0	$v \in V \backslash V_\alpha$	$(k,\overline{\alpha})$,$(l,\overline{\alpha})$,$(m,\overline{\alpha})$
(a,α)	-1	$a \in A_1, A_2, B_1, D_1$	(i_j,α),(j_k,α),(k_l,α), (l_m,α)
$(b,\overline{\alpha})$	0	$b \in B_2$	$(l_k,\overline{\alpha})$
$(c,\overline{\alpha})$	-1	$c \in D_2$	$(m_l,\overline{\alpha})$
(v,a)	-0.5	$v \in V_\alpha$, $a \in A_1$	(i,i_j), (i_j,j)
(v,a)	-1	$v \in V \backslash V_\alpha$, $a \in A_2$	(j_k,k)
(v,a)	0	$v \in V_\alpha$, $a \in A_2$	(j,j_k)
(v,b)	-0.5	$v \in V \backslash V_\alpha$, $b \in B_1, D_1, D_2$	(k,k_l),(l,k_l),(l,l_m), (l,m_l),(m,l_m),(m,m_l)
(v,b)	0	$v \in V \backslash V_\alpha$, $b \in B_2$	(l_k,k), (l_k,l)
(v,w)	0.5	$v,w \in V \backslash V_\alpha$, $f(v) \neq f(w)$	(k,l)
(v,w)	1	$v,w \in V \backslash V_\alpha$, $f(v) = f(w)$	(l,m)
$(\alpha,\overline{\alpha})$	M	$(\alpha,\overline{\alpha})$	$(\alpha,\overline{\alpha})$

In other words, a vertex v is assigned color α if the cut C separates v from the terminal α and, v is assigned its old color $f(v)$ if C separates v from $\overline{\alpha}$. Clearly this implies:

Lemma 2 *A coloring f^C corresponding to a cut C on G^α, which separates the two terminals α and $\overline{\alpha}$, is one α-expansion away from the initial coloring f. Also, an α-expansion move is equivalent to a cut C on G^α, which separates the two terminals α and $\overline{\alpha}$.*

For each of the four gadgets, a maxcut C on G^α severs some of the edges of the gadgets, and the sum of the weights of the edges severed is consistent with whether the corresponding edge (i,j) is a conflict or not in f^C.

Property 1 For any maxcut C and for any edge $(i,j) \in E$ such that at least one of i and j is colored with α in f

a) *If (α,i), $(\alpha,j) \in C$ then either $(\alpha,i_j) \in C$ or (i,i_j), $(i_j,j) \in C$*

b) *If $(\overline{\alpha},i)$, $(\overline{\alpha},j) \in C$ then no other edge from the gadget is in C*

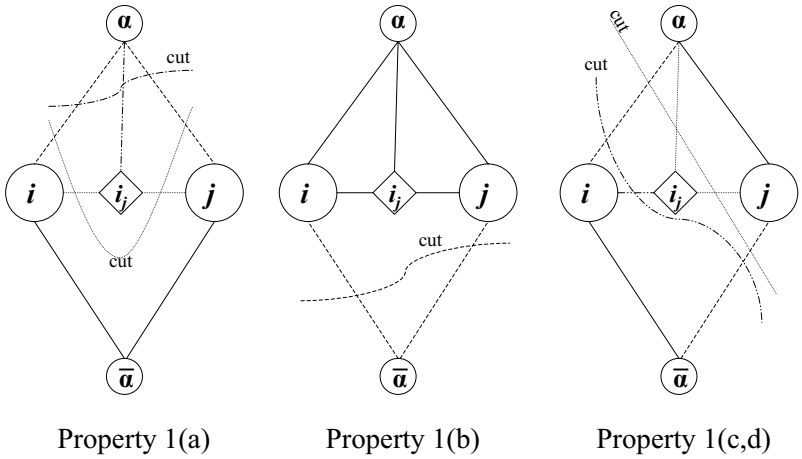

Figure 3.3: Properties of a maxcut C on G^α for two vertices i,$j \in V$ such that at least one of them is colored with α.

c) *If* (α, i), $(\overline{\alpha}, j) \in C$ *then either* $(i, i_j) \in C$ *or* (α, i_j), $(i_j, j) \in C$

d) *If* (α, j), $(\overline{\alpha}, i) \in C$ *then either* $(i_j, j) \in C$ *or* (α, i_j), $(i, i_j) \in C$

Property 1 follows from the maximality of $w(C)$ and it is illustrated in Fig. 3.3.

In the case that $f(i) = f(j) = \alpha$, since the weight of the edges that connect i and j to $\overline{\alpha}$ has weight $-M$, the only maxcut possible is one of the cuts described in Property 1(a). Since both of the cuts separate i and j from α, the colors of i and j stay unchanged: $f^C(i) = f^C(j) = \alpha$. If $(\alpha, i_j) \in C$, or if (i, i_j), $(i_j, j) \in C$ then the cost is -1. In both cases, the cost incurred is truly consistent with the fact that (i, j) is a conflict in f^C.

In the case that one of i and j is colored with α in f, assume w.l.o.g. $f(i) = \alpha$, $f(j) = \beta$, the possible cuts are the ones described in Property 1 (a) or (c). The cuts that sever $(i, \overline{\alpha})$ are not possible since the weight of $(i, \overline{\alpha})$ is $-M$. The cost of the cuts having Property 1(a) is -1, which is consistent with the fact that (i, j) is a conflict in f^C. The cost of the cuts having Property 1(c) is 0, which is consistent with the fact that (i, j) is not a conflict in f^C.

Property 2 For any maxcut C and for any edge $(k, l) \in E$ such that $f(k) \neq \alpha$, $f(l) \neq \alpha$:

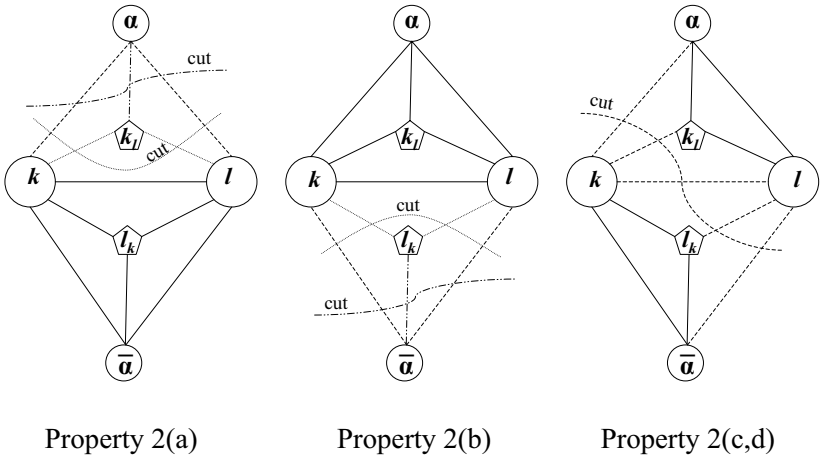

Figure 3.4: Properties of a maxcut C on G^α for two vertices k,$l \in V$ such that none of them is colored with α.

a) *If* (α, k), $(\alpha, l) \in C$ *then either* $(\alpha, k_l) \in C$ *or* (k, k_l), $(k_l, l) \in C$

b) *If* $(\overline{\alpha}, k)$, $(\overline{\alpha}, l) \in C$ *then either* $(\overline{\alpha}, l_k) \in C$ *or* (k, l_k), $(l_k, l) \in C$

c) *If* (α, k), $(\overline{\alpha}, l) \in C$ *then* (k, k_l), $(l_k, l) \in C$

d) *If* (α, l), $(\overline{\alpha}, k) \in C$ *then* (k, l_k), $(k_l, l) \in C$

Property 2 follows from the maximality of $w(C)$ and from the fact that no subset of C is a cut. Property 2 is illustrated in Fig. 3.4.

In the case that $f(k) = f(l) = \beta$, all the cuts described in Property 2 are possible. The cost of the cuts that have Property 2(a) or (b) is -1, which is consistent with the fact that (k, l) is a conflict in f^C in both cases. The cost of the cuts that have Property 2(c) or (d) is 0, which is consistent with the fact that (k, l) is not a conflict in f^C in either case, since the color of exactly one of k, l is changed to α.

In the case that $f(k) = \beta$, $f(l) = \gamma$, all the cuts described in Property 2 are possible. The cost of the cuts that have Property 2(a) is -1, which is consistent with the fact that (k, l) is a conflict in f^C. The cost of the cuts that have Property 2(b), (c) or (d) is 0, which is consistent with the fact that (k, l) is not a conflict in f^C in those cases, since the color of

at least one of k, l is not changed to α. Lemma 2, Property 1 and Property 2 implies:

Theorem 2 *A coloring f^{C^*} corresponding to a maxcut C^* on G^α is one α-expansion away from the initial coloring f. Moreover the optimal α-expansion move is equivalent to a maxcut C^* in G^α and the number of conflicts for the new coloring f^{C^*} is x if $w(C^*) = M - x$.*

3.2. Algorithms

In this section we first introduce swap-move, check-bipartite and expansion-move algorithms. These three algorithms are used as subroutines in the expansion-swap algorithm, which is the main algorithm.

3.2.1 Swap-Move Algorithm

Input: A coloring f of G with K colors.

1. Set Success:=0, Any_Imp(swap):=0
2. Set Improvement(swap):= 0
3. For each pair of colors $\{\alpha, \beta\} \subset L$
   - 3.1 Find $\hat{f}$ that minimizes the number of conflicts among all possible new colorings within one α-β-swap of f
   - 3.2 IF the number of conflicts is reduced, f:= $\hat{f}$, Improvement(swap):=1, Any_Imp(swap):=1
4. IF f has no conflicts, return f with Success := K
5. IF Improvement(swap) = 1, goto 2
6. Return f with Success:= 0 & Any_Imp(swap).

3.2.2 Expansion-Move Algorithm

Input: A coloring f of G with K colors.

1. Success:=0, Any_Imp(expansion):=0
2. Set Improvement(expansion):= 0
3. For each color $\alpha \in L$
   3.1 Find $\hat{f}$ that minimizes the number of conflicts among all possible new colorings within one α-expansion of f
   3.2 IF the number of conflicts is reduced,
   f:= $\hat{f}$, Improvement(expansion):=1, Any_Imp(expansion):=1
4. IF f has no conflicts, return f with Success:= K & Any_Imp(expansion)
5. IF Improvement(expansion)= 1, goto 2
6. Return f with Success:= 0 & Any_Imp(expansion).

3.2.3 Check-Bipartite Algorithm

If the conflict graph of an infeasible coloring f of G with K colors is bipartite, one can remove all conflicts without creating new conflicts and end up with a feasible $K+1$ coloring by coloring the vertices of one partition with a new color $K+1$. A short algorithm based on this observation is given below.

Input: A coloring f of G with K colors.

1. If f has conflicts, let $G^{'} = (V^{'},E^{'})$ be the subgraph induced by the conflicting vertices
   1.1 Starting from an arbitrary source vertex, color the vertices and their neighbors in alternation with colors α and β.
   1.2 If the resulting coloring is proper, then $G^{'}$ is bipartite. Introduce a new color $K+1$ to color the vertices in one partition, return f with Success:= $K+1$.

3.2.4 Expansion-Swap Algorithm

Expansion-swap is the main algorithm that takes advantage of both neighborhoods introduced and uses the swap-move, check-bipartite and expansion-move algorithms as subrou-

tines. Namely, when the swap-move no longer improves the current solution, expansion-move tries to improve the current solution without starting from scratch. When expansion-move is unable to improve the solution, swap-move starts running again. When neither algorithm is able to improve the solution, first we check if the conflict graph is bipartite with the Check-bipartite algorithm. If it is not bipartite, then at Step 8, a new color $K+1$ is introduced by finding $\hat{f}$ that minimizes the number of conflicts among f' within one $(K+1)$-expansion of f.

Introducing a new color by an expansion move is a better approach than introducing it by finding an independent set on the conflict graph, since expansion allows for the creation of new conflicts in exchange for removing more current conflicts. However in the independent set case, coloring adjacent vertices with the new color is not permitted. This is easy to see when we assume that the conflict graph is a dense graph, such as a clique, where the maximum independent set is only one vertex.

Introducing the new color $K+1$ will definitely improve the solution if not actually remove all the conflicts. After this improvement, the swap-move phase will search for a better solution, and the whole cycle repeats until a feasible coloring is found.

Input: A graph G and an initial number of colors K.

1. Randomly color G with K colors, resulting in coloring f
2. Swap-move(G, f, K)
3. IF Success = 0, Expansion-move(G, f, K)
4. IF Success $>$ 0 return f
5. IF Any_Imp(expansion) = 1, Swap-move(G, f, K); ELSE goto 7
6. IF Success $>$ 0 return f;
   ELSE IF Success = 0 & Any_Imp(swap) = 1, goto 3
7. Check-bipartite(G, f, K)

8. IF Success = 0, find $\hat{f}$ that minimizes the number of conflicts among all possible new colorings within one $(K+1)$-expansion of f, set $f := \hat{f}$

9. IF f has no conflicts, return f with Success $:= K+1$; ELSE set $K := K+1$, goto 2.

3.2.5 Finding a Maximum Cut

Given an undirected graph G with edge weights, the *MAX-CUT problem* consists of finding a maxcut of G. MAX-CUT is a well-known NP-Hard problem[35]. Since all our algorithms rely on solving MAX-CUT problems several times, the solution times for our algorithms can be expected to be out of our limits for a local search heuristic. So rather than determining the maxcut at each move, we can find a "good" cut, which has a weight that is close to the weight of a maxcut. This allows us to search the introduced neighborhood approximately and fast. However, since Theorem 2 is dependent on the cut found being a maxcut, we can not use the total weight of the cut found to calculate the number of conflicts. But notice that we can still use any cut to find a new coloring as Lemma 2 holds for any cut. For this reason, we use the cut obtained to define the new coloring and we calculate the actual number of conflicts by checking the adjacency matrix and the new coloring.

There are many heuristic and approximation algorithms that have been computationally tested and/or with theoretical performance guarantee. Goemans and Williamson [36] proposed a randomized algorithm that uses semidefinite programming to achieve a performance guarantee of 0.87856. More recent algorithms for solving the semidefinite programming relaxation are particularly efficient, because they exploit the structure of the MAX-CUT problem. Burer, Monteiro, and Zhang [16] proposed a rank-2 relaxation heuristic for MAX-CUT and described a computer code, called *Circut* [15], that produces better solutions in practice than the randomized algorithm of Goemans and Williamson. Circut does not assume the edge weights are positive. This property is necessary for our algorithm as the graphs created by our algorithm have edges with negative weights. Moreover, since the performance of Circut on many different problems has been shown to be very good, and the

code is available for outside use, we decided to use Circut to solve MAX-CUT problems in our algorithms.

3.3. Experimental Results

We first give the implementation details of our algorithms and describe the instances in this section. We then present our experimental results.

3.3.1 Implementation Details

Our algorithm is implemented in C. Since the MAX-CUT solver code, Circut, is implemented in Fortran90, the input/output transaction between the main code and Circut is made through text files. For large size problems, writing into and reading from files takes very long times. This becomes a serious issue especially for the expansion graphs created for the expansion move since the size of the expansion graphs are much larger than the original graph. To overcome this disadvantage, we used the following strategy for large graphs: Expansion moves were only used for introducing a new color when swap-move is stuck with the current solution. We introduce the new color by finding the best expansion move on the conflict graph rather than the original graph, since the color of non-conflicting vertices do not change during an expansion move. The conflict graph is smaller than the original graph in almost all cases, and becomes smaller as the number of conflicts is reduced during the execution of the algorithm. This observation made it possible for us, not fully but at least partially, to use the expansion move idea for large instances.

In addition to the modification described above, we have made two more changes in the original expansion-swap algorithm in order to decrease the execution time: First, we use the simple 1-exchange moves after Step 5 and Step 9 of the expansion-swap algorithm. Second, after introducing a new color at Step 9, we have looked at the best swap moves only between the new color and the old ones, but not between all the old colors.

3.3.2 Instance Description

We tested the algorithms on some of the benchmark instances proposed for COLOR02/03/04 [83]. Here we briefly describe the instance classes.

le: Leighton graphs with guaranteed coloring size. These are structured graphs generated by a procedure[57].

queen: Given an n by n chessboard, a queen graph is a graph with n^2 nodes, each corresponding to a square of the board. Two nodes are connected by an edge if the corresponding squares are in the same row, column or diagonal.

myciel: Graphs based on the Mycielski transformation. These graphs are difficult to solve because they are triangle free (clique number 2) but the coloring number increases in problem size.

k-Insertions and FullIns: These are a generalization of myciel graphs with inserted nodes to increase graph size but not density.

DSJ: DSJC are standard (n,p) random graphs. DSJR are geometric graphs. These are instances from Johnson [46].

3.3.3 Summary of Results

We run the expansion-swap algorithm on a 450 MHz Sun UltraSPARC-II workstation with 1024MB of RAM. We tested our algorithms on some of the benchmark instances proposed for COLOR02/03/04 [83].

Table 3.2 and Table 3.3 compare the results of the Expansion-Swap algorithm(ES) to the results of some other heuristics. These results are summarized in [83]. Not all heuristics reported their results for all instances. Thus many cells in the table are empty. The columns in the table consist of the name of the graph, number of vertices (n), number of edges (m), density of the graph in percentages (d), optimum solution(OPT), lower bound(LB), results due to (Croitoru et al.(CL)[22], Galinier et al.(GH)[32], Bui and Patel(BP)[14], Phan and Skiena(PS)[72], Chiarandini and Stuetzle(CS)[18], results for expansion-swap

(ES) algorithm, and time in seconds for ES (time).

Of the 68 test instances solved with the Expansion-Swap Algorithm, there are 18 instances with reported chromatic numbers [83]. Of these 18 instances our algorithm found the optimal solution for 10 of them.

The results of the ES algorithm for 32 instances are either equal to the optimal solution, or as good as the best result found by other heuristics listed. The results of ES for these instances are highlighted in bold in Tables 3.2 and 3.3. For 15 instances, ES either could not find the optimal solution or at least one of the other heuristics obtained a better solution. For 14 instances, ES obtained the worst results. And for the remaining 7 instances, we cannot make a comparison as we only have the results of ES but we see that these results are only one more than the clique number of 4 of these 7 instances.

In terms of instance types, we can say that our algorithm performed very well on myciel and FullIns instances, and on school1-nsh and mugg100-25. It also has a good performance on all other graphs except DSJ instances and latin-square-10. For DSJ and latin-square, the quality of the solutions are not as good, especially for the large and dense instances.

In terms of solution times, 16 instances are solved in less than 1 second, 39 instances are solved in more than 1 second but in less than 1 minute, 5 instances are solved in more than 1 minute but less than 4 minutes, 4 instances are solved in more than 4 minutes but in less than 8 minutes, and the remaining 4 instances are solved in more than 8 minutes but in less than 16 minutes.

Figure 3.5 presents the relationship of the solution time with the density of the graphs. The x-axis is the ordered list of 68 instances solved. The instances are ordered in ascending order of density. Instance number 1 is the least dense and instance number 68 is the most. The y-axis is the solution time. As one would expect, the hardest instances are those with high density, though this does not fully explain the heuristic's running time since some high density instances can be solved quickly.

Table 3.2: Comparison of the results of the expansion-swap(ES) algorithm to the results of other heuristics.

Graph	n	m	d	OPT	LB	CL	GH	BP	PS	CS	ES	time
le450_5a	450	5714	6	5	5			5	14		**5**	*3.6*
le450_5b	450	5734	6	5	5			5	13		**5**	*8.6*
le450_5c	450	9803	10								**5**	*4.2*
le450_5d	450	9757	10	5	5				16		**5**	*4.1*
le450_15a	450	8168	8		15	18		15	23	15	*18*	*51.2*
le450_15b	450	8169	8	15	15	18		15	23	15	*18*	*46.4*
le450_15c	450	16680	17		15	27	15		32	16	*25*	*90.8*
le450_15d	450	16750	17		9		15		31	16	*26*	*36.5*
le450_25a	450	8260	8								*26*	*21.3*
le450_25b	450	8263	8								*26*	*19.8*
le450_25c	450	17343	17		25		26		36	26	*32*	*29.3*
le450_25d	450	17425	17		13		26		37	26	*31*	*101*
queen8_8	64	728	36	9	9						*10*	*7.2*
queen8_12	96	1368	30								*13*	*4.5*
queen9_9	81	2112	65					10			*11*	*12.2*
queen10_10	100	2940	59								*13*	*3.5*
queen11_11	121	3960	55					12			*14*	*4.5*
queen12_12	144	5192	50								*15*	*12.8*
queen13_13	169	6656	47					14		14	*16*	*108*
queen14_14	196	8372	44								*17*	*75.9*
queen15_15	225	10360	41					17			*19*	*9.9*
queen16_16	256	12640	39						21	18	*19*	*30.0*
myciel5	47	236	22								**6**	*0.5*
myciel6	95	755	17					7			**7**	*0.9*
myciel7	191	2360	13					8			**8**	*1.4*
1-Insertions_4	67	232	10		4	4		4			*5*	*0.4*
1-Insertions_5	202	1227	6		4	6			6		**6**	*0.9*
1-Insertions_6	607	6337	3			7			15		**7**	*5.1*
2-Insertions_3	37	72	11								**4**	*0.2*
2-Insertions_4	149	541	5	4	4	5		4	5	5	*5*	*0.4*
2-Insertions_5	597	3936	2		4	6			11		**6**	*4.2*
3-Insertions_3	56	110	7								**4**	*0.2*
3-Insertions_4	281	1046	3		3	5			5	5	**5**	*1.2*
3-Insertions_5	1406	9695	1			6			29	6	**6**	*35.4*
4-Insertions_3	79	156	5		3			4			**4**	*0.2*
4-Insertions_4	475	1795	2		3				7		**5**	*2.3*

Table 3.3: Continuation of Table 3.2.

Graph	**n**	**m**	**d**	**OPT**	**LB**	**CL**	**GH**	**BP**	**PS**	**CS**	**ES**	**time**
1-FullIns_3	30	100	23	4	4	4					**4**	*0.2*
1-FullIns_4	93	593	14	5	5	5					**5**	*0.4*
1-FullIns_5	282	3247	8	6	6	6			7		**6**	*1.1*
2-FullIns_3	52	201	15		5	5					**5**	*0.3*
2-FullIns_4	212	1621	7		5	6			7		**6**	*0.7*
2-FullIns_5	852	12201	3		6	7			23		**7**	*9.1*
3-FullIns_3	80	346	11	5	5	6					**6**	*0.4*
3-FullIns_4	405	3524	4		6	7			11	7	**7**	*4.6*
3-FullIns_5	2030	33751	2		6	8			59	8	**8**	*53.7*
4-FullIns_3	114	541	8	7	7	7					**7**	*1.0*
4-FullIns_4	690	6650	3		7	8			19		**8**	*7.6*
4-FullIns_5	4146	77305	1			9				9	*11*	*325*
5-FullIns_3	154	792	7	8	8				8		**8**	*2.2*
5-FullIns_4	1085	11395	2						27		**10**	*11.8*
DSJC125.1	125	736	9	5	5			5	7		*6*	*1.0*
DSJC125.5	125	3891	50		12	20		18	21		*21*	*11.8*
DSJC125.9	125	6961	90		30			42	46		*48*	*50.2*
DSJC250.1	250	3218	10	8	8			9			*10*	*3.3*
DSJC250.5	250	15668	50		13	37		22		28	*36*	*46.9*
DSJC250.9	250	27897	90		35			72	79		*82*	*230*
DSJC500.1	500	12458	10		6	16	12		20	12	*15*	*42.0*
DSJC500.5	500	62624	50		16	66	48	51		50	*61*	*257*
DSJC500.9	500	112437	90		42		126			127	*156*	*838*
DSJR500.1	500	3555	3	12	12	12					**12**	*5.3*
DSJR500.1c	500	121275	97	63	63	56			105		*94*	*418.1*
DSJR500.5	500	58862	47	26	26			129	155	124	*143*	*474*
DSJC1000.1	1000	49629	10		6		20		41		*26*	*50.5*
DSJC1000.5	1000	249826	50		17		84				*111*	*793*
DSJC1000.9	1000	449449	90		54		224				*289*	*797*
latin_sq_10	900	307350	76					101		99	*123*	*901*
school1_nsh	352	14612	24	14	14			14	33		**14**	*29.3*
mugg100_25	100	166	3								**4**	*0.3*

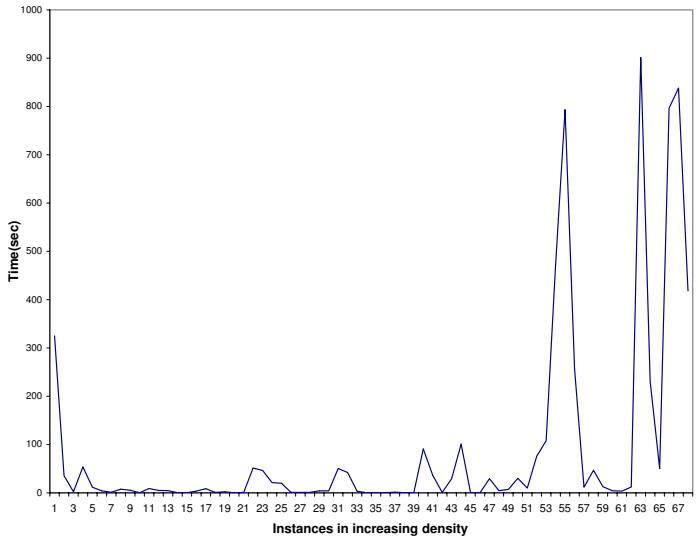

Figure 3.5: Relationship of the solution time with the density of the graphs.

3.4. Summary and Future Research

We studied a new local search algorithm using two very large-scale neighborhoods for the GCP. The first type of move allows us to swap the colors of sets of vertices. The second type of move allows any set of vertices to change their colors to a particular color. The algorithm proposed combines these two types of moves.

The key part of the algorithm is efficiently finding the best neighboring solution to the current solution by solving a MAX-CUT problem. Since MAX-CUT is a hard problem, we considered approximate algorithms that are able to find "good" solutions very fast. It is important to note that the success of the algorithms presented in this paper hinges on fast algorithms that can solve MAX-CUT problems optimally or approximately.

This study is one of the first attempts to solve the GCP using local search in very large neighborhoods. Although we could not fully take advantage of the neighborhoods by solving the MAX-CUT problems optimally, the results we present here are promising. However, further research efforts are still required to make large scale neighborhood techniques fully

competitive.

One possible extension is to use exact or better approximate algorithms and to fully integrate them with the main code to solve the MAX-CUT problems. Another one is to investigate if a best-improvement variant of the Expansion-Swap algorithm would perform better than the current first-improvement search approach we use. That is instead of accepting the first improving move, using the move that gives the best improvement in conflicts.

Chapter 4

Home-Delivered Meals Location-Routing Problem

The public, grant-making foundations, and corporate and private donors, fund the non-profit sector in the United States. With its funding, this sector provides valuable human services for underprivileged and needy segments of the population.

Facility location planning for non-profit delivery systems is a critical, but difficult problem. We find that there are three distinct types of human services in regard to facility location planning: home delivered services (e.g., meals on wheels and home nursing), site delivered services (e.g., literacy training and youth recreation), and residential programs (e.g., elderly nursing homes and drug treatment programs). The most difficult of the three, and one that this chapter addresses, is home-delivered services.

Home-delivered meals (HDM) provision refers to the task of delivering hot meals to the homebound infirm and elderly to ensure adequate nutrition and independent living. HDM, often referred to as "meals-on-wheels," is a service usually provided by volunteers who cook and deliver meals. This service is inexpensive in terms of direct expenditures by volunteers. However, coordination is often difficult due to variations in demands or supply of volunteers, and financial support for purchase of meal ingredients. HDM planning is a topic of interest

to public-sector planning agencies and funding bodies: many regions are experiencing a shift in demand for HDM service as a result of outmigration from central cities and aging of suburban populations. Such population shifts result in increased demand for HDM service in areas that are currently underserved.

This chapter is organized as follows. We define HDM-LRP and summarize the related literature in Section 4.1. Section 4.2 summarizes the proposed solution method. Next, computational results on benchmark problems are given in Section 4.3. In Section 4.4, we present the case study on the Allegheny County, PA data. Section 4.5 summarizes and identifies a number of research extensions.

4.1. Problem Definition and Related Literature

The general planning problem that this paper addresses is the location-routing problem for home-delivered meals, referred to as HDM-LRP. In this problem, the goal is to simultaneously choose "facility" (kitchen) locations that provide "products" (meals) to spatially dispersed customers via routes driven by multiple vehicles. Each vehicle leaves a facility, visits multiple customer locations, and returns to the same facility when customer deliveries are completed. Each customer, if served, must be served on a single route by a single kitchen. In addition, the duration of each route should not exceed the maximum allowable duration. Furthermore, the number of routes assigned to each kitchen should not exceed the maximum allowable number of routes. Finally, a kitchen, if located, cannot serve more than its maximum capacity of meals.

Traditional LRPs have minimized the sum of fixed facility location costs, vehicle operating costs, and vehicle routing costs, the latter usually approximated by total distance travelled [71, 56]. If current demand cannot be met by a realistic set of kitchens, an effectiveness measure must be also optimized. Thus, we treat HDM-LRP as, fundamentally, a multi-objective planning problem: namely, minimizing the total cost, and maximizing the

total demand served.

For the purposes of model simplicity, a number of assumptions are made. First it is assumed that the planning horizon is a single period (but see results presented in [51] for a multi-period LRP). Second, while it is impractical to identify exact locations of all customers currently receiving HDM services in typical study areas, it is assumed that exact customer locations are known in the current planning period (but see results presented in [52] for a stochastic LRP).

LRP is a generalization of two already difficult problems: the Facility Location Problem (FLP) and the Vehicle Routing Problem (VRP). Locational decisions are usually made on a strategic level, whereas routing decisions are solved at an operational level. A mathematical formulation of HDM-LRP is given in Appendix 6.2. This model, in the traditional three-index row-based style, is based on those presented in [71, 50].

Both FLP and VRP have been shown to be NP-hard [20, 48, 58], so the location routing problem is NP-hard as well. There are few exact-solution approaches to LRP. One of the earliest studies was an exact algorithm for the single facility LRP [53]. They formulated the problem as an integer-linear program and used a constraint relaxation technique to solve it. Following this study, a similar approach is used to solve both the uncapacitated and capacitated multi-facility LRP [55, 54]. A three-layer multi-commodity, capacitated distribution system is formulated as a nonlinear mixed integer program, and Benders' decomposition method is applied [11].

Due to the exponential growth in the problem size, exact methods for LRP have been limited to small and medium size instances. Indeed, LRPs have been solved to optimality for at most 80 nodes [56]. In [10], optimal solutions to LRP for 25 facilities and 150 customers, under the assumption that vehicles do not have to return to the facility after making deliveries, is reported.

The difficulty of LRP in general precludes exact techniques for realistically sized problems. However, researchers have developed a number of heuristic methods for solving LRP.

These include location-allocation first, route second [63], improvement/exchange [69], and a combination of solutions to multiple related combinatorial optimization problems [71, 40]. In recent years metaheuristics are increasingly used to solve LRPs. These include tabu search [77], nested heuristic methods with tabu search [68], two-phase tabu search [85], simulated annealing [91], and threshold accepting and simulated annealing [60]. The literature is inconclusive as to the relative efficacy of these alternative approaches. In this paper, HDM-LRP is solved using a Genetic Algorithm (GA) and this appears to be the first study that uses GA to solve LRP.

LRPs are typically solved by decomposing the larger problem into subproblems, and then solving these subproblems either sequentially or iteratively. We are not aware of any justification for this method aside from the computational inefficiency of solving LRP as a whole. We suspect that another reason might be that most of the heuristics proposed for solving LRP are local search variants. Thus, it is necessary to define a proper representation of a solution and a neighborhood structure. This is straightforward for the subproblems FLP and VRP, but finding a neat way of representing solutions and neighborhood structures for the LRP is less clear. This may be due to the fact that the decisions for the two subproblems are, by definition, unrelated; the idea of decomposition also seems quite intuitive. Moreover, a significant amount of literature already exists for the subproblems. In this paper, we use a holistic approach as opposed to the decomposition approach.

A review of the relevant literature finds scant research on the location-routing model being applied to the home-delivered meals planning problem.

A Geographic Information Systems (GIS)-based interactive heuristic is implemented in [38] to determine locations of HDM kitchens and routes by which volunteer drivers should deliver meals to cover service gaps at least cost. Model results indicate that seven additional kitchens could cover about 10 percent of the uncovered demand for HDM service in Allegheny County.

The work in [38] is extended in [47] by formulating an explicit mathematical model for

the HDM location-routing problem and by evaluating the GIS-based interactive heuristic by comparing travel times generated by the heuristic to those imputed to actual HDM data.

There has been limited research in the more general area of HDM planning. Low-technology approaches to designing HDM delivery routes for individual kitchens are presented in [7], but the authors did not address the problem of designing an entire network of HDM kitchens and delivery drivers. A spatial decision support system for HDM-LRP is developed in [88], but in using an location-allocation-first, route-second approach, this paper does not address the multi-kitchen planning problem from an LRP perspective.

4.2. Solution Strategy

As discussed in the previous section, exact methods for LRP are computationally impractical for realistic instances. There remains a need for effective computational approaches to solve large LRPs. Metaheuristics, such as tabu search, genetic algorithms, simulated annealing, neural networks, and ant colony optimization, are widely used to solve important practical combinatorial optimization problems. All of these aim to search the solution space more effectively than conventional approaches. They show great promise in solving difficult combinatorial problems such as LRP.

Among the metaheuristics mentioned, GA is an adaptive heuristic search method based on population genetics, and it borrows its vocabulary from that domain. The basic concepts of a GA were primarily developed in [42] and described later in [37]. The GA consists of a population of chromosomes; in essence, a set of character strings that are analogous to the base-4 chromosomes that we see in our own DNA that evolve over a number of generations and are subject to genetic operators at each generation. Each chromosome represents a potential solution to the problem being solved. This solution is obtained by means of a encoding/decoding mechanism. Most of the developmental work of GA theory was performed using a binary-coded GA, and, historically, is the most widely used

representation. In a binary coding each chromosome is a vector comprised of zeroes and ones, where each bit represents a gene. However different encodings are also possible.

Initially, a feasible set of chromosomes are needed, which can be generated randomly or by using a heuristic. Each chromosome has an associated *fitness value*; a set of "best fit" chromosomes from each generation survive into the next generation. The genetic operations applied to chromosomes are *crossover* and *mutation*. Typically, crossover is defined such that two individuals (the parents) combine to produce two more individuals (the children). But asexual crossover or single-child crossover are defined as well, which referred to as "reproduction" in this paper. The primary purpose of the crossover operator is to transmit genetic material from the previous generation to the subsequent generation. Mutation is a genetic operator that alters one or more gene values in a chromosome from its initial state. The mutation operator introduces a certain amount of randomness to the search. It can help identify solutions that crossover alone might not.

In this study, GAs are chosen to solve HDM-LRP for two reasons. First, as opposed to the other metaheuristics, GA uses a population of solutions in each iteration, instead of a single solution. Thus, the outcome of a GA is also a population of solutions, making GAs suitable for solving multi-objective optimization problems. Since the ideal strategy for multi-objective optimization requires multiple trade-off solutions to be found, a GA's population-approach can be suitably utilized so that it finds multiple solutions in a single simulation run. Second, a review of the relevant research does not find any models where GAs have been applied to LRP, but they have been successfully applied to its two subproblems, namely Facility Location [21], and Vehicle Routing [81].

A GA for a particular problem must have the following five components [67]:

- a genetic representation for potential solutions to the problem,
- genetic operators that alter the composition of children,
- a way to create an initial population of potential solutions,

- an evaluation function that plays the role of the environment, rating solutions in terms of their "fitness", and
- values for various parameters that the GA uses (population size, probabilities of applying genetic operators, etc.)

4.2.1 Representation and Genetic Operators

This paper proposes two different representation schemes with corresponding genetic operators for LRP, namely Facility-Route (F-R) Representation and Binary Representation.

Facility-Route (F-R) Representation

A set of possible facility locations $\{1, .., N\}$ and a set of customers $\{N + 1, .., M + N\}$ are given. If facility i is open, the corresponding entry in the chromosome is 1, otherwise the entry is 0. In addition, there is a segment in the chromosome for each open facility where the first entry is the facility number, and the following entries are the customers on a route, in order, followed by the same facility number at the end. If there is more than one route at the same facility, each is separated by the facility number. The following example demonstrates the representation of two solutions:

Example 1: Given four facility locations, {1, 2, 3, 4}, and ten customers, {5, 6, 7, 8, 9, 10, 11, 12, 13, 14}, we present Solution 1 and Solution 2 and their F-R representations. Network representation of Solution 1 and Solution 2 are given in Fig. 4.1 and Fig. 4.2, respectively.

Solution 1:	F-R representation for Solution 1:
Facility 1: Route 1: 5-6-7; Route 2: 9-11	1 0 1 0
Facility 2: Closed	1 5 6 7 1 9 11 1
Facility 3: Route 1: 8-13-12-14-10	3 8 13 12 14 10 3
Facility 4: Closed	

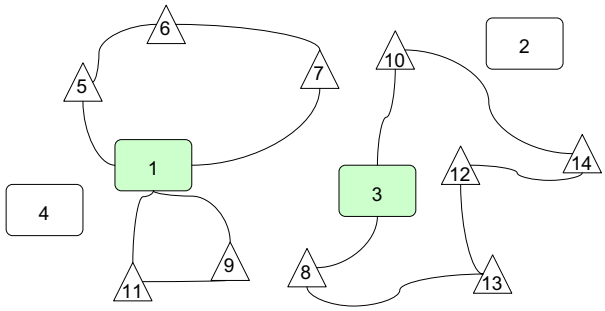

Figure 4.1: Network representation of Solution 1. Rectangles represent facilities and triangles represent customers. Shaded facilities (1 and 3) are open, others (2 and 4) are closed. There are three routes: 1-5-6-7-1, 1-9-11-1, and 3-8-13-12-14-10

Solution 2:	F-R representation for Solution 2:
Facility 1: Closed	0 1 1 1
Facility 2: Route 1: 12-7-10 Route 2: 13-14	2 12 7 10 2 13 14 2
Facility 3: Route 1: 8-9-6	3 8 9 6 3
Facility 4: Route 1: 5-11	4 5 11 4

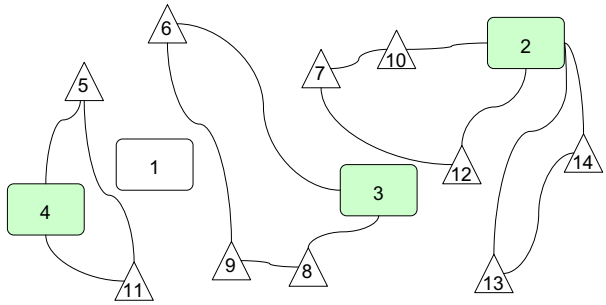

Figure 4.2: Network representation of Solution 2. Rectangles represent facilities and triangles represent customers. Shaded facilities (2,3 and 4) are open, 1 is closed. There are four routes: 2-12-7-10-2, 2-13-14-2, 3-8-9-6-3, and 4-5-11-4

F-R Genetic Operators

Facility Crossover: Two solutions are taken, and a one-point crossover is applied, where a crossover point is randomly selected and all the data beyond that point is swapped between the two parent chromosomes, using the binary genes corresponding to the facilities. Du-

plicate customers are eliminated based on a predetermined dominance rule among the two solutions for a particular offspring. This operator is not guaranteed to generate offspring that serve all the customers currently served by the parents. This operator is demonstrated in the following example:

Example 2: Given the two solutions in Example 1, and assuming that the crossover point is selected after the second facility and that the parent that defines the first facility for an offspring is the dominant one. So, for Offspring 1 (Offspring 2), all routes for facilities transmitted from Solution 1 (Solution 2) are retained and duplicate customers in facilities transmitted from Solution 2 (Solution 1) are removed. Facility Crossover is illustrated below:

Binary Parts:

Solution 1:	1 0 |1 0	Offspring 1:	1 0 1 1
Solution 2:	0 1 |1 1	Offspring 2:	0 1 1 0

Non-Binary Parts Before Repair:

Offspring 1:	1 5 6 7 1 9 11 1 3 8 ~~9~~ ~~6~~ 3 4 ~~5~~ ~~11~~ 4	Offspring 2:	2 12 7 10 2 13 14 2 3 8 ~~13~~ ~~12~~ ~~14~~ ~~10~~ 3

Non-Binary Parts After Repair:

Offspring 1:	1 5 6 7 1 9 11 1 3 8 3	Offspring 2:	2 12 7 10 2 13 14 2 3 8 3

Note that Offspring 1's facility 4 is closed during the repair process, as all the customers are removed from its routes. Note also that customers 10, 12, 13, and 14 are not served in the first offspring, and customers 5, 6, and 9 are not served in the second offspring.

It is not hard to see that if the parents are feasible, then the offspring generated with this operator are also feasible. The reason is as follows. Each facility that is open in an offspring is also open in the parent that gave the genetic material corresponding to that facility. Moreover, no extra customers are allocated to the facilities. Thus, the capacity

constraints for the facilities in the offspring are all satisfied. In addition, the length of all the routes of an offspring are no longer than the corresponding routes in the parents, since the customers are either retained in their original routes or removed as duplicates. Finally, the number of routes for each facility never increases in an offspring since there is no route exchange between the facilities and also no new route is generated. However, some existing routes from the non-dominant parent may be removed if all the customers on that route are already inherited from the dominant parent.

Facility Reproduction: This operator does not promote a mutual exchange of genetic material between two parents, but it operates in the following way: A parent receives a fragment of genetic material from another parent, and inserts it as an initial set of open facilities and routes. The fragment must consist of facilities that are not already open in the receiving solution. After insertion, a repair process checks the original facilities and routes of the receiving individual and removes all customers that also appear in the inherited material. The donor is not modified. This operator can be considered as a special case of facility crossover, except that it maintains coverage of all customers served by either of the parents. This operator is demonstrated in the following example:

Example 3: Given Solutions 1 and 2 of Example 1, and assuming that Solution 2 donates its second and fourth facilities. Facility Reproduction is illustrated below:

Binary Parts:

Solution 1: 1 0 1 0 Offspring 3: 1 1 1 1
Solution 2: 0 1 1 1

Non-Binary Parts Before Repair	Non-Binary Parts After Repair
Offspring 3: 1 ~~5~~ 6 ~~7~~ 1 9 ~~11~~ 1	Offspring 3: 1 6 1 9 1
2 12 7 10 2 13 14 2	2 12 7 10 2 13 14 2
3 8 ~~13~~ ~~12~~ ~~14~~ ~~10~~ 3	3 8 3
4 5 11 4	4 5 11 4

This operator also generates a feasible offspring from feasible parents. Every facility (and the routes of it) that is received from the donor are absolutely untouched, so they remain feasible. The facilities from the receiver might loose customers if the same customers are already inherited from the donor. But this can only decrease the amount of demand served from those facilities, shorten the routes and decrease the number of routes. Thus, feasibility is preserved.

Route Crossover: Take two solutions, and swap equal number of routes. Similar to the Facility Crossover, this operator is not guaranteed to generate offspring that serve all the customers even if the parents serve all the customers. This operator is demonstrated in the following example.

Example 4: Swap the route of facility 3 in Solution 2 with the first route of facility 1 in Solution 1, and swap the route of facility 4 in Solution 2 with the second route of facility 1 in Solution 1. Route Crossover is illustrated below:

Binary Parts:

Solution 1:	1 0 1 0	Offspring 4:	1 0 1 0
Solution 2:	0 1 1 1	Offspring 5:	0 1 1 1

Non-Binary Parts Before Repair:

Offspring 4:	1 8 9 6 1 5 11 1 3 ~~8~~ 13 12 14 10 3	Offspring 5:	2 12 ~~7~~ 10 2 13 14 2 3 5 6 7 3 4 9 11 4

Non-Binary Parts After Repair:

Offspring 4:	1 8 9 6 1 5 11 1 3 13 12 14 10 3	Offspring 5:	2 12 10 2 13 14 2 3 5 6 7 3 4 9 11 4

If the parents are feasible, then the offspring generated with this operator might only violate the capacity constraints of some facilities. However this can be avoided by verifying that a facility's capacity is not exceeded before allowing the exchange. The other constraints

are not violated. The reason is as follows. The length of all the routes of an offspring are no longer than the corresponding routes in the parents, since the customers are either retained in their original routes or removed as duplicates. The number of routes for each facility never increases in an offspring since route exchanges are only allowed between the same set of facilities.

Route Reproduction: Similar to the Facility Reproduction, this operator also does not promote a mutual exchange of genetic material between two parents. Instead, a route is taken from the donor and attached to the closest feasible facility as a new route. Next, a repair process checks the original routes of the receiving individual and removes all customers that also appear in the inherited material. The donor is not modified. This operator can be considered as a special case of route crossover, except the fact that it maintains coverage of all customers served by either of the parents. This operator is demonstrated in the following example.

Example 5: Take Solutions 1 and 2, and assume that Solution 1 donates the two routes of its first facility; assume that the first route is attached to facility 3 and second route is attached to facility 4 of Solution 2. Route Reproduction is illustrated below:

Binary Parts:

Solution 1: 1 0 1 0 Offspring 6: 0 1 1 1
Solution 2: 0 1 1 1

Non-Binary Parts Before Repair		Non-Binary Parts After Repair	
Offspring 6:	2 12 ~~7~~ 10 2 13 14 2 3 8 ~~9~~ ~~6~~ 3 5 6 7 3 4 ~~5~~ ~~11~~ 4 9 11 4	Offspring 6:	2 12 10 2 13 14 2 3 8 3 5 6 7 3 4 9 11 4

If the parents are feasible, then the offspring generated with this operator might violate the capacity constraints of some facilities. However this can be avoided by verifying that a facility's capacity is not exceeded before allowing the exchange. Similarly, the routes are

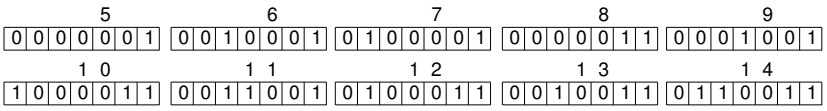

Figure 4.3: Solution 1 in binary representation. From right to left, the first P=3 bits are reserved for the facility where the customer is allocated. The next Q=1 bit is reserved for the route that the customer is on. If it is 0, the customer is on route 1; if it is 1, the customer is on route 2. Finally the last R=3 bits are reserved for the position of the customer on the route.

not allowed to be attached to facilities that already has the maximum number of routes allowed. The route length constraints are not violated since the length of all the routes of an offspring are no longer than the corresponding routes in the parents.

Mutations

Customer Swap: Select two customers and swap them. Selected customers can belong to the same route or to different routes.

Displacement: Select a sub-route and insert it into another place.

Clearly these operators might create an infeasible solution when applied to a feasible solution, so they should be applied with caution.

Binary Representation

In this representation, each customer is represented by a binary string of length $P+Q+R$. From right to left, the first P bits are reserved for the facility to which the customer is allocated. The next Q bits represent the route on which the customer is located. Finally the last R bits are reserved for the position of the customer on the route. Thus, the total length of a solution is $N(P+Q+R)$. The binary representation of Solution 1 is given in Fig. 4.3.

Binary Genetic Operators

Below are two types of crossovers and a mutation for the Binary Representation.

One Point Crossover: A crossover point is randomly selected. All data beyond that point is swapped between the two parent chromosomes. The resulting chromosomes are the children. This operator is demonstrated in the following example. Notice that the examples provided for the binary genetic operators are distinct from the LRP examples examined earlier.

Example 6: Parent 1: *11001*|*010* Offspring 7: *11001*111
Parent 2: 00100|111 Offspring 8: 00100*010*

The offsprings generated by this operator is likely to generate infeasible solutions. The reason is as follows. If a crossover point is selected such that the customers to the left of the point from parent 1 and to the right of the point from parent 2 are concentrated on some subset of facilities and/or routes, then the capacity constraint on those facilities and route length constraint on the routes might be violated. The remaining constraint, number of routes at each facility, might be violated only if the crossover point happens to be in the segment reserved for the route number of a customer. In that case, the number of routes of the facility that serves that customer might exceed the allowed limit.

Two Point Crossover: Two points are randomly selected. Everything between the two points is swapped between the parent chromosomes, rendering two child chromosomes. This operator is demonstrated in the following example.

Example 7: Parent 3: *11*|*001*|*010* Offspring 9: *11*|100|*010*
Parent 4: 00|100|111 Offspring 10: 00|*001*|111

This operator also is likely to generate infeasible offsprings. The reason is very similar to the reason explained for the one point crossover.

Mutation: This simply inverts the value of a chosen gene (0 becomes 1 and 1 becomes 0).

Comparison of F-R and Binary Representations

One of the most desirable properties of genetic operators is enabling feasible parents to produce feasible offspring. If the genetic operators do not have this property, then the infeasible solutions must be addressed by some mechanism, such as penalizing infeasibilities, removing infeasible solutions from the population.

As discussed in the corresponding sections, the Facility Crossover and Reproduction operators of F-R Representation has this desirable property. Although Route Crossover and Reproduction operators may create infeasible solutions when applied directly,any resulting infeasibilities can be checked easily. However, the crossover operators of Binary Representation do not have this desirable property. This makes F-R Representation more attractive than the Binary Representation.

F-R operators are harder to implement than the Binary operators. However, since the feasibility issue is more important when compared to the easiness of implementation, we choose the F-R representation as the basis for implementation.

4.2.2 Initial Population of Solutions

The initial set of solutions are formed by using two heuristics. The first one is a greedy heuristic. Starting from a possible facility location, the closest customer is added to the route until the time limit for that route is exceeded. Then, another route starts from the same facility. This is repeated until the maximum number of routes allowed for the facility is reached. After that, the same procedure is repeated for another possible facility location until all customers are visited. This heuristic is run as many times as needed with a different

sequence of possible facility considerations.

The second heuristic works as follows: A random set of facility locations is selected to be open. Then, customers are assigned to the nearest open facility. When all the customers are assigned, routes are formed at each open facility using the nearest neighbor approach.

4.2.3 Evaluation Function

We consider two objective functions for the HDM-LRP. The first minimizes total cost. The other maximizes the number of customers served, or equivalently minimizes the number of customers not served. For the single objective version of the problem we only consider the first objective. For the bi-objective version of the problem we take both objectives and combine them to create an evaluation function equal to the total cost divided by the number of customers served. Thus, the fitness measure is the cost per customer served. A solution with a low fitness value is considered a better solution.

One might argue that such a fitness value might result in a very low demand satisfaction in the optimum solution. In fact the best solution is not serving any customers at all, which results in a cost of zero. To prevent this, we require at least one of the facilities to be open. Since there is a fixed facility cost associated with each facility, once a facility is opened, increasing the demand served from that facility would result in a decrease in the cost per demand since the fixed cost will be shared by more customers. The optimum solution, therefore, balances the two objectives.

4.2.4 Additional Features

In order to increase the quality of the solutions we have added two more features to the GA, which we explain in this section.

Route Improvement:

After generating the initial population, a TSP heuristic is used to improve the vehicle routes of the solutions. The heuristic algorithm is due to [61] and the C code that uses this algorithm is due to [26]. This heuristic is also used to improve the routes of the populations generated later in the execution of the GA. Since applying the improvement heuristic at each generation will be time consuming, it is only applied to some of the generations. For instance one option is applying the route improvement to populations at every 50^{th} generation. The frequency of the improvement is a parameter that can be adjusted by the user.

Annealing Acceptance for Mutation Operators:

Given that the mutation operators cause random changes in the solutions, these changes might deter a good solution and create an inferior solution. This is acceptable during the initial generations but as the GA proceeds and the population evolves, mutations that increase the cost of a solution may be harmful for the population. This observation lead us to use an annealing type of approach to control the mutation operators. The increase in the cost due to a mutation is limited by a parameter called "tolerance", which decreases as the number of generations increase. If the increase in the cost exceeds the tolerance, the new solution obtained after a mutation is rejected and the original solution is restored. The tolerance parameter is decreased at each iteration using an exponentially decaying function dependent on the generation number. Thus, after several generations, the tolerance approaches zero. Very few of the solutions obtained by mutations are accepted since the probability of mutation is low and tolerance approaches zero. To balance the impact of this mechanism, the probability of executing a mutation operator is increased exponentially as the number of generations increase. So, as the number of generations increase the algorithm tries to mutate more solutions and at the same time the proportion of mutations accepted decreases as the tolerance approaches zero.

4.2.5 The Algorithm

1. Set $t = 0$

2. Form $N_{initial}$ solutions to form the set $CurrentSolutions(t)$.

3. Optimize Routes of solutions in $CurrentSolutions(t)$.

4. While the *Stopping Criteria* is not met do:

   (a) Set $t = t + 1$

   (b) Create $N_{initial}/2$ new solutions by:

      i. Randomly select two solutions from the current population of solutions and designate one of them as donor and the other as receiver.

      ii. Randomly pick one of the Reproduction (or Crossover) Operators, according to their corresponding probabilities, and apply it on the two selected solutions to create a(two) new solution(s).

      iii. If the new solution(s) created exist in $TempSolutions(t)$ or in $CurrentSolutions(t-1)$, discard it and goto Step 4(b)i. Otherwise add new the solution(s) into the set of $TempSolutions(t)$

   (c) For every solution S in $TempSolutions(t)$ and $CurrentSolutions(t-1)$ do:

      i. Apply Displacement(Customer Swap) to S with $DisplacementProbability$ ($CustomerSwapProbability$) and create $S^{'}$. If Cost($S^{'}$) - Cost(S) $\leq Tolerance$, replace S with $S^{'}$. Otherwise discard $S^{'}$ and restore S.

   (d) Replace the inferior solutions in $CurrentSolutions(t-1)$ with the superior solutions in $TempSolutions(t)$ and create $CurrentSolutions(t)$

   (e) If $mod(t, RouteOptFreq) = 0$ Optimize Routes of solutions in $CurrentSolutions(t)$.

4.3. Experimental Results

In this section, we first present implementation details for the new GA. Then, we describe the test data sets. Last, we present and discuss computational results.

4.3.1 Implementation Details

The implementation and experimentation is divided into two phases. The first phase is the "LRP Phase", where the aim is to test the proposed GA on some instances available in the literature and compare the results with previous works results. After comparing the results obtained by the GA with the previous works' results, the next step is the "HDM-LRP Phase", where there do not exist any benchmark instances and there exist only one relevant previous study. The experimental results given here are the results of the LRP Phase. The results for the HDM-LRP Phase are given in Section 4.4.2.

In order to test our algorithm on benchmark instances and compare our results with previous results, we modified some of the characteristics of HDM-LRP. The basic characteristics of the problem considered is as follows: The objective is only to minimize the total cost. Full demand satisfaction is imposed as a constraint. We impose a vehicle capacity constraint that restricts the length of each route instead of the time limit constraint of HDM-LRP. Moreover, as opposed to HDM-LRP, there is no limit on the number of the vehicles available.

The problems considered have a single objective, and full demand satisfaction is a constraint. Since the crossover operators described in earlier sections do not guarantee to generate offspring that serve all customers, they are excluded from the solution process. Only the reproduction operators are used to solve the LRP instances.

The parameters of the GA have been set empirically: We used an initial population size $N_{initial} = 500$. At each generation we created 250 new solutions by applying Reproduction Operators to the current solutions. At iteration t, a new solution is created by using Facility

Reproduction with probability $0.5/t^{0.25}$ and by using Route Reproduction with probability $1 - 0.5/t^{0.25}$. Displacement is applied to a solution with $DisplacementProbability = 0.1 * t^{0.33}$ and Customer Swap is applied with $CustomerSwapPropability = 0.2 * t^{0.33}$. A mutation is accepted if the change in the cost of the solution do not exceed $Tolerance$, which is initially set equal to 2% of the cost of best solution in the initial population, $CurrentSolutions(0)$. For $t > 0$, $Tolerance = Tolerance/t^{0.5}$. The algorithm is terminated if the best solution cost is not improved in 50 consecutive generations.

4.3.2 Instance Description

The first set of instances that serve as a test bed for the GA is tested are due to [85]. These instances are randomly generated with different characteristics such as problem size, spatial distribution of customers and route structure. The set is composed of instances with number of customers equal to 100, 150 or 200 and number of potential facilities equal to 10 or 20.

The second set of instances are obtained from the compilation in [6]. These instances are derived from various articles in the literature. We tested the GA on the larger instances, where the number of potential facilities is at least 8 and the number of customers is at least 75.

4.3.3 Summary of Results

In order to demonstrate the difficulty of the problem, we first formulated the LRP as a mixed-integer-program (MIP) and run it on OPL studio 3.7, where CPLEX MIP solver is used to solve the problem. We compared the results obtained by the MIP with the results of the GA. We then tested the GA on benchmark instances described in Section 4.3.2 and compared the results with the results obtained by other algorithms. We performed our computational tests an on a PC with 1.6 GHz Pentium M Processor and 1GB of RAM.

Table 4.1: Results of the MIP Model on small LRP instances. The columns in the table consist of the name of the instance, number of customers (n) and potential facilities (m), number of variables (Var.) and the constraints (Const.) of the MIP, cost of the best feasible solution (Cost) and best lower bound (LB) found by the MIP, time in seconds for MIP (Time), cost of the solution found by the GA (Cost), and time in seconds for the GA (Time)

			MIP					**GA**	
Problem	**n**	**m**	**Var.**	**Const.**	**Cost**	**LB**	**Time**	**Cost**	**Time**
Christofides69-75x10	75	10	86786	68850	–	485.6	44234.5	851.7	59.6
Christofides69-45x6	45	6	23461	18844	964.5	353.7	26378.7	600.5	28.9
Christofides69-30x4	30	4	6971	5691	569.3	296.3	17944.2	459.8	1.6
Christofides69-15x4	15	4	2186	1536	292.9	191.6	13527.9	282.6	0.2
Christofides69-15x3	15	3	991	772	282.2	209.9	12187.5	287.9	0.1
Christofides69-10x3	10	3	351	258	219.5	219.5	109.4	222.0	0.1

Comparison of GA with the Mixed Integer Program

The instance that we solved is one of the smallest instances this paper considers and is due to [19] with 75 customers and 10 possible facility locations. This instance is also one of the problems solved by the GA, where the related results are demonstrated in Table 4.1. The MIP formulation has 68850 constraints and 86786 variables. The computer ran out of memory after 11 hours of execution without finding a feasible solution. The lower bound it found (485.7) is far less than the current known best lower bound (744.7).

We then tested the performance of MIP on the subproblems of this instance. Starting from a subproblem with 10 customers and 3 potential facilities, we increased the number of customers and facilities to create new instances. The results are demonstrated in Table 4.1. Except for the smallest instance, MIP could not obtain an optimal solution before exhausting the computer's memory. The gap between the best feasible solution found and the best lower bound increases as the instance size increases. The GA found better solutions than the MIP for all the subproblems except the smallest two. In those two smallest subproblems, the difference between the costs is very small. For the smallest size subproblem, the solution found by the GA has a cost almost equal to the optimal cost. Solution times for

the GA are multiple orders of magnitude better than those of the MIP.

Comparison of the GA with Other Algorithms

Table 4.2 presents the results of the GA and two other heuristics, SAV1 and Tabu Search, on the first problem set. The results for the SAV1 and Tabu Search (TS) are presented in [85]. In that study, they implemented the SAV1 heuristic, which was initially proposed in [79], to compare their Two-Phase TS Algorithm. The SAV1 heuristic, also known as savings heuristic, assumes all potential facilities to be open initially, and uses approximate routing costs for open facilities to determine the facility to be closed. The TS algorithm, on the other hand, coordinates two tabu search mechanisms. The first seeks a good facility configuration, the second seeks good routes that corresponds to this configuration.

The GA found better solutions than the SAV1 for all but one of the instances. The average improvement of GA over the SAV1 is 3.2%. The GA obtained better solutions than the Tabu Search for 28 instances and obtained a solution with the same cost for one instance and worse solutions for the remaining 7 instances. The average improvement of GA over the Tabu Search is 0.9%.

The GA is a slower algorithm than the SAV1 and the Tabu Search Algorithms. This is expected because the GA generates a population of solutions. In addition we note that no particular effort has been made to speed-up the GA. Nevertheless, the range of solution times for GA have a mean of 9 minutes.

Table 4.3 presents the results of the GA on the second problem set. The GA obtained solutions that have lower costs than the current best upper bounds for all six instances. The average improvement of GA over the upper bounds is 3.5%.

Table 4.2: Results of the GA on the first set. The columns in the table consist of the name of the instance, number of customers (n), number of potential facilities (m), cost found and time in seconds for SAV1, Tabu Search (TS) and Genetic Algorithm (GA), % improvement obtained by the GA over SAV1 and TS.

Problem	**n**	**m**	**SAV1 cost**	**SAV1 time**	**TS cost**	**TS time**	**GA cost**	**GA time**	**% Impr Over SAV1**	**% Impr. Over TS**
P111112	100	10	1596.0	1.0	1556.6	5.0	1498.7	179.8	6.1	3.7
P111212	100	10	1457.0	1.0	1443.4	3.0	1428.8	167.3	1.9	1.0
P112112	100	10	1245.8	1.0	1231.1	4.0	1206.5	62.6	3.2	2.0
P112212	100	10	830.5	1.0	825.1	3.0	787.3	44.4	5.2	4.6
P113112	100	10	1326.1	1.0	1317.0	3.0	1278.4	139.0	3.6	2.9
P113212	100	10	927.2	1.0	920.8	4.0	917.2	368.4	1.1	0.4
P111122	100	20	1557.1	3.0	1531.9	3.0	1480.5	83.9	4.9	3.4
P111222	100	20	1555.9	3.0	1511.4	4.0	1481.3	200.8	4.8	2.0
P112122	100	20	1140.1	3.0	1132.0	2.0	1127.1	351.0	1.1	0.4
P112222	100	20	744.7	3.0	740.6	3.0	781.3	189.9	-4.9	-5.5
P113122	100	20	1316.5	3.0	1274.5	4.0	1259.1	248.2	4.4	1.2
P113222	100	20	1109.3	3.0	1042.2	3.0	1038.1	128.2	6.4	0.4
P131112	150	10	2066.2	3.0	2001.0	12.0	1986.2	429.7	3.9	0.7
P131212	150	10	2113.6	2.0	2022.1	14.0	2037.7	435.0	3.6	-0.8
P132112	150	10	1653.4	4.0	1555.8	9.0	1479.4	865.1	10.5	4.9
P132212	150	10	1238.0	2.0	1231.3	9.0	1230.7	1037.6	0.6	0.1
P133112	150	10	1848.4	3.0	1762.5	9.0	1755.3	618.5	5.0	0.4
P133212	150	10	1247.3	3.0	1264.6	10.0	1243.4	263.5	0.3	1.7
P131122	150	20	1977.4	8.0	1892.8	12.0	1887.5	293.3	4.5	0.3
P131222	150	20	1931.6	8.0	1855.0	13.0	1856.2	145.5	3.9	-0.1
P132122	150	20	1554.5	9.0	1478.8	12.0	1498.8	380.3	3.6	-1.4
P132222	150	20	953.3	9.0	948.3	9.0	945.5	648.9	0.8	0.3
P133122	150	20	1496.4	7.0	1488.3	9.0	1442.0	366.6	3.6	3.1
P133222	150	20	1192.6	8.0	1182.3	9.0	1182.4	652.9	0.9	0.0
P121112	200	10	2463.2	5.0	2379.5	22.0	2349.9	162.9	4.6	1.2
P121212	200	10	2395.9	4.0	2288.2	23.0	2292.8	425.1	4.3	-0.2
P122112	200	10	2203.4	9.0	2158.6	20.0	2130.6	1092.0	3.3	1.3
P122212	200	10	1560.8	8.0	1549.8	18.0	1504.9	1508.6	3.6	2.9
P123112	200	10	2312.8	5.0	2056.1	23.0	2010.5	2370.3	13.1	2.2
P123212	200	10	1849.8	6.0	1877.3	20.0	1833.1	1640.3	0.9	2.4
P121122	200	20	2289.4	15.0	2211.7	22.0	2254.6	373.9	1.5	-1.9
P121222	200	20	2417.3	19.0	2355.8	26.0	2320.0	448.3	4.0	1.5
P122122	200	20	1805.9	20.0	1787.0	18.0	1775.0	1033.6	1.7	0.7
P122222	200	20	1122.9	19.0	1113.0	18.0	1110.8	980.1	1.1	0.2
P123122	200	20	2046.5	15.0	2002.4	20.0	1999.3	783.3	2.3	0.2
P123222	200	20	1423.8	15.0	1414.8	17.0	1469.0	184.0	-3.2	-3.8
Average			1610.3	6.4	1566.8	11.5	1552.2	536.2	3.2	0.9

Table 4.3: Results of the GA on the second set. The columns in the table consist of the name of the instance, number of customers(n), number of potential facilities(m), currently known lower(LB) and upper bounds(UB), cost found by the GA,% improvement obtained by the GA over the UB, and the time in seconds(time) for the GA.

Problem	**n**	**m**	**vc**	LB	UB	**GA**	**% Imp.**	**time**
Christofides69-75x10	75	10	140	744.7	886.3	851.7	3.9	59.6
Daskin95-88x8	88	8	9000000	356.4	384.9	375.7	2.4	64.3
Christofides69-100x10	100	10	200	788.6	889.4	868.2	2.4	69.1
Or76-117x14	117	14	150	12048.4	12474.2	12071.9	3.2	451.0
Min92-134x8	134	8	850	–	6238.0	6040.5	3.2	34.0
Daskin95-150x10	150	10	8000000	43406.0	46642.7	44011.3	5.6	129.6
Average							3.5	134.6

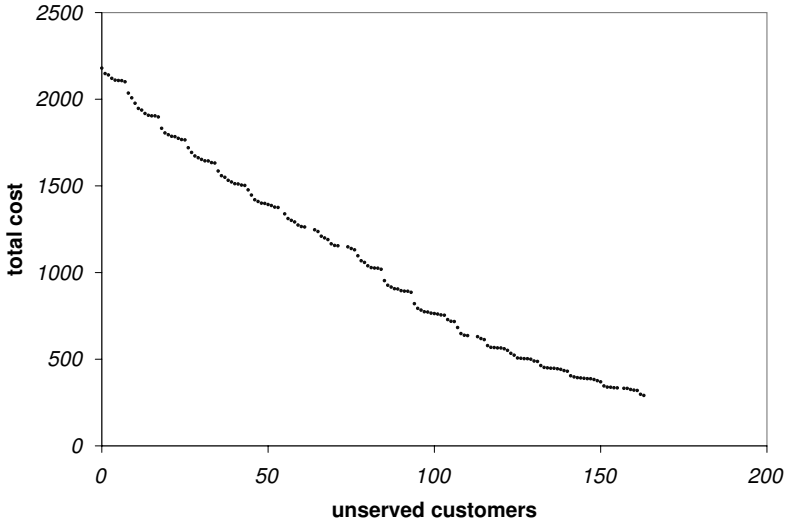

Figure 4.4: The trade-off curve for instance P122112, with n=200 and m=10.

The Bi-Objective LRP

Although the focus of the experimental results in this section is for the single objective version of the LRP, we briefly discuss the bi-objective version of the problem as well. As stated before, the GA has the advantage of generating a population solutions after a single run. Thus, for a multi-objective problem, a decision maker may choose from a set of solutions with different objective values. This is especially important for public sector problems, such

as the HDM problem considered in this study, where a decision maker has to consider budget restrictions as well as customer service levels. In order to demonstrate this advantage of the GA, we run it using the Facility Crossover operator along with the Reproduction and Mutation operators, on one instance from the first set of instances, namely the P122112. The tradeoff curve for this instance is illustrated in Fig. 4.4. To obtain this curve, the GA kept only the non-dominated solutions at each generation. A solution S_1 is called *dominated* if there exists a solution S_2 that has a lower cost than or serves more customers than S_1. This trade-off curve illustrates the following points:

- the overall shape of the curve resembles a hyperbola, where the rate of decrease in total cost decreases as the number of unserved customers increase,
- the graph itself consists of several small curves that have a similar shape with the graph. Moreover, there is significant cost change at the end of these segments, which is most likely due to a difference in the number of open facilities at the two neighboring end-of-segment solutions.
- the total cost for the solution that serves all customers is 2179.54. The costs found by the SAV1, TS and the GA by solving the single objective version of LRP are 2203.4, 2158.6, and 2130.6, respectively. So the solution found by the bi-objective GA is better than the cost of SAV1 and has a cost close to the costs of the TS and single-objective GA.

4.4. Case Study: HDM-LRP in Allegheny County, PA

Allegheny County, Pennsylvania has a total population of 1.29 million, and 0.23 million of these are 64 or older. The county has 63 HDM distribution centers (kitchens), which are primarily pre-existing facilities such as churches and schools, delivering daily hot meals to slightly over 4,000 home-bound elders. It is estimated that a little over 80 percent of the demand is met by current facilities, with a net benefit, in avoiding costs of residential

nursing homes, to be in excess of $100 million annually [38]. The typical facility has four to six routes driven by volunteers with a dozen stops and one or two clients per stop. The major fixed costs of a new facility are organizational: gaining commitment from a facility, securing funding for operations, recruiting and organizing networks of drivers, and designing routes. Federal subsidies have the provision that recipients be home-bound persons aged 64 or older and that delivered meals be at least 140 F. With insulated carriers, the temperature requirement translates into a 45 minute time limit for delivery of the last meal on a driver's route.

4.4.1 Data

Implementing HDM-LRP for policy analysis requires a variety of data elements. However, for many volunteer public-sector services, reliable data for planning purposes are difficult to acquire. We present approximate methods to estimate the following data: customer demands, kitchen locations, point-to-point distances, vehicle travel costs, and kitchen fixed costs for our study area of Allegheny County, PA. Note initially that our focus is limited to a portion of Allegheny County consisting of the land area east of the confluence of the Ohio, Allegheny and Monongahela Rivers, and bounded to the north and south, respectively, by the latter two rivers. Note that our dollar-valued costs are rarely paid in actual transactions between HDM operators and agencies that support them. Thus, these costs should be interpreted as social "opportunity costs" that represent the value of the labor or materials in their best alternative use. In Allegheny County, we are able to acquire accurate records of customer locations for only 10% of the 63 known home-delivered meals kitchens. The agency that manages HDM services, the Area Agency on Aging, has no methodology for estimating the larger population that contains HDM-eligible clients. Finally, this population, even if recorded, would change over time based on secular demographic trends. For this reason, we use a GIS-based demand forecasting method [38]. This method uses the following data: forecasts of the elderly population at the block level, estimates of the percentage of elderly

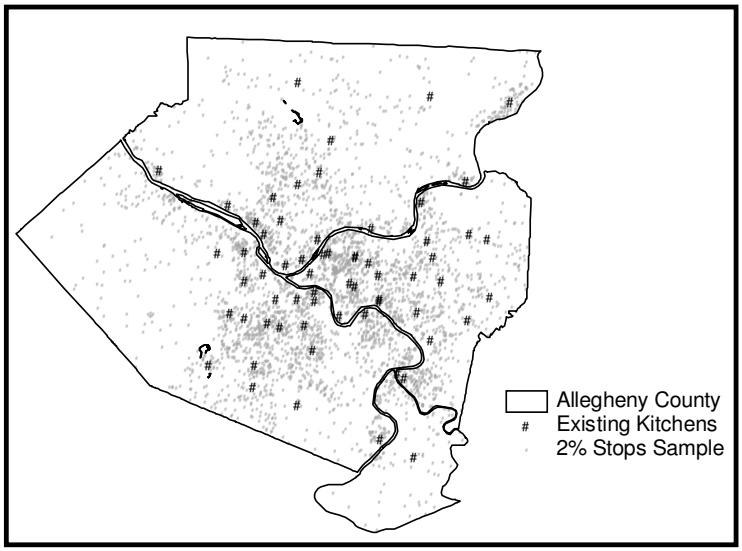

Figure 4.5: Client and Kitchen Locations for Allegheny County, PA

who use HDM, and estimates of the number of clients per HDM customer location. With these data, we sample a number of points within each Census block groups consistent with the data elements listed above, and reverse-geocode these sample points to a street address. The resulting demand sample for Allegheny County consists of 4379 stops and 5069 total clients. The clients are generated using a conservative demand estimate of 2% of the elderly population.

The demand sample for the Allegheny County subset examined in this case study consists of 1003 stops and 1135 total clients. HDM kitchen locations and service capacities were provided to us by Area Agency on Aging. There are 25 kitchens within the Allegheny County subset examined by this paper. Figure 4.6 illustrates the locations of HDM clients generated for the Allegheny County subset examined in this paper as well as the existing kitchen locations.

Travel costs between customer locations are based on travel distances. We compute distances between simulated potential customers and existing and potential HDM kitchens

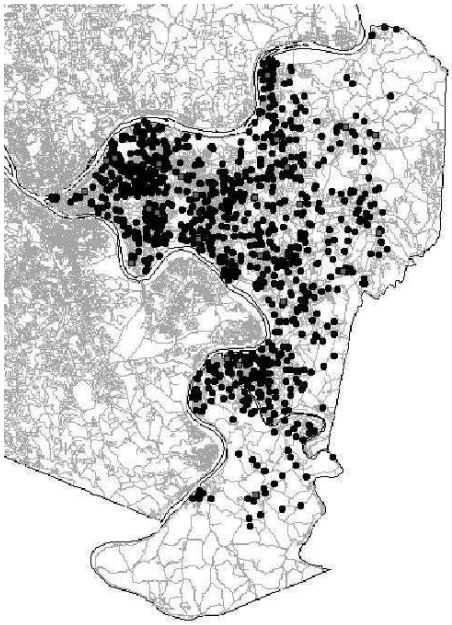

Figure 4.6: Client and Kitchen Locations for the portion of Allegheny County, PA studied in the case study.

using a street network for Allegheny County and a customized script created in the ArcView language Avenue. To convert these distances to dollars, we assume an average travel speed of 25 miles per hour and a standard mileage reimbursement rate for vehicle travel accounting for fuel costs and wear-and-tear provided by the U.S. Internal Revenue Service for September 2005 through December 2005 of \$0.485/mile. We also assume a meal preparation cost of \$3 per delivered meal. Fixed costs for locating an HDM kitchen are assumed to be classified as facility acquisition, services planning or facility maintenance costs. Facility acquisition costs are based on the time, in hours, spent to gain commitment from a facility such as a school or church to serve as an HDM kitchen, and the time, in hours, spent to secure funding for operations. We assume that each facility acquisition cost component requires 20 hours and 10 hours, respectively, and, like all labor inputs identified here, are valued at a wage rate of \$8 per hour. Service planning costs are based on the time required to recruit and organize a network of drivers and to design routes. We assume that each service

planning cost component requires 8 hours per driver and 5 hours, respectively.

We assume that these costs are $250 per month and $320 per month, respectively. For a 5-year planning horizon, the present values of these costs are $13,743 and $17,591, respectively. Finally, we assume that kitchens can be acquired and designed to serve small, medium and large client populations. Due to a lack of management expertise and resources at typical HDM kitchens and funding agencies, we assume that HDM kitchens face decreasing scale economies, i.e. that, as compared with a medium-size facility serving 30 - 60 meals daily, a smaller facility has total fixed costs that are 80% of the medium facility's costs while a larger facility has total fixed costs that are 50% higher.

4.4.2 Computational Results

After running the GA on the sample data, we realized the cost is dominated by the total meal cost. 75% of the cost is the meal cost, 12% is the fixed facility costs and the remaining 13% is the variable costs.

The estimated costs for fixed facility and variable routing costs are much lower than the private sector costs, as they are basically opportunity costs. Thus, the fact that meal costs dominates the total cost should not imply that they are unimportant. Even though the meal costs is the major component, our understanding is that the reason that all the customers are not being able to be served is the lack of proper allocation of resources to demands. The budget is not restricting the Area Agency of Aging to serve all the customers. So, if we were to improve on facility and routing costs, more customers can be served.

Figure 4.7 shows the trade off between the total cost and the total number of customers served. The relationship is almost linear because of the dominance of meal costs within the total cost.

Figure 4.8 shows the tradeoff between the total cost without the meal cost and the total number of customers served. The rate of increase in the cost increases as the number of customers served increases.

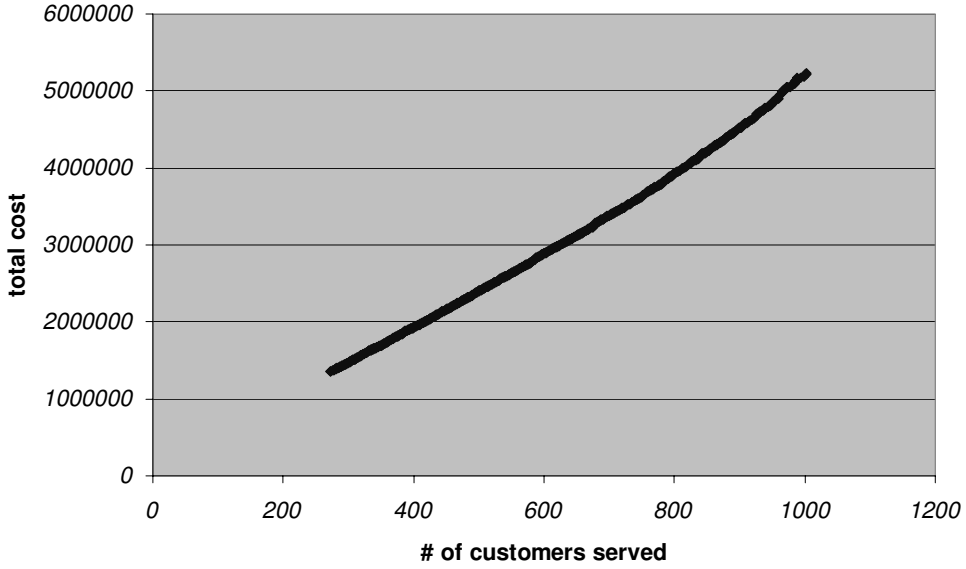

Figure 4.7: Tradeoff curve for HDM-LRP on the Allegheny County Data: Total Cost vs. Total Number of Customers Served.

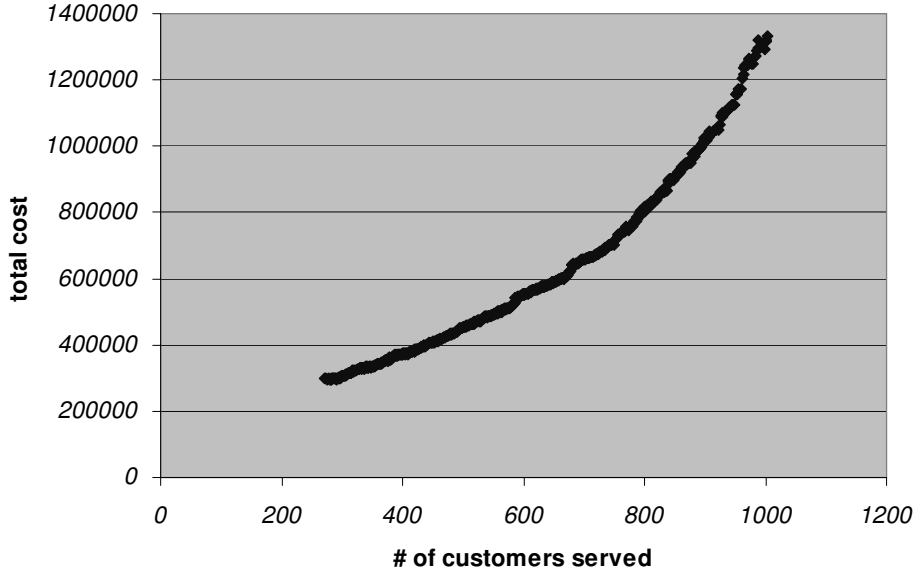

Figure 4.8: Tradeoff curve for HDM-LRP on the Allegheny County Data: Total Cost(excluding the meal costs) vs. Total Number of Customers Served.

Comparison with the status quo

A natural comparison that is needed to be made is the comparison of the new solution with the current practice. However this is not possible as we do not have actual routes for the kitchens. To be able to have a rough estimate we used a nearest neighbor heuristic to generate routes and run a TSP heuristic to improve these routes. This solution has a cost $1,625,064. Since there is no computerized planning is made for the used routes and kitchens, we can assume that the actual costs should be no better than what is obtained by this estimate. The new solution has a cost of $1,333,235. The reduction in cost is almost $300,000, which corresponds to an 18% decrease.

These results imply that closing unused kitchens and opening new ones (or increasing capacities of existing ones) closer to suburbs will be cost efficient at the same level of customer service, namely 100%.

Spatial Properties

By looking at the depictions of the routes for the two extreme point solutions on the tradeoff curve, we can note some spatial properties. The cost-minimizing solution, which has the lowest customer service has routes that are more compact and serving more customers. Namely, the average number of customers per route is 21. On the other hand this figure is 17 for the service maximizing solution, which has the highest cost with full demand satisfaction.

4.5. Summary and Future Research

In this paper we have attempted to extend the research literature on optimization-based planning models for home-delivered meals (HDM) service provision. We have motivated the problem by noting that in Allegheny County, PA, there is significant unmet demand for HDM service.

This study proposes a Genetic Algorithm for the Home-Delivered Meals Location-Routing Problem and tests this algorithm on benchmark LRP instances. We also used the algorithm in a case study on the large HDM data for Allegheny County, PA.

We tested our algorithm on two sets of single objective LRP problems and presented the computational results. We have shown that the GA performs better than the SAV1 and Tabu Search methods for most of the instances in the first set and it finds solution that have a cost lower than the current best upper bound for the instance in the second set. We also demonstrated a trade-off curve for one instance for the bi-objective version of the problem.

Trade-off curves are very useful for decision makers who has to plan for HDM or similar services that can be formulated as a LRP. Such graphs help them to see the trade-off between the level of demand satisfaction and the total cost and gives them the opportunity to pick a solution they think more appropriate. We generated two such graphs in the case study where we solved the HDM-LRP problem defined on a portion of Allegheny County.

There are a number of features of the most general description of the HDM planning problem that have not yet been incorporated into HDM-LRP but, if so addressed, could increase the utility of the model for planners. Given that demands for HDM services vary over space and over time, it is appropriate to examine extensions to HDM-LRP that address multi-period planning strategies with explicit forecast horizons.

Sensitivity analysis will be a natural addition to the current work to see how the results change when of the components of the model, such as the cost structure changes. This work can also be extended by adding *equity* or *fairness* as a third objective into the problem.

Chapter 5

Optimization of Customer and Supplier Logistics at a Large Automotive Parts Manufacturer

In the automotive industry, traditionally each company in the supply chain plans and pays for the shipments from its suppliers. For instance, automotive manufacturers pick up and pay for the cost of shipments from the first tier suppliers; first tier suppliers do the same for the supplies they get from second tier suppliers and so on and so forth. The nature of these supplies is interesting in the sense that while there is material flow from lower tier to higher tier suppliers, there is also significant flow of empty containers (return dunnage) in the other direction as shown in Figure 5.1.

There are two main types of transportation offered by carriers: truck-load (TL) and less than truck-load (LTL). TL carriers are dedicated trucks carrying a single customer's load directly from origin to a destination potentially stopping along the way to fill the truck. The final destination can be different than the origin, which makes the route a one-way route; or it can be the same, which then makes the route a round-trip route (also referred

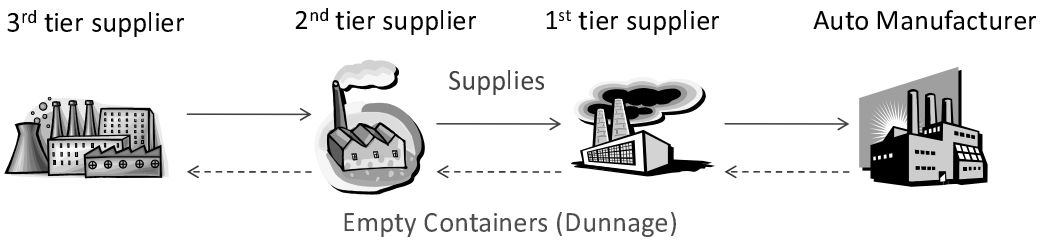

Figure 5.1: Logistics in the auto industry involve transfer of materials and dunnage in opposite directions.

to as a "*milkrun*"). In either case, there may be intermediate stops between the origin and destination. LTL carriers, on the other hand, provide a shared service on the same truck and use an airline-type hub-and-spoke system with shipments. An LTL carrier usually assumes the responsibility for routing each shipment from the origin to the destination.

The cost structures of these two types of carriers are very different and it may be more advantageous to use one or the other depending on the destination(s), the amount, and the density of the shipment. Since the unit cost (i.e. cost per pound per mile) of round-trip TL carriers is significantly less than the unit cost of one way TL carriers or LTL carriers, companies try to utilize round trip TL trucks that carry supplies from their suppliers and carry empty containers in the rest of the round trip.

Notably, the amount of space occupied by empty containers is significantly less than the space they occupy when they are full. This is due to the fact that some of the containers are designed to collapse when they are empty and thus occupy less volume. Moreover, not all shipments are made with returnable containers but rather with materials such as cardboard, which are disposed after they are used. Because of this, the ratio of the volume of returned empty containers to the volume of products shipped range from 0 to 1. Furthermore, the weight of the empty containers is also significantly less than their weight when they are full. Interestingly, the relatively low weight of the containers also allows higher stacking of containers, which may not be possible when the containers are full due to total weight

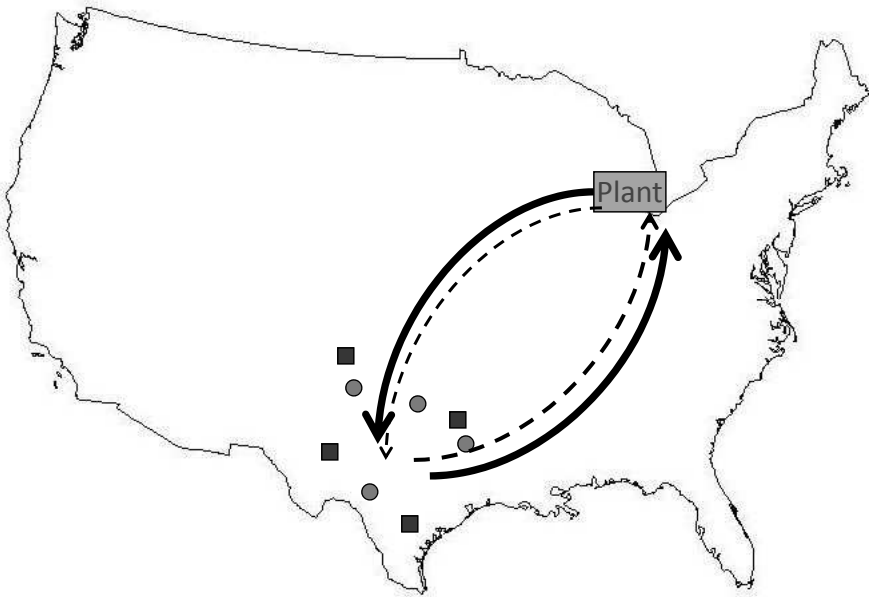

Figure 5.2: Uneven flows from and to the suppliers and customers, represented by squares and circles, respectively, in both directions going over long distances can be matched for better coordinated utilization.

restrictions on trucks. Thus, the amount of reverse flow of empty containers do not match the amount of product flow. This leaves plenty of unused capacity on the return segment of a round trip route. With this observation, we realize the following opportunity for a first tier supplier.

5.1. Matching Opposite Flows: A Unique Opportunity

A company that has customers and suppliers located in the same region, which is far away from the manufacturing plant, can combine the flow of inbound supplies and empty supplier containers with the flow of outbound finished products and empty customer containers. This is illustrated in Figure 5.2, where customers, represented by circles, and suppliers, represented by squares, are located in Texas and they are far away from the manufacturing plant, represented by a rectangle, located in Michigan. The product flows are represented

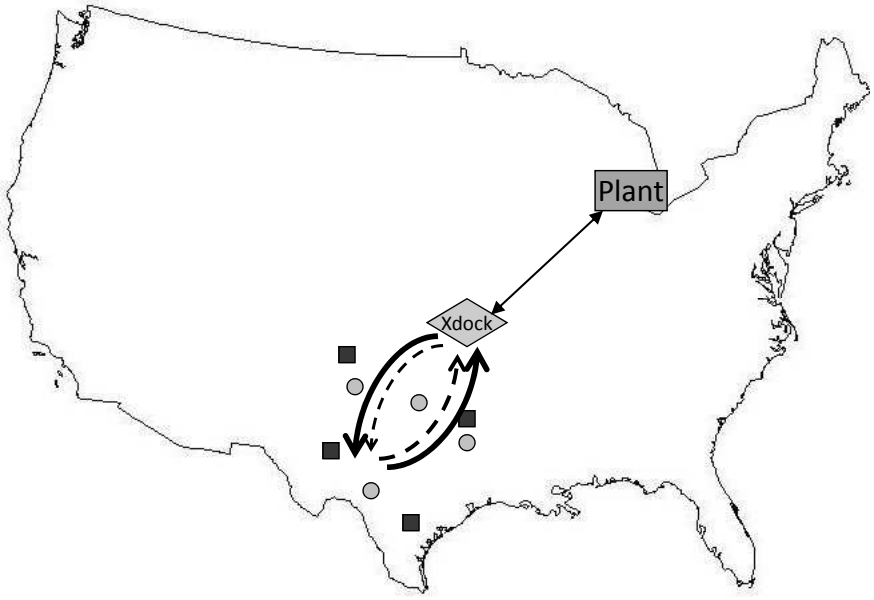

Figure 5.3: Using a crossdock close to the customers and suppliers to better match the uneven flows.

by solid lines whereas the empty container flows are represented with dashed lines, which are drawn thinner than the solid lines to represent the imbalance mentioned earlier.

This opportunity is not possible for automotive manufacturers since trailers that can carry automobiles are very different than the trailers that carry their supplies stored in containers. This is also a less potential for a second tier supplier who may not have as much volume in both directions. But a first tier supplier can utilize this opportunity by taking over the delivery of its products to its automotive manufacturer customers.

5.1.1 Crossdocking

A crossdock is a facility where materials are unloaded from incoming trucks and loaded into outbound trucks with little or no storage in between. Such a facility located close to the customers and suppliers can also be used as a consolidation point to be able to match the two opposite flows better, as depicted in Figure 5.3. Locational decisions, such as locating

a crossdock, are usually made on a strategic level as they have long term effects. In the case of using a crossdock facility, one alternative is owning the facility by making an investment. Another alternative is renting the facility, which may require less capital investment than the first one, but still require a commitment for a period of time. A third alternative is using an existing facility operated by a 3rd Party Logistics Firm (3PL) and paying a fee on a per container basis. This last alternative might be desirable if the total volume that will be going through the crossdock is not very large and if the company does not desire to make an investment or a long term rental contract.

5.2. Motivation

The Auto Parts Manufacturer, which we do this project for and to whom we refer as the Client in this paper due to confidentiality reasons, applies lean manufacturing practices in its production system for the last few years. Lean manufacturing is the production of goods using less of everything compared to traditional mass production. Lean manufacturing is a generic process management philosophy inspired by the Toyota Production System(TPS) [87] and implemented in the Client's plant. Our study was motivated by the Client's desire to improve its logistics operations by removing some of the inefficiencies and waste in the supply chain as explained in previous sections. In addition to direct transportation cost savings, attaining additional benefits, such as reducing inventories, is also desired since one of the principals of lean is evening out the production flow by reducing batch sizes and increasing delivery frequencies internally and, if possible, externally.

5.3. Problem Description

In this project, we look into selecting a subset of customers and suppliers of the Client and combining their shipments on the same routes. We also consider the use of a crossdock located close to the customer and supplier base to use as a consolidation center. We in-

vestigate whether such changes bring net savings to the system, where the system consists of a single plant of an automotive parts manufacturer (the Client), customers (automotive manufacturers) and suppliers (second tier suppliers). Every node in the system may have both pick ups and deliveries: Each customer has a demand from the Client and it may also have a supply if the containers used are returnable. Similarly each supplier has a supply to the Client and it may also have a demand if the containers used are returnable. The supply and demand at each node has to be served simultaneously by the same vehicle.

In addition to the vehicle routing aspect of the problem, we also have to decide on whether to use a crossdock as a transshipment node. The crossdock cost is a variable cost that is charged on a per container basis, so there is no fixed rental or investment cost. The location of the potential crossdock is predetermined.

We look at the problem from a supply chain coordination perspective. In the new system, the Client will make all the transportation arrangements including shipments for its customers. To implement this change, materials supplied to customers will have to be re-priced to factor in shipping costs to the Client, which is currently incurred by the customers.

5.4. Literature Review

At the core of this problem is vehicle routing, which is defined as the process of selecting paths in a transportation network along which to send physical traffic. The goal is to provide services to spatially dispersed customers via delivery routes that have one or more customers on them.

There are variants of the Vehicle Routing Problem(VRP) based on different aspects, such as the number of depots (one/multiple), cost structure(fixed/variable/both), time to service a particular node(specified/time windows/unspecified), size of fleet(one/multiple), type of fleet(identical/non-identical), nature of demands(deterministic/stochastic), existence of

pickups along with demands at each node and vehicle capacity(imposed/non-imposed).

The basic version of the problem is defined as follows: There is only one depot where the vehicles begin their routes and we are given its location. We also know the locations and demands of customers. We have a number of available vehicles that have a finite capacity. This can be a physical capacity, such as the weight or volume, or it can be a restriction imposed due to the nature of the system, such as the total distance or total time of the route. The goal is to find a set of routes that minimize the total distance the vehicles travel.

VRP is a very difficult problem to solve. In fact, even with the very recent developments, the size of the problems that can be solved by exact solution approaches are in the order of a hundred customers. Among the most effective exact methods are the branch and cut method in [62], the branch & cut & price method in [30] and the set partitioning approach with cuts in [4].

Due to the difficulty of VRP, most of the techniques developed are heuristics, including local search methods, population based algorithms and learning mechanisms. These methods generate quality solutions very fast but do not guarantee optimality. The best performers, in terms of accuracy and computing time seems to be the hybrid methods presented in [66], [80] and [74] that combine population search and local search.

In the version of the VRP that we are considering in this paper, every node may have both pick ups and deliveries. The general class of such problems is referred to as Pickup and Delivery Problems (PDPs) in the literature. To the best of our knowledge and based on the examination of a very recent survey on PDPs [9], the version of the VRP considered in this paper has not been studied before. The closest versions are the *VRP with Simultaneous Pickups and Deliveries*(VRPSPD), and *VRP with Pickups, Deliveries and Transshipments*(VRPPDT), as defined in [9]. If one has to decide on where and how many facilities to locate, along with the vehicle routing decisions of how to serve customers, and if locating a facility require a fixed cost, then the problem is referred to as the *Location Routing Problem*, which is a more difficult problem than the VRP and its variants.

5.5. Modeling Assumptions

Cost Linearity: In order to compare the cost of the new system with the existing one, we need to know the cost of customers' pickups from the Client. To estimate those costs, we assume that the cost of a truck is linear in the percentage fill rate: Once the shipment from the Client to a customer is removed from customer's current route, the customer can, and will, fill in the free space with other suppliers' material, which are already on the existing route of the customer, by decreasing the frequency of that route.

This assumption can be justified by noticing the fact that customers will re-optimize their routes after removing the Client's plant from their list of stops. We can not know the effect of not having the Client on any of their routes as this requires the knowledge of all of their suppliers and their pickup schedules from them. This effect can be positive or negative, so our best guess is that, customer's networks are large enough that removing the Client from their system will have a neutral effect.

A Closed System for Empty Containers: We assume that the returnable container flow is a closed loop and whatever returnable container is sent (received) to (from) customers (suppliers) comes (send) back to the Client (suppliers).

5.6. Problem Formulation

We developed a network flow model combined with vehicle routing from the crossdock and the plant. The formal formulation is given in the Appendix. In this model, we allow two different means of transportation for each customer and supplier. One is being on a milkrun starting and ending either at the plant or the crossdock. In the other alternative, a customer (supplier) can receive (send) products from (to) and send (receive) empty containers to (from) the crossdock using LTL carriers. The consolidated shipments at the crossdock are shipped either with round trip (or one-way) TL carriers between the crossdock and the plant or the crossdock may be on milkruns that have some other intermediate stops.

In this model, in addition to the transportation cost, we also have a crossdock cost. The goal is to minimize the total cost in the new system.

The constraints that we need to satisfy are similar to the basic VRP: We need to pick up all supplies and empty customer containers. We need to deliver all customer products and supplier empty containers. Each customer and supplier can use only one mode of transportation and the demands and supplies can not be split into more than one mode and even into more than one truck (This constraint, of course, implicitly assumes that none of the demand or supply values exceed the capacity of a truck. In this project, this assumption holds for the customers and suppliers of our Client). There is also a limit on the total weight and total volume of the trucks. As for the volume, at most 70% of the total volume should be utilized while planning for the shipments. This is the current practice of the Client to ensure that there will be enough space left for unexpected demand fluctuations, which may not be captured during the planning phase. In addition to these constraints, there are additional constraints imposed by the Client to ensure the quality of customer service and have easy-to-coordinate routes.

5.6.1 Customer Sensitive Constraints

- *There can be at most one customer per new milkrun.* This ensures that we will not have to coordinate shipments for two customers.
- *If a customer is on a new milkrun, then it should be the first stop on the milkrun.* This ensures that the promised delivery times can be met by the Client.
- *Customers should not see degradation in mode of shipment* (LTL carriers are considered less reliable than TL carriers).

In addition, the Client also wants the following constraints to be satisfied:

- *The maximum number of stops on a route to be at most 4 (excluding the origin).* This is the current practice the Client has with its suppliers and it is thought that a higher

number of stops on a route may be difficult to coordinate and schedule.

- *If the crossdock is on a new milkrun, then it should be the last stop.* This gives the Client more accurate information and also more control on the arrival time of trucks to its Plant, as there will be no intermediate stops between the crossdock and the Plant.

In the next section, we discuss the data collection and preprocessing phases of this project.

5.7. Data & Preprocessing

We collected customer and supplier data for a quarter of a year. We also collected detailed packaging information, such as volume and weight when full and empty and when collapsed if collapsible. After that, we computed necessary statistics such as average demand, supply, weights, space utilization for all customers and suppliers. Each customer has an average demand that needs to satisfied, but it also has an average supply, which is the average amount of empty returnable containers. This supply value is calculated separately for each customer, based on the type of containers used for the shipments of that customer. Similarly, every supplier has a supply that needs to be delivered to the Client, but it also has a demand value, which is the average amount of empty returnable containers that needs to be delivered back to the supplier.

Moreover, we identified current customer and supplier routes. The supplier routes were easy to obtain as they are being formed by the 3PL that our Client works with. But the customer routes were completely unknown. So, to obtain the route information and the demand load on these routes, we interviewed truck drivers to piece together this information when they come to our Clients plant for pickups.

Scenarios	No of Customers	No of Suppliers	Constraint Set	Savings
Scenario 1	5	8	Basic	26%
Scenario 2	5	8	All	14%
Scenario 3	5	18	All	25%

Table 5.1: Results for different scenarios.

5.7.1 Cost Structure

We estimated the TL costs based on the expertise of the traffic engineers and studies conducted by the 3PL that the Client works with. We used a single fixed per mile cost for TL carriers irrespective of the routes as we see that on average the costs of routes do not differ significantly based on the endpoints of the route. The LTL costs are estimated using the rate software of USF-Holland, an LTL carrier. We tuned these costs by comparing with actual LTL shipment data. The crossdock cost is also a variable cost that is charged on a per container basis for which there is no fixed rental cost or investment.

5.7.2 Initial Analysis

For each potential customer, we computed the savings for the case where the customer do not pick up from the plant and the Client delivers it from its crossdock instead. We considered several cases depending on customer pickup type (TL/Oneway TL/LTL) and identified most promising customers based on volume and total potential savings and included these promising customers in the optimization model.

5.8. Solution & Results

Our results are obtained by using the average demand data and supply data over a quarter of a year for the customers and suppliers, respectively. Using this average data, we solved increasingly complex scenarios and the results are summarized in Table 5.1. In this and later tables, the savings percentage reflect the reduction in cost as a percent of the current

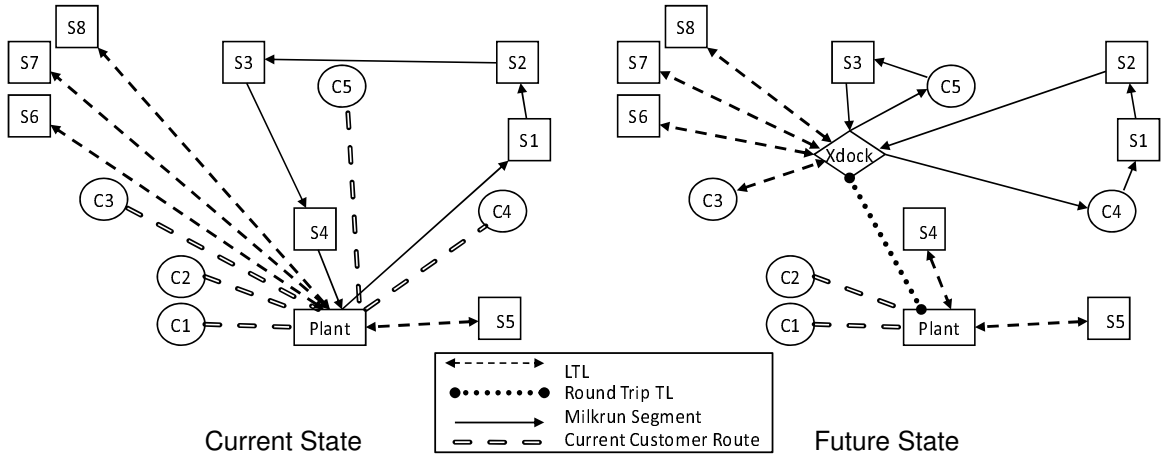

Figure 5.4: Illustration of current and future states for Scenario2.

costs. In the first scenario, we have 5 customers and 8 suppliers and we do not impose the additional customer sensitive constraints. In the second scenario, we have the same set of 5 customers and 8 suppliers, as we have in the first scenario, but now we also impose the customer sensitive constraints. In the third scenario we have the same set of 5 customers but a larger set of 18 suppliers and we impose the customer sensitive constraints.

When we compare the percentage savings among Scenario 1 and 2, we see that the extra customer sensitivity constraints reduce these savings, which is not surprising and reflects the general trade-offs seen in business. Moreover, when we compare Scenario 2 and 3, we see that including more suppliers in the model increased the savings significantly as it allowed more options and better matching of the flows from and to the plant.

To illustrate how a solution looks like, we depict the current and future states for Scenario 2 in Figure 5.4, where customers and suppliers are represented by circles and squares, respectively. The future state is significantly different than the current state. In the future state, C1 and C2 continue using their old milkruns. C3, S6, S7 and S8 use LTL service from and to the crossdock. There are two milkruns starting and ending at the crossdock. The remaining two suppliers (S4 and S5) use LTL service from and to the plant.

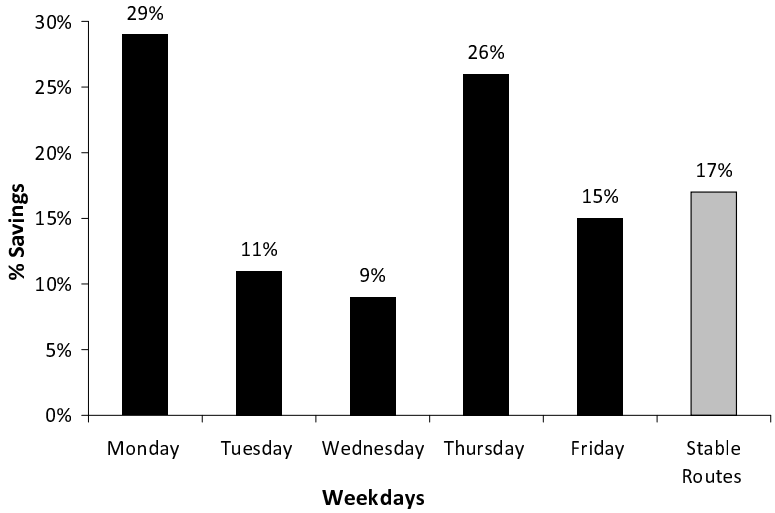

Figure 5.5: Flexibility analysis over the weekdays. The savings fluctuate significantly as different days have different demand structures.

There is also a dedicated round-trip TL truck between the crossdock and the plant.

5.8.1 Flexibility Analysis

We investigate the flexibility of the savings based on the demand variability over the weekdays. For this we compute the average data for each weekday and separately solve the optimization model over these average daily data. To keep the consistency in the transportation modes across the days, we fix the customer transportation modes in the model based on the solution we previously obtained on the overall average data in Scenario 3. Thus, if a customer uses its old milkrun in the solution of Scenario 3, then that customer will be forced to use its old milkrun on each of the week days. Similarly of a customer uses a new milkrun in the solution of Scenario 3, then that customer will be forced to use a new milkrun on each of the days, but in that case we will allow the model to pick different routes on the different days of the week as long as it is a new milkrun. For each week day, the average savings for the week day are shown in Figure 5.5, where we see that the savings fluctuate significantly across the days. The savings are highest on Monday and Thursday,

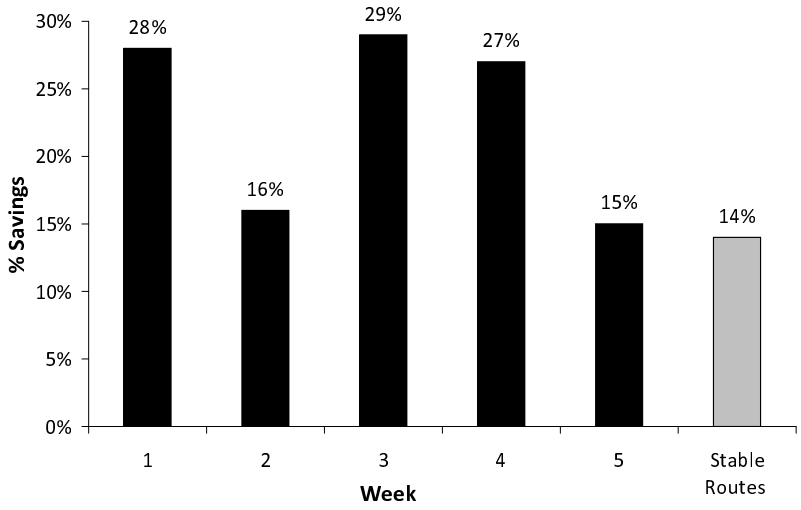

Figure 5.6: Flexibility analysis over the weeks of the quarter. The savings fluctuate significantly as different weeks have different demand structures.

because one of the customers, which is placed on a new milkrun based on the result of Scenario 3, has demands on these two days only. The overall average savings is 18%.

We also solve the model simultaneously on the average data of Mondays and Tuesdays, which have significantly different demand patterns. In that case, we force customers and suppliers to have the exact same routes, which we refer to as *stable routes*, on both days. Interestingly forcing to have the same routes on both days reduce the savings by only 1% (from 18% to 17%). We should note that this difference might have been different if were able to solve the model simultaneously over the average data of five days, instead of just two. But we were unable to do that as that model becomes very large and difficult to solve with the current computational capabilities we have.

We do a similar flexibility analysis for the weeks of the quarter. To find stable routes over the different weeks, we solve the model simultaneously over the 1^{st} and 12^{th} weeks of the quarter, which are far away from each other and have significantly different demand patterns. The results are illustrated in Figure 5.6. The overall average savings is 23%. We see that the savings across the weeks fluctuates but the intensity of the fluctuation is less

	% Savings	
Truck Fill Rate	Average Data	90th Percentile
70%	25%	14%
80%	27%	16%
90%	29%	20%
100%	32%	21%

Table 5.2: Max truck volume utilization on average and 90th percentile data.

than the fluctuation we have seen across the days of the week in Figure 5.5. Interestingly forcing to have the same routes on both days reduce the savings by only 9% (from 23% to 14%).

We also do a "worst-case" scenario analysis in the following way: We take the 90th percentile of the demand and supply requirements of every customer and supplier and solve the model over that data. Obviously we should note such a demand and supply pattern is extremely unlikely. Nevertheless, the results will be useful to see the robustness of the model. Since we are feeding the model with very high demand and supply data, we test the model with different maximum volume utilization values. As noted in the problem formulation, the current value required is 70%. The results are illustrated in Table 5.2. In each of maximum volume utilization rates, the savings with the high demand supply rate are significantly ($\sim 10\%$) less than the savings with the average data. We also see that as we increase the max truck fill rate, the savings increase in both cases.

5.8.2 Utilizing the Crossdock

One alternative setting that extensively utilizes the crossdock may be appealing to the Client, because in such a system, there will be dedicated trucks between the crossdock and the plant and all the routing will be made at the crossdock and this will be an easy-to-manage system. As clearly indicated in Figure 5.7, forcing the flow of customers and suppliers, which do not use their old routes in the new solution, to go through the crossdock reduces the savings we obtained before as this restricts the solution space. When we solve

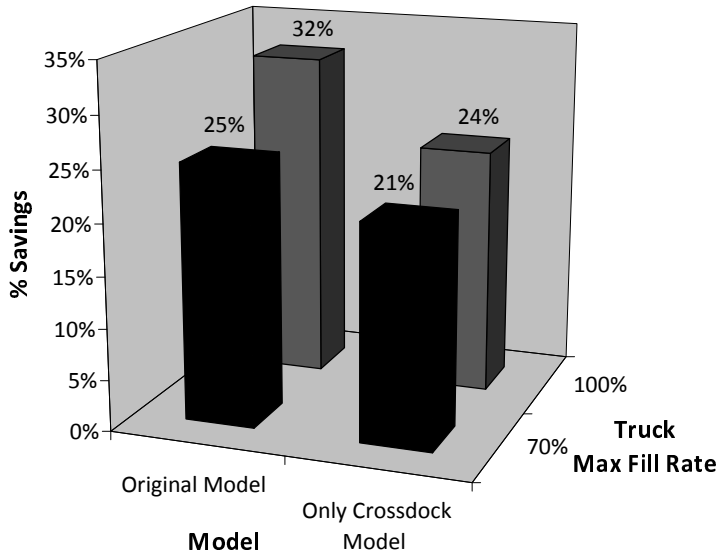

Figure 5.7: Forcing the flow of customers and suppliers, which do not use their old routes, to go through the crossdock reduces the savings at both truck fill rate limits.

the model with this restriction, we see that the savings is actually reduced to 21%, as opposed to the 25% we had before. These savings are at the 70% max truck fill rate. If we allow 100% truck fill rate, the savings goes up by only 3% (to 24%), whereas without the restriction, the savings were much higher (32%).

5.8.3 Inventory Considerations

In addition to the savings in transportation costs, this new system will result in reduced inventories for some of the customers, suppliers and also for the Client because the new system assumes that deliveries and pickups will be made every day. This means an increase in the frequency of deliveries and pickups for some customers and suppliers, which results in reduced inventories since shipments will be made in smaller batches. This is illustrated in Figure 5.8, where the graph on the top is for the case where a customer's demand occurs only twice a week, namely on Mondays and Thursdays, the graph below is for the case where the same customer's demand, which is equal to 2 trucks per week, is spread to five

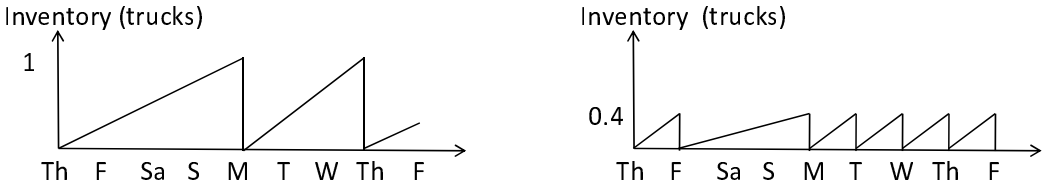

Figure 5.8: Inventory reduction when the frequency of shipments is increased from twice a week (graph on the left) to five times a week (graph on the right).

weekdays. In the first case, the inventory is built from Thursday to Monday to meet the demand on Monday and from Monday to Thursday to meet the demand on Thursday. In the second case, Monday's demand is built from Friday to Monday and every other weekday's demand is built during the previous day. Clearly the inventory amount in the second case is significantly lower (60%) than the first case, not only for the Client but also for the customer. Thus, our solution methodology is aligned with the philosophy and current implementation of the Client's lean production system and represents a win for all involved parties.

5.8.4 Other Considerations

There are some other potential side benefits of this planning model and the new solution it generates. With this tool, the Client will be able to offer different service options to its customers, such as delivering to their location, or allowing them to pickup from the crossdock instead of the plant. The frequency of the shipments can also be adjusted based on customers needs. In addition, the Client thinks that this kind of system-wide improvement will potentially increase their chances to have more demand from auto manufacturers that value continuous improvement and lean practices.

A key challenge in implementing the proposed solution is the perceived lack of control the customers feel over their inbound shipments, which they rightly view as being crucial to their operations. Another business consideration is how the delivery contracts should be re-priced to add the increased delivery costs for the Client, and how this will impact negotiating for customer contracts.

5.9. Summary and Future Research

We studied the customer and supplier logistics planning at a leading automotive parts manufacturer. In this project, we selected a subset of customers and suppliers to consolidate their shipments on the same routes. We also considered the use of a crossdock as a consolidation point. We developed a network flow model combined with vehicle routing from the crossdock and the plant to minimize the system-wise logistics costs. We identified the potential for significant cost savings over the current system based on our model assumptions.

We presented our findings to a wide range of audience at the Client. These include the plant managers and directors and managers of Customer Service, Logistics & Planning departments of two plants. Moreover, we have presented it to the CEO and also to the Logistics Director of the Client. They were happy with the overall outcome, the tangible results that showed real potential for cost savings and with the different design alternatives that we presented. The systematic analysis of the logistics processes that we performed was also highly appreciated.

The challenges mentioned above have prevented the immediate implementation of our proposed solution since it involves renegotiation of customer contracts. However, our analysis of the matching opposite and complementary flows has triggered an ongoing discussion on similar issues in our Client's overall logistics operations, where both flows are within the purview of (different) locations of the Client. The benefits of supply chain coordination, which our study clearly revealed, has provided them with strong evidence for significant savings by overall coordination of logistics operations within the Client's network.

5.9.1 Methodological Extensions

An alternative to the IP model we used in this study is a route-based restricted model. In that, we would consider only a set of routes instead of all possible ones in solving the

problem. For that model, we need to generate a set of good routes using heuristics. We certainly want routes to have low costs. But also we want to generate routes that have daily stable loads as we will run the model every day. For this, one can build routes that have suppliers with negatively correlated shipments on the same routes. So, when the shipment of a supplier is high in one day then it is likely that the other suppliers supply will be low and the load on this route will be stable.

Chapter 6

Appendix

6.1. Appendix for Chapter 2

6.1.1 Structural Properties of TUP

Let P represent the TUP and let $P_{(R)}$ be a relaxation of P with constraint set R, where $R \subset \{1,2,3,4,5\}$. Let $G(s,u) = (H_s, A_s)$ be the game assigned to umpire u in slot s and let H_s be the home team and A_s be the away team for that game. Let C(s,u) represent the constraint which says that Umpire u cannot visit a city more than once in slots s to s+n-1.

Lemma 1 *For $P_{(1,2,4)}$, for $d1 = 0$, for any tournament, for any game there exists a feasible schedule for a single umpire that covers that game.*

Proof: In a given tournament let $G(k,1) = (H_1, A_1)$ be the game assigned to umpire 1 in slot k and let H_1 be the home team and A_1 be the away team for that game. We will show that we can assign games to Umpire1 in slots $\{k_1, ..., k_2\}$, where $k_1 = max(k-n+1, 1)$ and $k_2 = min(k+n-1, 4n-2)$, without violating any C(s,1) for $k_1 \leq s \leq k_2$ (Notice that these constraints are the only constraints of type 4 including slot k).

Umpire1 can be assigned to one of at least $n-1$ games in slot k_1 since C(k_1,1) is the only constraint that includes both k_1 and k. Since only G(k,1) and G(k_1,1) are scheduled, Umpire1 can be assigned to one of at least $n-2$ games in slot k_1+1, and by using a similar argument we can say that Umpire1 can be assigned to one of at least $n-3$ games in slot k_1+2, ..., and to one of at least n-(n-1) = 1 games in slot $k-1$ without violating C(k_1,1). Notice that once G(k_1,1) is scheduled, the critical constraint for slots $\{k_1+1, ..., k-1\}$ became C(k_1,1). All the other constraints including slot s, where $s \neq k_1$, will include a smaller subset of already scheduled games, thus will not be violated by the new game assignment.

Now we will consider slots k+1 through k_2. Umpire1 can be assigned to at least one of the games in slot k+1 without violating C(k_1+1,1), to at least one of the games in slot

k+2 without violating C($k_1 + 2$,1), ..., and to at least one of the games in slot k_2 without violating C(k,1). For each slot $s \in \{k+1, ..., k_2\}$, C(s-n+1,1) includes the most already scheduled games for Umpire1, thus it is the most restricting constraint. All the other constraints including slot s will include a smaller subset of already scheduled games, thus will not be violated by the new game assignment. □

Lemma 2 *For $P_{(1,2,4)}$ and when $n - d1 = n + 1$ in constraint 4: If n is even, there exists a tournament and a game such that no umpire schedule can cover that game.*

Proof: We will show that we can construct such a tournament. Let H_1 be the venue for a game in slot one that is assigned to Umpire1 and let $H^n = \{H_2, ..., H_n\}$ be a set of distinct venues assignable to Umpire1 in slots 2 through n, respectively. This means that the set of venues in slot n+1 has to be set to $H^n \cup \{H_1\}$ to ensure infeasibility of Umpire1's schedule. If we allow a game with a venue H_p be in the tournament in slot s such that $H_p \notin \{H_1, ..., H_n\}$ and $2 \leq s \leq n$, then we can switch the current game assigned to Umpire1 in slot s with H_p and satisfy constraint 4. Thus, no such game should be allowed to be in the tournament. This means that each of the sets of venues in slots 2 to n+1 has to be $H^n \cup \{H_1\}$.

To complete the tournament, we need to build the rest of the games for slot 1. We can do that by pairing the teams in $H^n \cup \{H_1\}$. Thus rest of the teams will be paired as well. Notice that we can do this if n is even. In this tournament, each team H_i, $1 \leq i \leq n$, plays home in only n consecutive games (in slots 2 to n+1) against the same set of n teams ($\{H_j \mid j \notin \{1, .., n\}\}$), which poses no problem in terms of tournament feasibility.

In this tournament, given that we assign Umpire1 to H_1 in slot 1, we can not extend the schedule for Umpire1 for n more slots without violating constraint 4.□

Lemma 3 *For $P_{(1,2,4)}$ and when $n - d1 = n + 1$ in constraint 4: If n is odd, for any tournament, for any game there exists a feasible schedule for a single umpire that covers that game.*

Proof: Let's assume that this claim is incorrect, which means that there exists a tournament and a game such that no umpire schedule can cover that game.

Let H_1 be the venue for a game in slot one that is assigned to Umpire1 and let $H^n = \{H_2, ..., H_n\}$ be a set of distinct venues assignable to Umpire1 in slots 2 through n, respectively. This means that the set of venues in slot n+1 has to be $H^n \cup \{H_1\}$ to ensure infeasibility of Umpire1's schedule. If we allow a game with a venue H_p be in the tournament in slot s such that $H_p \notin \{H_1, ..., H_n\}$ and $2 \leq s \leq n$, then we can switch the current game assigned to Umpire1 in slot s with H_p and satisfy constraint 4. Thus, no such game should be allowed to be in the tournament. This means that each of the sets of venues in slots 2 to n+1 has to be $H^n \cup \{H_1\}$.

To complete the tournament for the first n+1 slots, we need to build the rest of the games for slot 1. In slot 1, a team $H_i \in H^n \cup \{H_1\}$ can not play against a team $H_j \notin H^n \cup \{H_1\}$. Because H_i is already scheduled to play home in n consecutive games against the same set of n teams in slots 2 to n+1. Thus, every team in $H^n \cup \{H_1\}$ has to play against a team in $H^n \cup \{H_1\}$. But since we assumed that n is odd, at least one team in $H^n \cup \{H_1\}$ has to be paired with a team not in $H^n \cup \{H_1\}$. This contradicts with our initial assumption, thus the claim is proved.□

Lemma 4 *For $P_{(1,2,4)}$ and when $n - d1 = n + 2$ in constraint 4: If n is odd, there exists a tournament and a game such that no umpire schedule can cover that game.*

Proof: We will show that we can construct such a tournament. Let H_1 be the venue for a game in slot one that is assigned to Umpire1 and let $H^{n+1} = \{H_2, ..., H_{n+1}\}$ be a set of distinct venues assignable to Umpire1 in slots 2 through n+1, respectively. This means that the set of venues in slot n+2 has to be a subset of $H^{n+1} \cup \{H_1\}$ to ensure infeasibility of Umpire1's schedule. If we allow a game with a venue H_p be in the tournament in slot s such that $H_p \notin \{H_1, ..., H_{n+1}\}$ and $2 \leq s \leq n+1$, then we can switch the current game assigned to Umpire1 in slot s with H_p and satisfy constraint 4. Thus, no such game should be allowed to be in the tournament. This means that each of the sets of venues in slots 2 to n+2 has to be a subset of $H^{n+1} \cup \{H_1\}$.

To complete the tournament, we need to build the rest games for slot 1. We can do that by pairing the teams in $H^{n+1} \cup \{H_1\}$. Thus rest of the teams will be paired as well. Notice that we can do this if n is odd. In this tournament, each team H_i, $1 \leq i \leq n+1$, plays home in only n games against the same set of n+1 teams, which poses no problem in terms of tournament feasibility.

In this tournament, given that we assign Umpire1 to H_1 in slot 1, we can not extend the schedule for Umpire1 for n+1 more slots without violating constraint 4.□

Theorem 1 *For $P_{(1,2,4)}$, for any tournament, for any game there exists a feasible schedule for a single umpire that covers that game. Moreover, in constraint 4 we have the following two properties: When n is even, n is an upper bound for the existence of this feasibility. When n is odd, n+1 is an upper bound for the existence of the feasibility.*

Proof: Lemmas 1, 2, 3 and 4 collectively prove theorem 1.

Theorem 2 *For $P_{(1,2,5)}$, for $d2 = 0$, for any tournament, for any game there exists a*

feasible schedule for a single umpire that covers that game.

Proof: We will prove a stronger statement, which implies Theorem 2. That is, if we replace $\lfloor \frac{n}{2} \rfloor$ with $\lceil \frac{n}{2} \rceil$, the claim still holds. The reason we use $\lfloor \frac{n}{2} \rfloor$ in the definition of Constraint 5 instead of $\lceil \frac{n}{2} \rceil$ is that, once we consider the complete set of constraints, we observe that P becomes infeasible for many instances.

In a given tournament let G(k,1) be the game in consideration. We will show that we can assign games to Umpire1 in slots $\{k_1, ..., k_2\}$, where $k_1 = max(k - \lceil \frac{n}{2} \rceil + 1, 1)$ and $k_2 = min(k + \lceil \frac{n}{2} \rceil - 1, 4n - 2)$, without violating any C(s,1) for $k_1 \leq s \leq k_2$ (Notice that these constraints are the only constraints of type 5 including slot k).

Umpire1 can be assigned to one of at least $n - 2$ games in slot k_1 since C(k_1,1) is the only constraint that includes both k_1 and k. Since only G(k,1) and G(k_1,1) are scheduled, Umpire1 can be assigned to one of at least $n - 4$ games in slot $k_1 + 1$, and by using a similar argument we can say that Umpire1 can be assigned to one of at least $n - 6$ games in slot $k_1 + 2$, ..., and to one of at least $n - 2(\lceil \frac{n}{2} \rceil - 1) \geq 1$ games in slot $k - 1$ without violating C(k_1,1). Notice that once G(k_1,1) is scheduled, the critical constraint for slots $\{k_1 + 1, ..., k - 1\}$ became C(k_1,1). All the other constraints including slot s, where $s \neq k_1$, will include a smaller subset of already scheduled games, thus will not be violated by the new game assignment.

Now we will consider slots k+1 through k_2. Umpire1 can be assigned to at least one of the games in slot k+1 without violating C($k_1 + 1$,1), to at least one of the games in slot k+2 without violating C($k_1 + 2$,1), ..., and to at least one of the games in slot k_2 without violating C(k,1). For each slot $s \in \{k + 1, ..., k_2\}$, C($s - \lceil \frac{n}{2} \rceil + 1, 1$) includes the most already scheduled games for Umpire1, thus it is the most restricting constraint. All the other constraints including slot s will include a smaller subset of already scheduled games, thus

Table 6.1: Partial tournament for 12 teams.

Slot Numbers			
1	**2**	**3**	**4**
(1,2)	(1,6)	(1,10)	(6,1)
(3,11)	(2,10)	(2,6)	(10,2)
(4,5)	(3,12)	(3,8)	(3,11)
(6,10)	(4,9)	(4,5)	(4,7)
(7,9)	(5,7)	(9,7)	(5,9)
(8,12)	(8,11)	(11,12)	(12,8)

will not be violated by the new game assignment.□

Conjecture 1 *For $P_{(1,2,5)}$ and when $\lfloor \frac{n}{2} \rfloor$ is replaced with $\lceil \frac{n}{2} \rceil$+1 in constraint 5: There exists a tournament and a game such that no umpire schedule can cover that game.*

Although we do not have a proof for this claim, we can find such a tournament for 2n=10 and 2n=12 by formulating an integer program. The model results in partial tournaments for slots 1,2,3, and $\lceil \frac{n}{2} \rceil + 1 = 4$ such that no single umpire can cover a specific game in slot 1 without violating Constraint 5. The partial tournament for 12 teams is given in Table 6.1.1. A pair (i,j) means that team i plays team j at team i's home in the slot the pair is placed. In this tournament the game is that can not be officiated by any umpire is (1,2). □

6.1.2 Initial IP Formulation for the TUP

Problem Data

- S, T, U = sets of all slots, teams and umpires, respectively;
- OPP[t,i] = $\begin{cases} j, & \text{if team i plays against team j at venue i in slot t} \\ -j, & \text{if team i plays against team j at venue j in slot t} \end{cases}$
- d_{ij} = travel miles between venues i and j ;

The following constants are defined to have a more readable model:

- $n_1 = n - d_1 - 1$
- $n_2 = \lfloor \frac{n}{2} \rfloor - d_2 - 1$;
- $N_1 = \{0, .., n_1\}$;
- $N_2 = \{0, .., n_2\}$;

Variables

- $x_{isu} = 1$ if game played at venue $i \in T$ in slot $s \in S$ is assigned to umpire $u \in U$ (binary);
- $z_{ijsu} = 1$ if umpire u referees game played at venue i in slot t, then referees game played at venue j in slot $t + 1$ (binary);

The Initial IP Model

$$\text{minimize} \sum_{i \in T} \sum_{j \in T} \sum_{u \in U} \sum_{s \in S : s < |S|} d_{ij} z_{ijsu}$$

subject to

$$\sum_{u \in U} x_{isu} = 1, \ \forall\, i \in T, s \in S : OPP[s,i] > 0 \tag{6.1}$$

$$\sum_{i \in T : OPP[s,i] > 0} x_{isu} = 1, \ \forall\, s \in S, \ u \in U \tag{6.2}$$

$$\sum_{s \in S : OPP[s,i] > 0} x_{isu} \geq 1, \ \forall\, i \in T, \ u \in U \tag{6.3}$$

$$\sum_{s_1 \in N_1} x_{i(s+s_1)u} \leq 1, \ \forall\, i \in T, \ u \in U, \ s \in S : s \leq |S| - n_1 \tag{6.4}$$

$$\sum_{s_2 \in N_2} (x_{i(s+s_2)u} + \sum_{k \in T : OPP[s+s_2,k] > 0} x_{k(s+s_2)u}) \leq 1, \ \forall\, i \in T, \ u \in U, \ s \in S : s \leq |S| - n_2 \tag{6.5}$$

$$x_{isu} + x_{j(s+1)u} - z_{ijsu} \leq 1, \ \forall\, i, j \in T, \ u \in U, \ s \in S : s \leq |S| \tag{6.6}$$

Here is a short description of the constraints. (6.1): every game must be assigned to a single umpire; (6.2): every umpire is assigned to exactly one game per slot; (6.3): every umpire sees every team at least once at the team's home; (6.4): no umpire should visit a venue more than once in any $n - d_1$ consecutive slots; (6.5): no umpire should see a team twice in any $\lfloor \frac{n}{2} \rfloor - d_2$ consecutive slots; (6.6): if umpire u is assigned to a game at venue i in slot s and to a game at venue j in slot $s + 1$, then the umpire should move from i to j in slot s.

6.1.3 Improved IP Formulation for the TUP

Problem Data

- S, T, U = sets of all slots, teams and umpires, respectively;
- OPP[t,i] = $\begin{cases} j, & \text{if team i plays against team j at venue i in slot t} \\ -j, & \text{if team i plays against team j at venue j in slot t} \end{cases}$
- d_{ij} = travel miles between venues i and j ;

The following constants are defined to have a more readable model:

- $n_1 = n - d_1 - 1$
- $n_2 = \lfloor \frac{n}{2} \rfloor - d_2 - 1$;
- $N_1 = \{0, .., n_1\}$;
- $N_2 = \{0, .., n_2\}$;

Variables

- $x_{isu} = 1$ if game played at venue $i \in T$ in slot $s \in S$ is assigned to umpire $u \in U$ (binary);
- $z_{ijsu} = 1$ if umpire u referees game played at venue i in slot t, then referees game played at venue j in slot $t + 1$ (binary);

The Improved IP Model

$$\text{minimize} \sum_{i \in T} \sum_{j \in T} \sum_{u \in U} \sum_{s \in S : s < |S|} d_{ij} z_{ijsu}$$

subject to

$$\sum_{u \in U} x_{isu} = 1, \ \forall\, i \in T, s \in S : OPP[s,i] > 0 \tag{6.7}$$

$$\sum_{i \in T : OPP[s,i] > 0} x_{isu} = 1, \ \forall\, s \in S, \ u \in U \tag{6.8}$$

$$\sum_{s \in S : OPP[s,i] > 0} x_{isu} \geq 1, \ \forall\, i \in T, \ u \in U \tag{6.9}$$

$$\sum_{s_1 \in N_1} x_{i(s+s_1)u} \leq 1, \ \forall\, i \in T, \ u \in U, \ s \in S : s \leq |S| - n_1 \tag{6.10}$$

$$\sum_{s_2 \in N_2} (x_{i(s+s_2)u} + \sum_{k \in T : OPP[s+s_2,k] > 0} x_{k(s+s_2)u}) \leq 1, \ \forall\, i \in T, \ u \in U, \ s \in S : s \leq |S| - n_2 \tag{6.11}$$

$$x_{isu} + x_{j(s+1)u} - z_{ijsu} \leq 1, \ \forall\, i, j \in T, \ u \in U, \ s \in S : s \leq |S| \tag{6.12}$$

We strengthened the formulation with the following additional valid inequalities:

$$x_{isu} = 0, \ \forall\, i \in T, \ u \in U, \ s \in S : OPP[s,i] < 0 \tag{6.13}$$

$$z_{ijsu} - x_{isu} \leq 0, \ \forall\, i, j \in T, \ u \in U, \ s \in S : s < |S| \tag{6.14}$$

$$z_{ijsu} - x_{j(s+1)u} \leq 0, \ \forall\, i, j \in T, \ u \in U, \ s \in S : s < |S| \tag{6.15}$$

$$\sum_{i \in T} z_{ijsu} - \sum_{i \in T} z_{ji(s+1)u} = 0, \ \forall\, j \in T, \ u \in U, \ s \in S : s < |S| - 1 \tag{6.16}$$

$$\sum_{i \in T} \sum_{j \in T} z_{ijsu} = 1, \ \forall\, u \in U, \ s \in S : s < |S| \tag{6.17}$$

6.1.4 Initial CP Model for the TUP in OPL

Model Parameters:

T = {1,..,2n} is the set of teams

S = {1,..,4n-2} is the set of slots

U = {1,..,n} is the set of umpires

$$\text{opponents[t,i]} = \begin{cases} j, & \text{if team i plays against team j at venue i in slot t} \\ -j, & \text{if team i plays against team j at venue j in slot t} \end{cases}$$

dist[i,j] = distance between venues i and j

Decision Variables:

assigned[u,t]= the venue that umpire u is assigned in slot t.

The formulation in the OPL language[45] is as follows:

```
minimize with linear relaxation
   sum (u in U, t in S: t < 4*n-2) dist[assigned[u,t],assigned[u,t+1]]


subject to {

//Constraints (1)&(2)
forall(t in S) {
    distribute( all(i in G) 1,
                all(j in T: opponents[t,j] < 0) -1*opponents[t,j],
                all(u in U) assigned[u,t]); };
//Constraint (3)
forall(u in U)
    atleast(all(i in T) 1, all(i in T) i, all(t in S)assigned[u,t]);
```

```
//Constraint (4)
forall(u in U, t in S: t<= (4*n-2)-(n-d1-1) ){
   atmost(all(i in T) 1, all(j in T) j,
        all(r in [0..n-d1-1]) assigned[u,t+r]);
};
//Constraint (5)*********************************************************
forall(u in U, t in S: t<= (4*n-2)-(n/2-d2-1), j in T,
       r in [1..n/2-d2-1], k in T: opponents[t,k] = j ) {
   (assigned[u,t] = k) => ((assigned[u,t+r]<>j)&(assigned[u,t+r]<>k));};

forall(u in U, t in S: t>= n/2-d2, j in T, r in [1..n/2-d2-1],
       k in T: opponents[t,k] = j ) {
   (assigned[u,t] = k) => ((assigned[u,t-r]<>j)&(assigned[u,t-r]<>k));};

forall(u in U, t in S: t<= (4*n-2)-(n/2-d2-1), j in T,
       r in [1..n/2-d2-1], k in T: opponents[t,k] = j,
       i in T: opponents[t+r,i] = j ) {
   (assigned[u,t] = k) => (assigned[u,t+r] <> i); };

forall(u in U, t in S: t>= n/2-d2, j in T, r in [1..n/2-d2-1],
       k in T: opponents[t,k] = j, i in T: opponents[t-r,i] = j ) {
  (assigned[u,t] = k) => (assigned[u,t-r] <> i); };
//*********************************************************************
};
```

6.1.5 Improved CP Model for TUP in OPL

Model Parameters:

T = {1,..,2n} is the set of teams

S = {1,..,4n-2} is the set of slots

U = {1,..,n} is the set of umpires

$$\text{opponents[t,i]} = \begin{cases} j, & \text{if team i plays against team j at venue i in slot t} \\ -j, & \text{if team i plays against team j at venue j in slot t} \end{cases}$$

dist[i,j] = distance between venues i and j

Decision Variables:

assigned[u,t]= the venue that umpire u is assigned in slot t.

The formulation in the OPL language[45] is as follows:

```
minimize with linear relaxation
   sum (u in U, t in S: t < 4*n-2) dist[assigned[u,t],assigned[u,t+1]]
subject to {
//Constraints (1)&(2)
forall(t in S) {
    distribute( all(i in G) 1,
                all(j in T: opponents[t,j] < 0) -1*opponents[t,j],
                all(u in U) assigned[u,t]); };
//Constraint (3)
forall(u in U)
    atleast(all(i in T) 1, all(i in T) i, all(t in S)assigned[u,t]);
//Constraint (4)
forall(u in U, t in S: t<= (4*n-2)-(n-d1-1) ){
   alldifferent(all(r in [0..n-1]) assigned[u,t+r]);};
```

```
//Constraint (5)*************************************************************

forall(u in U, t in S: t<= (4*n-2)-(n/2-d2-1), j in T,
       r in [1..n/2-d2-1], k in T: opponents[t,k] = j ) {
   (assigned[u,t] = k) => ((assigned[u,t+r]<>j)&(assigned[u,t+r]<>k));};

forall(u in U, t in S: t>= n/2-d2, j in T, r in [1..n/2-d2-1],
       k in T: opponents[t,k] = j ) {
   (assigned[u,t] = k) => ((assigned[u,t-r]<>j)&(assigned[u,t-r]<>k));};

forall(u in U, t in S: t<= (4*n-2)-(n/2-d2-1), j in T,
       r in [1..n/2-d2-1], k in T: opponents[t,k] = j,
       i in T: opponents[t+r,i] = j ) {
   (assigned[u,t] = k) => (assigned[u,t+r] <> i); };

forall(u in U, t in S: t>= n/2-d2, j in T, r in [1..n/2-d2-1],
       k in T: opponents[t,k] = j, i in T: opponents[t-r,i] = j ) {
  (assigned[u,t] = k) => (assigned[u,t-r] <> i); };
//*************************************************************************
};

search {
    forall(t in Slots)
      forall(u in Umps)
         tryall (i in Teams: opponents[t,i] > 0)
            assigned[u,t] = i; };
```

6.2. Appendix for Chapter 4

Mathematical Programming Formulation for HDM-LRP

Model Parameters:

J = {j | j = 1,...,M} is the set of potential kitchen locations

I = {i | i = M+1,..., N+M} is the set of customers

K = {k | k = 1,...,P} is the set of vehicle routes

c_{ij} = distance from point i to point j; $i, j \in I \cup J$

t_{ij} = travel time from point i to point j; $i, j \in I \cup J$

τ_i = maximum time to unload meals for customer $i \in I$

d_i = number of meals required by customer $i \in I$

f_i = fixed cost of establishing a kitchen at site $j \in J$

v_j = variable cost per meal produced at kitchen $j \in J$

g_{ij} = cost per mile of travel from points i to j; $i, j \in I \cup J$

T_k = maximum allowable duration of route $k \in K$

W_j = number of vehicles available at kitchen $j \in J$

M_j = number of meals that can be served by kitchen $k \in K$

$D = \sum_{i \in I} d_i$ = total demand throughout service area

Decision Variables:

$$x_{ijk} = \begin{cases} 1, & \text{if point } i \text{ immediately precedes point } j \text{ on route } k,\ i, j \in I \cup J,\ k \in K\ ; \\ 0, & \text{otherwise.} \end{cases}$$

$$y_j = \begin{cases} 1, & \text{if a kitchen is located at site } j,\ j \in J\ ; \\ 0, & \text{otherwise.} \end{cases}$$

$$z_{ij} = \begin{cases} 1, & \text{if customer } i \in I \text{ is served from kitchen } j \in J \text{ ;} \\ 0, & \text{otherwise.} \end{cases}$$

U_{ik} are auxiliary variables used in subtour elimination constraints.

Model HDM-LRP:

$$optimize\{\sum_{j\in J} f_j y_j + \sum_{i\in I}\sum_{j\in J} v_j d_i z_{ij} + \sum_{k\in K}\sum_{i\in I\cup J}\sum_{j\in I\cup J} g_{ij}c_{ij}x_{ijk}, \sum_{i\in I}\sum_{j\in J} d_i z_{ij}\} \quad (1)$$

s.t.

$$\sum_{k\in K}\sum_{j\in I\cup J} x_{ijk} = 1 \quad \forall i \in I \quad (2)$$

$$\sum_{i\in I}\sum_{j\in J} d_i z_{ij} \leq M_j y_j \quad \forall j \in J \quad (3)$$

$$\sum_{i\in I}\tau_i \sum_{j\in I\cup J} x_{ijk} + \sum_{i\in I\cup J}\sum_{j\in I\cup J} t_{ij}x_{ijk} \leq T_k \quad \forall k \in K \quad (4)$$

$$U_{ik} - U_{jk} + Nx_{ijk} \leq N - 1 \quad \forall i, j \in I, k \in K \quad (5)$$

$$\sum_{i\in I\cup J} x_{ijk} - \sum_{i\in I\cup J} x_{jik} = 0 \quad \forall j \in I, k \in K \quad (6)$$

$$\sum_{j\in J}\sum_{j\in I\cup J} x_{ijk} \leq 1 \quad \forall k \in K \quad (7)$$

$$\sum_{h\in I\cup J} x_{ihk} + \sum_{h\in I\cup J} x_{jhk} - z_{ij} \leq 1 \quad \forall i \in I, j \in J, k \in K \quad (8)$$

$$\sum_{i\in I}\sum_{k\in K} x_{ijk} \leq W_j \quad \forall j \in J \quad (9)$$

$$x_{ijk} \in \{0, 1\} \quad \forall i \in I \cup J, j \in I \cup J, k \in K \quad (10)$$

$$y_j \in \{0,1\} \quad \forall j \in J \tag{11}$$

$$z_{ij} \in \{0,1\} \quad \forall i \in I, j \in J \tag{12}$$

$$U_{ik} \geq 0 \text{ and integer} \quad \forall i \in I, k \in K \tag{13}$$

Objective (1) is composed of two competing measures. The first, a measure of efficiency, is a total cost term equal to the sum of fixed kitchen location costs, variable kitchen throughput costs and variable delivery costs. This objective is minimized. (Note that by varying the variable costs v_j and g_{ij}, more or less weight is placed on these volunteer-donated activities.) The second objective, an effectiveness measure, represents the total demand served, which is to be maximized.

Constraints (2) ensure that each customer is served on a single route by a single kitchen. Constraints (3) ensure that a kitchen, if located, cannot serve more than its maximum capacity of meals. Constraints (4) ensure that the duration of each route does not exceed the maximum allowable duration. Constraints (5) require that every delivery route be connected to a kitchen. They also act as subtour elimination constraints. Constraints (6) ensure that a vehicle leaves every node that it enters. Constraints (7) ensure that a route is assigned to at most one kitchen. Constraints (8) are linking constraints that ensure that a customer may be allocated to a kitchen only if there is a route assigned to that kitchen that serves that customer. Constraints (9) ensure that the number of routes assigned to each kitchen does not exceed the maximum allowable number of routes. Constraints (11), (12), and (13) ensure that the routing variables, location variables, and allocation variables can take on values 0 or 1. Constrains (14) ensure that the auxiliary variables are nonnegative integers.

6.3. Appendix for Chapter 5

Optimization Model

Data and Parameters

pl = the plant.

xd = the crossdock.

C = set of all customers.

S = set of all suppliers.

N =set of all customers and suppliers: $N = C \cup S$.

$N^{'}$ = set of all nodes including the crossdock and the plant: $N^{'} = N \cup \{$ pl, xd $\}$.

C^{LTL} = set of customers currently on LTL routes.

S^{TL} = set of suppliers currently on milkruns.

R^{S} = set of current supplier milkruns.

S_{j}^{TL} = set of suppliers currently on milkrun $j \in R^{S}$.

$dist_{i,j}$ = distance between nodes $i, j \in N^{'}$.

d_{i} = average demand for node $i \in N$ in units of a truck load.

s_{i} = average supply for node $i \in N$ in units of a truck load.

d_{i}^{NPack} = average demand for node $i \in N$ in units of packages.

s_{i}^{NPack} = average supply for node $i \in N$ in units of packages.

d_{i}^{weight} = average weight of node $i \in N$'s demand in pounds.

s_{i}^{weight} = average weight of node $i \in N$'s supply in pounds.

$maxstops$ = maximum number of stops allowed on a route excluding the origin (pl or xd).

$maxvol$ = maximum percentage volume capacity allowed on a TL in units of a truck.

$maxweight$ = maximum weight capacity of a truck in pounds.

m = an arbitrary large number.

Costs

p^{TL} = per mile cost for round trip TL trucks.

p^{OneWay} = per mile cost for oneway TL trucks.

$p^{TL-Stop}$ = per stop charge for a TL truck.

$p^{LTL}_{i,pl}$ = LTL cost between $i \in S$ and the Plant.

$p^{LTL}_{i,xd}$ = LTL cost between $i \in N$ and the crossdock.

p^{h}_{xd} = per container handling cost at the crossdock.

$p^{TL}_{i}(C)$ = current milkrun cost for $i \in C$.

$p^{TL}_{j}(S)$ = current milkrun cost for current supplier route $j \in R^S$.

Decision Variables

We used three types of variables in this model: Binary, General Integer and Continuous.

Binary Variables

$Z_r = 1$ if the current supplier milkrun $r \in R^S$ is used, 0 otherwise.

$V_i = 1$ if node i uses its current milkrun, 0 otherwise, $\forall i \in N$.

$Y_i = 1$ if node i uses LTL to and from the Plant, 0 otherwise, $\forall i \in N$.

$W_i = 1$ if node i is on a new milkrun, 0 otherwise, $\forall i \in N$.

$X_i = 1$ if node i uses LTL to&from the crossdock, 0 otherwise,$\forall i \in N$.

$Dock^d_i = 1$ if node i's demand goes through the crossdock and 0 otherwise,$\forall i \in N$.

$Dock^s_i = 1$ if node i's supply goes through the crossdock and 0 otherwise,$\forall i \in N$.

$Visit^i_j = 1$ if i's demand or supply goes through node j,and 0 otherwise $\forall i \in N, \forall j \in N'$.

General Integer Variables

RT = number of round trip trucks between the crossdock and the Plant.

$A_{u,v}$ = number of times the milkrun arc (u, v) is used, $\forall u, v \in N'$.

$OW_{pl,xd}$ = Number of times the one way arc from the Plant to the crossdock is used.

$OW_{xd,pl}$ = Number of times the one way arc from the crossdock to the Plant is used.

Continuous (Flow) Variables

$F^{TL}_{u,v}(i)$ = flow of node i's demand flowing on milkrun arc (u,v), $\forall u,v \in N^{'}, \forall i \in N$.

$G^{TL}_{u,v}(i)$ = flow of node i's supply flowing on milkrun arc (u,v), $\forall u,v \in N^{'}, \forall i \in N$.

$F^{RT}_{pl,xd}(i)$ = flow of i's demand flowing from the Plant to the crossdock on RT trucks, $\forall i \in N$.

$G^{RT}_{xd,pl}(i)$ = flow of i's supply flowing from xd to pl on RT trucks, $\forall i \in N$.

$F^{OW}_{pl,xd}(i)$ = flow of i's demand flowing from pl to xd on one way trucks, $\forall i \in N$.

$G^{OW}_{xd,pl}(i)$ = flow of node i's supply from xd to pl on one way trucks, $\forall i \in N$.

Objective Function

$$\text{minimize}(MilkrunCost^{new} + StopCost^{new} + MilkrunCost^{old}_{customer} + MilkrunCost^{old}_{supplier}$$
$$+LTLcost_{xd} + LTLcost_{pl} + RTcost + OneWayCost + CrossdockCost)$$

where,

$MilkrunCost^{new} = p^{TL} * (\sum_{u,v \in N'} dist_{u,v} * A_{u,v})$: the total cost of all new milkruns.

$StopCost^{new} = p^{TL-Stop} * (\sum_{i \in N} W_i + A_{xd,pl} + A_{pl,xd} + RT)$: the total stopping cost incurred in the new milkruns.

$MilkrunCost^{old}_{customer} = \sum_{i \in C} V_i * p^{TL}_i(C)$: the total cost of current milkruns used by the customers in the new solution.

$MilkrunCost^{old}_{supplier} = \sum_{i \in R^S} Z_i * p^{TL}_i(S)$: the total cost of current milkruns used by the suppliers in the new solution.

$LTLcost_{xd} = \sum_{i \in N} X_i * p^{LTL}_{i,xd}$: the total cost of LTL trucks between nodes and crossdock.

$LTLcost_{pl} = \sum_{i \in S} Y_i * p^{LTL}_{i,pl}$: the total cost of LTL trucks between the nodes and the plant.

$RTcost = RT * p^{TL} * (dist_{pl,xd} + dist_{xd,pl})$: the total cost of round trip TL trucks between the plant and the crossdock.

$OneWayCost = p^{OneWay} * (OW_{pl,xd} * dist_{pl,xd} + OW_{xd,pl} * dist_{xd,pl})$: the total cost of one way TL trucks between the plant and the crossdock.

$CrossdockCost = p^h_{xd} * \sum_{i \in N} (Dock^d_i * d^{NPack}_i + Dock^s_i * s^{NPack}_i)$: total crossdock usage cost.

Constraints

For ease of reading, we grouped similar constraints below.

$$Y_i + V_i + W_i + X_i = 1, \ \forall i \in N \tag{6.18}$$

$$V_i = 0, \ \forall i \in S \backslash S^{TL}. \tag{6.19}$$

$$Y_i = 0, \ \forall i \in N \cup S^{TL}. \tag{6.20}$$

$$|S_i^{TL}| * Z_i \geq \sum_{j \in S_i^{TL}} V_j, \ \forall i \in R^S. \tag{6.21}$$

$$\sum_{u \in N'} F_{u,i}^{TL}(i) = d_i * W_i, \ \forall i \in N. \tag{6.22}$$

$$\sum_{u \in N'} G_{i,u}^{TL}(i) = s_i * W_i, \ \forall i \in N. \tag{6.23}$$

$$\sum_{u \in N \cup \{pl\}} (F_{u,xd}^{TL}(i) + F_{xd,u}^{TL}(i)) + F_{pl,xd}^{RT}(i) + F_{pl,xd}^{OW}(i) \leq m * Dock_i^d, \ \forall i \in N \tag{6.24}$$

$$\sum_{u \in N \cup \{pl\}} (G_{u,xd}^{TL}(i) + G_{xd,u}^{TL}(i)) + G_{xd,pl}^{RT}(i) + G_{xd,pl}^{OW}(i) \leq m * Dock_i^s, \ \forall i \in N \tag{6.25}$$

$$F_{i,u}^{TL}(i) + G_{u,i}^{TL}(i) + F_{u,u}^{TL}(i) + G_{u,u}^{TL}(i) + G_{pl,u}^{TL}(i) + F_{u,pl}^{TL}(i) = 0 \ \forall i \in N, \forall u \in N' \tag{6.26}$$

Short Description: (6.18): only one mode of transportation must be used by each customer and supplier; (6.19): suppliers that currently use LTL service, naturally, do not have old milkruns to be assigned in the new solution; (6.20): only the suppliers that currently use LTL service can use an LTL service to and from the plant; (6.21): if a supplier j that is on the current milkrun i uses that new milkrun in the new solution, Z_i becomes 1; (6.22 & 6.23): if node i is on a new milkrun, than the amount of i's demand flow into node i should be equal to its demand and the amount of i's supply flow out of node i should be equal to its supply; (6.24 & 6.25): If node i's flow passes through the crossdock, then the crossdock is used by node i; (6.26): unallowed flows are set to zero;

$$\sum_{v \in N'} A_{u,v} \leq 1 \ \ \forall u \in N \tag{6.27}$$

$$A_{u,v} + A_{v,u} \leq 1 \ \ \forall u, v \in N' \backslash \{pl\} \tag{6.28}$$

$$\sum_{v \in N'} A_{v,u} = \sum_{v \in N'} A_{u,v} \ \ \forall u \in N \tag{6.29}$$

$$\sum_{u \in N} A_{xd,u} = \sum_{u \in N} A_{u,xd} \tag{6.30}$$

Short Description: (6.27): at most one milkrun truck should enter each customer or supplier; (6.28): only one of the two arcs between two nodes can be used;(6.29, 6.30): number of milkrun trucks entering and leaving a node should be equal;

Flow Balance Type Constraints:

$$V_i * d_i + Y_i * d_i + F^{RT}_{pl,xd}(i) + F^{OW}_{pl,xd}(i) + \sum_{u \in N'} F^{TL}_{pl,u}(i) = d_i, \ \ \forall i \in N \tag{6.31}$$

$$V_i * s_i + Y_i * s_i + G^{RT}_{xd,pl}(i) + G^{OW}_{xd,pl}(i) + \sum_{u \in N'} G^{TL}_{u,pl}(i) = s_i, \ \ \forall i \in N \tag{6.32}$$

$$\sum_{u \in N'} F^{TL}_{u,j}(i) - \sum_{u \in N'} F^{TL}_{j,u}(i) = 0, \ \ \forall i, j \in N : i \neq j \tag{6.33}$$

$$\sum_{u \in N'} G^{TL}_{u,j}(i) - \sum_{u \in N'} G^{TL}_{j,u}(i) = 0, \ \ \forall i, j \in N : i \neq j \tag{6.34}$$

$$\sum_{u \in N'} F^{TL}_{u,xd}(i) + F^{RT}_{pl,xd}(i) + F^{OW}_{pl,xd}(i) - \sum_{u \in N'} F^{TL}_{xd,u}(i) - X_i * d_i = 0, \ \ \forall i \in N. \tag{6.35}$$

$$\sum_{u \in N'} G^{TL}_{xd,u}(i) + G^{RT}_{xd,pl}(i) + G^{OW}_{xd,pl}(i) - \sum_{u \in N'} G^{TL}_{u,xd}(i) - X_i * s_i = 0, \ \ \forall i \in N. \tag{6.36}$$

Short Description: (6.31 & 6.32): supply and demand satisfaction at the plant; (6.33 & 6.34): flow balance at nodes except the crossdock and the plant; (6.35 & 6.36) demand and supply flow balance at the crossdock;

Volume and Weight Capacity Constraints:

$$\sum_{i \in N} (F^{TL}_{u,v}(i) + G^{TL}_{u,v}(i)) \leq A_{u,v} * maxvol \;\; \forall u, v \in N^{'}. \tag{6.37}$$

$$\sum_{i \in N} F^{RT}_{pl,xd}(i) \leq RT * maxvol \tag{6.38}$$

$$\sum_{i \in N} G^{RT}_{xd,pl}(i) \leq RT * maxvol \tag{6.39}$$

$$\sum_{i \in N} F^{OW}_{pl,xd}(i) \leq OW_{pl,xd} * maxvol \tag{6.40}$$

$$\sum_{i \in N} G^{OW}_{xd,pl}(i) \leq OW_{xd,pl} * maxvol \tag{6.41}$$

$$\sum_{i \in N} (d^{weight}_{i} * F^{TL}_{u,v}(i) + s^{weight}_{i} * G^{TL}_{u,v}(i)) \leq maxweight * A_{u,v}, \;\; \forall u, v \in N^{'} \tag{6.42}$$

$$\sum_{i \in N} d^{weight}_{i} * F^{RT}_{pl,xd}(i) \leq maxweight * RT \tag{6.43}$$

$$\sum_{i \in N} s^{weight}_{i} * G^{RT}_{xd,pl}(i) \leq maxweight * RT \tag{6.44}$$

$$\sum_{i \in N} d^{weight}_{i} * F^{OW}_{pl,xd}(i) \leq maxweight * OW_{pl,xd} \tag{6.45}$$

$$\sum_{i \in N} s^{weight}_{i} * G^{OW}_{xd,pl}(i) \leq maxweight * OW_{xd,pl} \tag{6.46}$$

Short Description: (6.37, 6.38, 6.39, 6.40, 6.41): the total flow between two nodes can not exceed the total volume capacity on the TL trucks operating between those two nodes; (6.42, 6.43, 6.44, 6.45, 6.46): the total flow between two nodes can not exceed the total weight capacity on the TL trucks operating between those two nodes;

Customer Sensitive Constraints:

$$\sum_{k \in N'} G^{TL}_{k,j}(i) \leq Visit^i_j \ \ \forall i \in N, \forall j \in S \cup \{pl, xd\} : i \neq j \tag{6.47}$$

$$\sum_{k \in N'} F^{TL}_{j,k}(i) \leq Visit^i_j \ \ \forall i \in N, j \in N^{'} : i \neq j \tag{6.48}$$

$$\sum_{j \in N' : i \neq j} Visit^i_j \leq maxstops \ \ \forall i \in N. \tag{6.49}$$

$$X_i = 0 \ \ \forall i \in C \backslash C^{LTL} \tag{6.50}$$

$$A_{i,j} = 0 \ \ \forall i, j \in C \tag{6.51}$$

$$F^{TL}_{v,i}(i) + G^{TL}_{i,v}(i) = 0 \ \ \forall v \in N^{'}, \forall i \in C, \forall j \in C \backslash C^{LTL} : i \neq j \tag{6.52}$$

$$A_{i,j} = 0 \ \ \forall i \in S, \forall j \in C \tag{6.53}$$

$$\sum_{k \in N', j \in N : j \neq i} G^{TL}_{i,k}(j) \leq 0 \ \ \forall i \in C \tag{6.54}$$

$$\sum_{k \in N', j \in C} F^{TL}_{i,k}(j) \leq 0 \ \ \forall i \in S \tag{6.55}$$

$$\sum_{k \in N, j \in N} G^{TL}_{xd,k}(j) \leq 0 \tag{6.56}$$

Short Description: (6.47, 6.48): if node i's supply or demand flows through a node j, then node i's flow visits node j; (6.49): the number of nodes that node i's flow goes through is at most maxstops;(6.50): customers should not ship with LTL unless they currently use LTL; (6.51): no arc between two customers can be used;(6.52): no customer's demand or supply should go through another customer; (6.53): no arc from a supplier to a customer can be used;(6.54): no supply flow should go through a customer; (6.55): no customer demand flow should go through a supplier;(6.56): the crossdock should only be the last stop on a milkrun (no supply flow should flow from crossdock to other customers or suppliers).

Bibliography

[1] F.S. Al-Anzi, Y.N. Sotskov, A. Allahverdi, and G.V. Andreev. Using mixed graph coloring to minimize total completion time in job shop scheduling. *Applied Mathematics and Computation*, 182(2):1137–1148, 2006.

[2] M. Allen, G. Kumaran, and T. Liu. A combined algorithm for graph-coloring in register allocation. *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations, Ithaca, New York, USA*, 2002.

[3] C. Avanthay, A. Hertz, and N. Zufferey. A variable neighborhood search for graph coloring. *European Journal of Operational Research*, 151(2):379–388, 2003.

[4] R. Baldacci, N. Christofides, and A. Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming Series A*, 2007.

[5] N. Barnier and P. Brisset. Graph Coloring for Air Traffic Flow Management. *Annals of Operations Research*, 130(1):163–178, 2004.

[6] S.S. Barreto. Location-Routing Problems (LRP), http://sweet.ua.pt/∼ iscf143/_private/SergioBarretoHomePage.htm, 2008.

[7] J.J. Bartholdi, L.K. Platzman, R.L. Collins, and W.H. Warden. A Minimal Technology Routing System for Meals on Wheels. *Interfaces*, 13(3):1–8, 1983.

[8] J.C. Beck, P. Prosser, and E. Selensky. On the reformulation of vehicle routing problems and scheduling problems. *LNAI 2371, Proceedings of the Symposium on Abstraction, Reformulation and Approximation (SARA 2002)*, pages 282–289, 2002.

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

[10] R. Berger. Location-Routing Models for Distribution System Design. *Unpublished dissertation. Northwestern University, Department of Industrial Engineering and Management Sciences, Evanston, Ill., USA.*, 1997.

[11] J.H. Bookbinder and K.E. Reece. Vehicle routing considerations in distribution system design. *European Journal of Operational Research*, 37(2):204–213, 1988.

[12] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(11):1222–1239, 2001.

[13] D. Briskorn. *Sports Leagues Scheduling: Models, Combinatorial Properties, and Optimization Algorithms*. Springer Verlag, 2008.

[14] TN Bui and C. Patel. An Ant system Algorithm for Coloring Graphs. *Computational Symposium on Graph Coloring and Its Generalizations, COLOR02, Cornell University, Ithaca, NY*, 2002.

[15] S. Burer, RDC Monteiro, and Y. Zhang. CirCut: A Fortran 90 Code for Max-Cut, Max-Bisection and More,
http://www.caam.rice.edu/∼zhang/circut/.

[16] S. Burer, RDC Monteiro, and Y. Zhang. Rank-two relaxation heuristics for max-cut and other binary quadratic programs. *SIAM J. on Optimization*, 12(2):503–521, 2002.

[17] M. Chiarandini, I. Dumitrescu, and T. Stutzle. Local Search for the Colouring Graph Problem. A Computational Study. *TU Darmstadt, FB Intellektik, FG Informatik*

Hochschulstr. 10, 64289 Darmstadt, German.

[18] M. Chiarandini and T. Stutzle. An application of iterated local search to graph coloring. *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations*, pages 112–125, 2002.

[19] N. Christofides and S. Eilon. An algorithm for the vehicle dispatching problem. *Operational Research Quarterly*, 20(3):309–318, 1969.

[20] G. Cornuejols, M.L. Fisher, and G.L. Nemhauser. Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. *Management Science*, 23(8):789–810, 1977.

[21] E.S. Correa, M.T.A. Steiner, A.A. Freitas, and C. Carnieri. A Genetic Algorithm for the P-Median Problem. *Proc. 2001 Genetic and Evolutionary Computation Conf.(GECCO-2001)*, pages 1268–1275, 2001.

[22] C. Croitoru, H. Luchian, O. Gheorghies, and A. Apetrei. A new genetic graph coloring heuristic. *Computational Symposium on Graph Coloring and Generalizations COLOR*, 2, 2002.

[23] M.W. Dawande and J.N. Hooker. Inference-Based Sensitivity Analysis for Mixed Integer/Linear Programming. *Operations Research*, 48(4):623–634, 2000.

[24] D. de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19(2):151–162, 1985.

[25] A.R. Duarte, C.C. Ribeiro, S. Urrutia, and E.H. Haeusler. Referee Assignment in Sports Leagues. *Lecture Notes in Computer Science*, 3867:158, 2007.

[26] K.L. Dunn. Euclidian Traveling Salesman Problem Solver, http://fly.hiwaay.net:8000/∼ kdunn/problems/tsp.shtml, 2008.

[27] K. Easton, G. Nemhauser, and M. Trick. The traveling tournament problem description and benchmarks. *Seventh International Conference on the Principles and Practice of Constraint Programming (CP99)*, pages 580–589, 2001.

[28] K. Easton, G. Nemhauser, and M. Trick. Sports scheduling. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, 2004.

[29] A. Farmer, J.S. Smith, and L.T. Miller. Scheduling Umpire Crews for Professional Tennis Tournaments. *Interfaces*, 37(2):187, 2007.

[30] R. Fukasawa, H. Longo, J. Lysgaard, M.P. Aragão, M. Reis, E. Uchoa, and R.F. Werneck. Robust Branch-and-Cut-and-Price for the Capacitated Vehicle Routing Problem. *Mathematical Programming*, 106(3):491–511, 2006.

[31] P. Galinier and A. Hertz. A survey of local search methods for graph coloring. *Computers and Operations Research*, 33(9):2547–2562, 2006.

[32] P. Galinier, A. Hertz, and N. Zuffrey. Adaptive memory algorithms for graph colouring. *Computational Symposium on Graph Coloring and Generalizations (COLOR02), Ithaca, NY*, pages 75–82.

[33] M. Gamache, A. Hertz, and J.O. Ouellet. A graph coloring model for a feasibility problem in monthly crew scheduling with preferential bidding. *Computers and Operations Research*, 34(8):2384–2395, 2007.

[34] A. Gamst. Some lower bounds for a class of frequency assignment problems. *IEEE Transactions of Vehicular Technology*, 35(1):8–14, 1986.

[35] M.R. Garey and D.S Johnson. *Computers and Interactibility: A Guide to the Theory of NP-Completeness*. WH Freeman and Company: New York, 1979.

[36] M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

[37] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1989.

[38] W.L. Gorr, M.P. Johnson, and S. Roehrig. Spatial Decision Support System for Home-Delivered Services. *Journal of Geographic Systems*, 3(2):181–197.

[39] IE Grossmann and V. Jain. Algorithms for hybrid milp/cp models for a class of optimization problems. *INFORMS Journal on Computing*, 13:258–276, 2001.

[40] P.H. Hansen, B. Hegedahl, S. Hjortkjaer, and B. Obel. A heuristic solution to the warehouse location-routing problem. *European Journal of Operational Research*, 76(1):111–127, 1994.

[41] I. Harjunkoski and I.E. Grossmann. Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods. *Computers and Chemical Engineering*, 26(11):1533–1552, 2002.

[42] J.H. Holland. *Adaptation in natural and artificial systems.* University of Michigan Press Ann Arbor, 1975.

[43] J.N. Hooker. Planning and scheduling by logic-based benders decomposition. *Operations Research*, 55(3):588, 2007.

[44] J.N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96(1):33–60, 2003.

[45] ILOG Inc. ILOG OPL Studio 3.7 Language Manual, 2003.

[46] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning. *Operations Research*, 39(3):378–406, 1991.

[47] M.P. Johnson, W.L. Gorr, and S. Roehrig. Location/Allocation/Routing for Home-Delivered Meals Provision: Models & Solution Approaches. *International Journal of Industrial Engineering*, 9(1):45–56, 2002.

[48] R.M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, 43:85–103, 1972.

[49] S. Kirkpatrick, CD Gelatt, and MP Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[50] G. Laporte. Location Routing Problems. *Vehicle routing: Methods and Studies, (Eds.: B.L. Golden and A.A. Assad)*, pages 163–197, 1988.

[51] G. Laporte and P.J. Dejax. Dynamic location-routeing problems. *Journal of the Operational Research Society*, 40(5):471–482, 1989.

[52] G. Laporte, F. Louveaux, and H. Mercure. Models and exact solutions for a class of stochastic location-routing problems. *European Journal of Operational Research*, 39(1):71–78, 1989.

[53] G. Laporte and Y. Nobert. Exact Algorithm for Minimizing Routing and Operating Costs in Depot Location. *European Journal of Operational Research*, 6(2):224–226, 1981.

[54] G. Laporte, Y. Nobert, and D. Arpin. An exact algorithm for solving a capacitated location-routing problem. *Annals of Operations Research*, 6(9):291–310, 1986.

[55] G. Laporte, Y. Nobert, and P. Pelletier. Hamiltonian location problems. *European Journal of Operational Research*, 12(1):82–89, 1983.

[56] G. Laporte, Y. Nobert, and S. Taillefer. Solving a family of multi-depot vehicle routing and location-routing problems. *Transportation Science*, 22(3):161–172, 1988.

[57] F.T. Leighton. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84(6):489–503, 1979.

[58] J.K. Lenstra and A.H.G. Rinnooy Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11:221–227, 1981.

[59] G. Lewandowski and A. Condon. Experiments with Parallel Graph Coloring Heuristics and Applications of Graph Coloring. *Cliques, Coloring, and Satisfiability: Second Dimacs Implementation Challenge, October 11-13, 1993*, 1996.

[60] C.K.Y. Lin, CK Chow, and A. Chen. A location-routing-loading problem for bill delivery services. *Computers & Industrial Engineering*, 43(1-2):5–25, 2002.

[61] S. Lin. Computer solutions of the traveling salesman. *Bell System Tech. J.*, 44:2245–2269, 1965.

[62] J. Lysgaard, A.N. Letchford, and R.W. Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100(2):423–445, 2004.

[63] O.B.G. Madsen. Methods for solving combined two level location-routing problems of realistic dimensions. *European Journal of Operational Research*, 12(3):295–301, 1983.

[64] A. Mehrotra and MA Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8:344–354, 1996.

[65] NK Mehta. The application of a graph coloring method to an examination scheduling problem. *Interfaces*, 11(5):57–64, 1981.

[66] D. Mester and O. Bräysy. Active guided evolution strategies for large-scale vehicle routing problems with time windows. *Computers and Operations Research*, 32(6):1593–1614, 2005.

[67] Z. Michalewicz. *Genetic Algorithms+ Data Structures= Evolution Programs*. Springer, 1996.

[68] G. Nagy and S. Salhi. Nested Heuristic Methods for the Location-Routing Problem. *Journal of the Operational Research Society*, 47:1166–1174, 1996.

[69] I. Or and W.P. Pierskalla. A Transportation Location-Allocation Model for Regional Blood Banking. *IIE Transactions*, 11(2):86–95, 1979.

[70] R.I.L.E. Ordonez. *Solving the American League umpire crew scheduling problem using constraint login programming.* PhD thesis, Illinois Institute of Technology, 1997.

[71] J. Perl and M.S. Daskin. A Warehouse Location-Routing Problem. *Transportation Research*, 19B:381–396, 1985.

[72] V. Phan and S. Skiena. Coloring graphs with a general heuristic search engine. *Computational Symposium on Graph Coloring and its Generalizations*, pages 92–99, 2002.

[73] M. Pinedo and X. Chao. *Operations scheduling.* Irwin/McGraw-Hill Boston, Mass, 1999.

[74] C. Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers and Operations Research*, 31(12):1985–2002, 2004.

[75] R.V. Rasmussen and M.A. Trick. A Benders approach for the constrained minimum break problem. *European Journal of Operational Research*, 177(1):198–213, 2007.

[76] R.V. Rasmussen and M.A. Trick. Round robin scheduling–a survey. *European Journal of Operational Research*, 188(3):617–636, 2008.

[77] J. Renaud, G. Laporte, and F.F. Boctor. A tabu search heuristic for the multi-depot vehicle routing problem. *Computers and Operations Research*, 23(3):229–235, 1996.

[78] A. Schaerf. A Survey of Automated Timetabling. *Artificial Intelligence Review*, 13(2):87–127, 1999.

[79] R. Srivastava. Alternate solution procedures for the location-routing problem. *Omega International Journal of Management Science*, 21(4):497–506, 1993.

[80] CD Tarantilis and CT Kiranoudis. BoneRoute: An Adaptive Memory-Based Method for Effective Fleet Management. *Annals of Operations Research*, 115(1):227–241, 2002.

[81] J. Tavares, F.B. Pereira, P. Machado, and E. Costa. On the influence of GVR in vehicle routing. *Proceedings of the 2003 ACM symposium on Applied Computing*, pages 753–758, 2003.

[82] M.A. Trick. Challenge Traveling Tournament Instances, http://mat.gsia.cmu.edu/TOURN/.

[83] M.A. Trick. COLOR02/03/04: Graph Coloring and its Generalizations. http://mat.gsia.cmu.edu/COLOR04.

[84] M.A. Trick. Integer and constraint programming approaches for round robin tournament scheduling. *Lecture Notes in Computer Science*, 2740:63–77, 2003.

[85] D. Tuzun and L.I. Burke. A two-phase tabu search approach to the location routing problem. *European Journal of Operational Research*, 116(1):87–99, 1999.

[86] R.V.V. Vidal. *Applied Simulated Annealing*. Springer Verlag, 1993.

[87] J.P. Womack, D.T. Jones, and D. Roos. *The Machine that Changed the World: The Story of Lean Production*. HarperPerennial, 1991.

[88] D.W.S. Wong and J.W. Meyer. A Spatial Decision Support System Approach to Evaluate the Efficiency of a Meals-on-Wheels Program. *The Professional Geographer*, 45(3):332–341, 1993.

[89] MB Wright. Scheduling English cricket umpires. *Journal of the Operational Research Society*, 42(6):447–452, 1991.

[90] MB Wright. A Rich Model For Scheduling Umpires For An Amateur Cricket League. *Lancaster University Management School Working Paper*, 2004.

[91] T.H. Wu, C. Low, and J.W. Bai. Heuristic solutions to multi-depot location-routing problems. *Computers and Operations Research*, 29(10):1393–1415, 2002.

[92] H. Yildiz. Traveling Umpire Problem (TUP), http://www.andrew.cmu.edu/user/hakanyil/TUP/.

[93] T. Yunes, H. Barringer, J. Levine, and M. Trick. Scheduling Major League Baseball Umpires. *Presentation at the INFORMS Annual Meeting, Pittsburgh, PA, November 5-8, 2006.*