RICE UNIVERSITY

A Computational Study of Vehicle Routing Applications

by

Jennifer L. Rich

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

Doctor of Philosophy

Approved, Thesis Committee:

William J. Cook, Chairman
Noah Harding Professor of Computational
and Applied Mathematics

Robert E. Bixby
Noah Harding Professor Emeritus of
Computational and Applied Mathematics

Robin C. Sickles
Professor of Economics

Nathaniel Dean
Associate Professor of Computational and
Applied Mathematics

Houston, Texas

May, 1999

Abstract

A Computational Study of Vehicle Routing Applications

by

Jennifer L. Rich

This thesis examines three specific routing applications. In the first model, the scheduling of home health care providers from their homes, to a set of patients, and then back to their respective homes, is performed both heuristically and optimally for very small instances. The problem is complicated by the presence of multiple depots, time windows, and the scheduling of lunch breaks. It is shown that the problem can be formulated as a mixed integer programming problem and, in very small instances, solved to optimality with a branch–and–cut procedure. To obtain solutions for larger instances, though, a heuristic is shown to have more success.

The second application considers the vehicle routing problem with time windows, or VRPTW. The vehicle routing problem involves finding a set of routes starting and ending at a single depot that together visit a set of customers. In the VRPTW, there is an additional constraint requiring that each customer must be visited within a given time window. The best known solution procedures for solving the VRPTW use a set partitioning model with column generation. Within this framework, we present a new approach for generating valid inequalities, specifically k–path cuts, to improve the linear programming relaxation. Computational results are given for the standard

library of test instances. In particular, the results include solutions for ten previously unsolved instances.

The final application concerns the less-than-truckload, or LTL, trucking industry. An LTL carrier primarily handles shipments that are significantly smaller than the size of a tractor–trailer. Savings are achieved by consolidating shipments into loads at regional terminals and transporting these loads from terminal to terminal. The strategic load plan determines how to route the flow of consolidated loads from origin terminals to destination terminals cost effectively and allowing for certain service standards. To find good solutions to this problem, we apply a dual-ascent procedure to a related uncapacitated network design problem to obtain computational results for three different companies.

Acknowledgments

First and foremost, I want to thank each of my committee members for their continuous support and valuable comments. In particular, I thank my thesis advisor, Bill Cook, for sharing with me his knowledge and experience. As an advisor, he gave me both the freedom and the guidance that I required. I am especially grateful for his confidence in me – there were many times that without it, I would undoubtedly have lost faith in myself.

In addition, I credit much of my success at Rice to the wonderful and caring faculty and staff of the CAAM department. I thank Bob Bixby, not only for his valuable role on my committee, but also for inspiring my interest in Operations Research in his linear programming class during my first semester at Rice. More recently, I have benefited from the classes of David Applegate and Bill Cook. I particularly want to thank Professor Applegate for his help in preparing my presentation for my dissertation defense. My deep appreciation also goes to Michael Pearlman for his immeasurable help throughout my research. My very special thanks go to John Dennis, as he is greatly responsible for much of my success in my first years here at Rice. I am indebted to him for his invaluable support and advice over the years.

As with many dissertations, this work could not have been possible without the collaboration of many individuals. I thank Eddie Cheng, Sanjeeb Dash, Bruce Hoppe, Cassandra McZeal, Erica Zimmer Klampfl, and (of course) my advisor Bill Cook for the separate and important roles that they each played in this research. Thank you also to the LTL carrier Southeastern Freight Lines, and in particular Phil Teague and Braxton Vick, for providing us with our primary data set for our work on the strategic load planning problem.

I am also grateful for the friends that I have made during my time at Rice, those in the CAAM department and others. Erica, Maeve, Chao, Olena, Cassandra, Donald, Bill, Keith, Kevin, Colin, Jay and Deba are just a few who have been absolutely wonderful officemates, housemates, classmates, and friends. Cassandra, thank you not only for your valuable friendship, but also for your patience, helpful advice, and the many research–related discussions that we've had. To all of my friends — thank you for always being there for me.

Finally, I thank my family and Purvez for always believing in me. Mom, Dad, and Charlie, I am very lucky to be part of such a wonderful family that never sets limits on what each of us can accomplish! Purvez, I am truly grateful for all of the love, support, encouragement, and advice that you've given me while working on this dissertation. I can not imagine having done this without you!

Contents

Illustrations

Tables

Chapter 1

Introduction

Since routing decisions must be made in almost every industry that involves the transportation of goods or services, improved vehicle routing methods represent a significant savings potential to businesses throughout the world. In the U. S. economy alone, the importance of such distribution problems was established in a survey by Kearney [44] that estimates the annual distribution costs even in 1980 at $400 billion. These routing problems, though, are quite complex and frequently cannot be solved to optimality, given the current state of technology. Still, small improvements in the distribution network of large companies can yield significant savings.

The research presented in this thesis improves upon known methods for solving, either optimally or heuristically, three types of routing applications. One application from the American home health care industry requires a set of routes that schedule a nurse from home, to a set of patients, and then back home, subject to certain feasibility constraints. A randomized greedy algorithm in conjunction with insertion and exchange heuristics succeeds in finding relatively good solutions to this problem. In addition, an exact solution method tests the effectiveness of several types of cutting planes. In particular, a known algorithm for finding 2–path cutting planes in the vehicle routing problem with time windows is improved upon and modified for this specific home health care problem. A separate application applies the improvement made to the separation algorithm for 2–path cuts to the original vehicle routing problem with time windows. An extension of this new algorithm to k–path cuts where k is greater than two is successfully applied, as well. Finally, the thesis also examines an application from the less–than–truckload trucking industry. A heuristic procedure using an uncapacitated network design problem that is iteratively modified to approximate the true problem provides improved networks for data sets from three different trucking carriers.

This thesis establishes in chapter 2 some standard notation and terminology used to discuss various solution approaches in the vehicle routing field. It then gives a brief overview of some of the applications and work that has been done in the area of vehicle routing. Following this review, chapter 4 introduces a new model for an application of vehicle routing in the American home health care industry and considers some completed work on the model. In chapter 5, traditional vehicle routing problems with time windows are solved using a new algorithm for finding violated k–path cuts. In chapter 6, a routing problem from the less–than–truckload trucking industry is defined and an effective solution method is presented along with results from real–world data.

Chapter 2

Notation and Terminology

When applying mathematics to real–world applications, it is common to want to find the best solution to a problem given certain restrictions. Oftentimes, the problem is stated in terms of optimizing an objective function subject to a set of constraints. In many cases, the objective corresponds to minimizing cost, or, alternately, to maximizing output or profit. The constraints may refer to problem parameters such as production capability, labor restrictions and limited resources. If the objective function and the constraints are linear, then the problem can be formed as a *linear programming problem (LP)*:

$$\begin{aligned} \text{Minimize} \quad & Z = \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad (i = 1. \dots, m) \\ & l_j \leq x_j \leq u_j \qquad (j = 1, \dots, n) \end{aligned}$$

The vectors c, b, l, and u and the matrix A, made up of the a_{ij} terms, are constants in the formulation. The constraints may contain linear equalities as well as linear inequalities. If there is the additional requirement that some of the x_j variables must be integer, then the problem is called a *mixed integer linear programming problem (MIP)*. If all of the variables must be integer, then the problem is a pure *integer linear programming problem (IP)*.

Optimization problems, such as LP's, MIP's, and IP's, can be phrased as decision problems (that is, problems having a yes/no solution) by asking the question: does there exist a solution with value at most k? As a means of measuring the level of difficulty involved in solving an optimization problem, the corresponding decision problem (and thus, indirectly, the optimization problem) is often classified as $\mathcal{P}, \mathcal{NP}, \mathcal{NP}$–hard and/or $\mathcal{NP}$–complete. The complexity class $\mathcal{P}$ is the set of decision problems that are polynomial–time solvable. Problems in this class are generally considered tractable since the running time for instances of the problem can be bounded by some polynomial in terms of the size of the instance.

A decision problem is in the class $\mathcal{NP}$ if it can be *verified* by a polynomial–time algorithm. In terms of our optimization problem, this means that given a potential solution for any instance of our problem, a polynomial time algorithm can determine if that potential solution is feasible. Additionally, a problem A is $\mathcal{NP}$–complete if it is in $\mathcal{NP}$ and any instance of any other problem contained in $\mathcal{NP}$, say B, can be reduced in polynomial time to an instance of problem A. If the problem A satisfies the latter property, but is not necessarily in $\mathcal{NP}$, then the problem is said to be $\mathcal{NP}$–hard. Since the existence of a polynomial–time algorithm for some $\mathcal{NP}$–complete problem implies the existence of a polynomial–time algorithm for all $\mathcal{NP}$–complete problems, $\mathcal{NP}$–complete and $\mathcal{NP}$–hard problems are generally considered quite difficult to solve. This evidence that a decision problem corresponding to an optimization problem is hard thus provides evidence that the optimization problem is hard, as well. The reader is directed to Garey and Johnson [37] for a more detailed explanation of $\mathcal{NP}$–completeness and related issues.

A straightforward lower bound on the value of a MIP or IP can always be obtained by looking at the problem's LP *relaxation*. This is the linear programming problem obtained by removing all of the integer restrictions on the variables. Clearly, the LP relaxation contains all of the feasible points in the mixed integer or integer problem, as well as possibly fractional solutions. Thus, any feasible solution to the MIP or IP cannot have a lower objective function value than the optimal solution to the LP relaxation.

A common solution method for solving integer and mixed integer linear programming problems is an algorithm referred to as *branch and bound.* This is an enumerative, search tree approach that uses some lower bounding procedure in an attempt to prevent the enumeration of all possible solutions. In general terms, this algorithm "branches" by taking a problem and separating it into 2 or more subproblems. Throughout the method, the best known upper and lower bound values are maintained. At each stage of the algorithm, a problem $\mathcal{S}_i$ is selected from some set $\mathcal{S}$ of subproblems ($\mathcal{S}$ initially contains only the original problem). The subproblems contained in $\mathcal{S}$ are often referred to as the set of *active nodes* of the branch–and–bound tree. A lower bound is determined for $\mathcal{S}_i$. Solving an LP relaxation is a common means of acquiring this bound. If this lower bound is greater than or equal to the upper bound, then the problem is discarded. If an integer feasible solution that is lower than the current best upper bound is generated, then the best upper bound is updated accordingly. If the upper bound is new, then any problem in $\mathcal{S}$ with a lower bound greater than or equal to this new bound is deleted. If the current problem is not discarded, then the problem $\mathcal{S}_i$ is branched on and the generated subproblems are placed in the set $\mathcal{S}$. The next step in the algorithm is to start again by selecting another node from the set $\mathcal{S}$. The algorithm terminates when there are no more problems contained in the set $\mathcal{S}$.

The *branch–and–cut* method is a variation of the branch–and–bound algorithm using LP relaxations to calculate the lower bounds. In this method, additional *valid inequalities* are added to the polyhedral description of the solution space to improve the lower bounds. A violated valid inequality, also called a *cutting plane*, is a constraint that tightens the LP relaxation by cutting off a fractional solution that is infeasible for the MIP or IP, but would otherwise be feasible in the relaxation. By generating cutting planes in the initial formulation (that is, the root node of the search tree) and optionally at each node of the branch–and–bound tree, the lower bounds may become better, allowing for more nodes of the tree to be deleted sooner, thus reaching a solution faster. For example, in Figure 2.1 the feasible region of an LP relaxation is shown here as the polytope outlined in the heavy black lines. The gray dots refer to the integer–feasible solutions and the LP optimal solution is circled in a dash–dot line. Clearly, both lines, L_1 and L_2, represent valid inequalities since neither one eliminates any integer–feasible points. The L_2 line, however, is a cutting plane since it is valid and it cuts off a fractional solution. Adding the linear inequality corresponding to line L_2 to the LP relaxation will yield a new optimal solution. In the best case, the new solution will be integer, as is shown in the example. In many cases, the addition of cutting planes will succeed in reducing the size of the feasible region, but will not generate an integer solution. The smaller region, though, will hopefully produce a smaller branching tree.

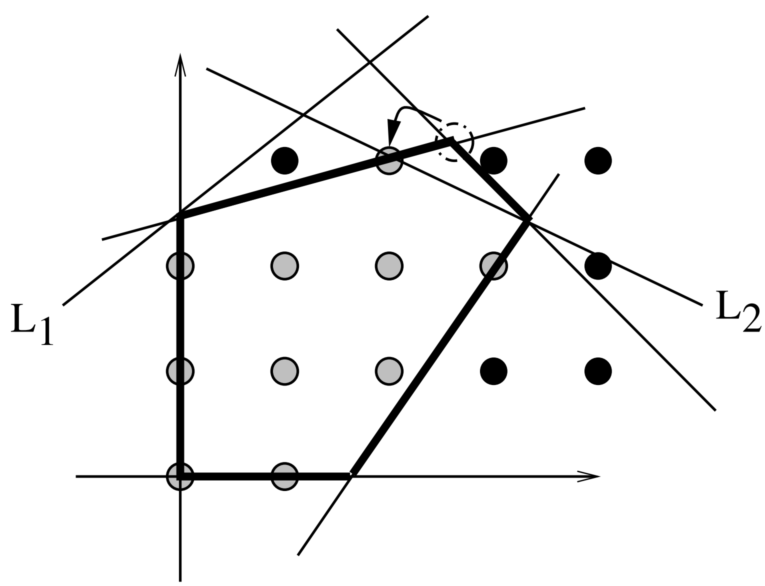

Figure 2.1 Using valid inequalites to tighten the LP formulation.

Chapter 3

Existing Literature on the Vehicle Routing Problem

The *vehicle routing problem*, or VRP, in its most basic form is the problem of routing a fleet of vehicles to a set of clients. The clients have known demands, each vehicle may have a capacity, and the routes must start and end at a depot. The routing must be done with respect to minimizing some value, typically the total distance traveled, the total number of vehicles, or some combination of the two. Side constraints may limit the number of available vehicles or the number of clients permissible on any single route. Other variations of the problem impose time window restrictions or contain multiple depots. The VRP and most of its variants are known to be $\mathcal{NP}$–hard problems. For example, the well–known *traveling salesman problem*, or TSP, is an $\mathcal{NP}$–complete problem which can be classified as a VRP. It is a one depot, one vehicle, non–capacitated vehicle routing problem. While the TSP has been an area of interest for researchers for many decades, study of the VRP began its rapid expansion only about 15 years ago. Much of the motivation for studying the VRP has come from the numerous real–world applications and the considerable savings that better solutions to these problems represent.

3.1 The TSP and m–TSP

Given a set of cities represented by nodes in a graph, the traveling salesman problem seeks the least cost route which visits each node once. The *multiple traveling salesman problem*, or m–TSP, is defined as finding a *set* of routes originating and terminating at a single node which visits each node once. As in the case of the TSP, the m–TSP can also be classified as a non–capacitated vehicle routing problem. It can be defined on the directed graph $G = (V, A)$ where V is a set of n nodes and A is the set of arcs. If the problem is symmetric, in the sense that going from point x to point y costs the same as going from y to x, then the set A of arcs may be replaced by the set E of undirected edges. Let $V = \{0, 1, \ldots, n-1\}$ where the depot is assumed to be node 0. The number of salesmen, m, can be bounded by a lower and upper bound: $\underline{m} \leq m \leq \overline{m}$. In most of the m–TSP's considered in the literature, the value of m is *fixed* (that is, $\underline{m} = \overline{m}$). This means that the solution must contain exactly m routes. In other cases of the m–TSP, the value of m is *free* (that is, $1 \leq m \leq n-1$), in which case, m is a variable of the problem.

It is useful to see the exact integer programming formulation of the m–TSP. First, some additional notation must be defined. Let S be a subset of nodes in the graph and $V \backslash S$ be the nodes in V excluding those in the set S. Then, $\gamma(S)$ is defined to be the set of edges having both endpoints in the set S. In other terms, for $S \subseteq V$, $\gamma(S) = \{(i, j) \in E : i, j \in S\}$. The variable x_{ij} is a 0–1 variable that indicates whether the edge (i, j) appears ($x_{ij} = 1$) or not ($x_{ij} = 0$) in the solution. When considering the x values of a set of edges T, we use the notation $x(T)$ to mean $\sum_{(i,j)\in T} x_{ij}$. These definitions are used throughout this thesis. An integer linear programming formulation of the asymmetric m–TSP can be written as follows.

$$\text{Minimize} \quad \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{3.1}$$

$$\text{subject to} \quad \sum_{j=1}^{n-1} (x_{0j} + x_{j0}) = 2m \tag{3.2}$$

$$\sum_{i \neq j} x_{ij} = 1 \qquad (j = 1, \ldots, n-1) \tag{3.3}$$

$$\sum_{i \neq j} x_{ji} = 1 \qquad (j = 1, \ldots, n-1) \tag{3.4}$$

$$x(\gamma(S)) \leq |S| - 1 \qquad (S \subseteq \{1, \ldots, n-1\}) \tag{3.5}$$

$$x_{ij} = 0 \text{ or } 1 \qquad ((i,j) \in A) \tag{3.6}$$

$$\underline{m} \leq m \leq \overline{m} \qquad m \text{ integer.} \tag{3.7}$$

So, if m is fixed, it can be removed from the formulation entirely. If $m = 1$, then the equivalent TSP (for an asymmetric graph) can be written with constraint (3.2) combined with constraints (3.3) and (3.4). Constraints (3.5) are called *subtour elimination constraints* because they forbid subtours — that is, cycles that do not include the depot. These constraints make the formulation quite large, since the number of subsets S grows exponentially with the size of V. Constraints (3.3), (3.4) and (3.6) actually represent another well studied problem known as the *assignment problem*, or AP.

Compared to the quantity of literature that can be found on the TSP, relatively few papers have been written on the m–TSP for either the symmetric or asymmetric case. This is due largely to the fact that several transformations exist which can convert any m–TSP into an equivalent 1–TSP. All of these transformations, though, introduce at least $\overline{m} - 1$ artificial nodes (see for example [6, 56, 64]). If m is free, this doubles the number of nodes in the problem. Laporte, Nobert, and Nga [55] observe that there is a known special case where finding an optimal solution to the m–TSP is equivalent to solving a standard TSP on the same graph. If the triangle inequality condition, which says that $c_{ij} + c_{jk} \geq c_{ik}$ for all i, j, k, holds, then there exists an optimal solution to the m–TSP such that the value of m equals the lower bound $\underline{m}$. Thus, if m is free and the triangle inequality holds, then the m–TSP reduces to a 1–TSP (or TSP) on the same graph. Appendix A includes a new transformation of the symmetric m–TSP in which m is free and the triangle inequality does not hold.

This transformation does not increase the number of nodes in the graph, and the solution to the TSP on the new graph yields the solution to the m–TSP.

Svestka and Huckfeldt [64] propose a solution strategy for solving the asymmetric m–TSP where m is fixed. Their method first transforms the problem into a 1–TSP and then solves a series of assignment problems embedded in a branch–and–bound scheme. Subtours are eliminated by a branching strategy developed by Bellmore and Malone [7]. Their method also uses a heuristic tour generator at the first node of the branch–and–bound tree. With this algorithm, problems become easier to solve the more salesman there are. They are able to solve randomly generated instances ranging from 10 to 60 nodes.

Another algorithm for the asymmetric case is given by Laporte, Nobert, and Nga [55]. They do not transform the problem into a 1–TSP; rather they reformulate the m–TSP, so that removing the subtour elimination constraints yields a standard maximum cost flow problem. Their method eliminates illegal subtours by using a branch–and–bound algorithm that includes and excludes arcs of a maximum cost flow problem at each iteration. They also develop a heuristic which they use inside the search tree to generate feasible routes from a graph containing subtours. While their algorithm allows for the possibility for m to be a variable, they only test their method with m being fixed in the data sets. They solve problem instances with the number of cities ranging from 80 to 160. The cost matrices in all cases were randomly generated from a uniform distribution and then rounded to the nearest integer.

For either the symmetric or asymmetric case of the m–TSP, Laporte and Nobert [52] implement a pure cutting plane approach to solving the problem. They do not require that the value of m be fixed. They test their method on non–Euclidean problem instances which are generated from a uniform distribution, and on Euclidean problem instances which are obtained by generating the (x, y) coordinate points from a uniform distribution. Using a simplex solver written in FORTRAN, the authors find solutions for symmetric, non–Euclidean instances of the m–TSP with up to 100 nodes by using subtour elimination cuts and Gomory cuts. The authors used the Gomory cuts as a means of attaining integrality; thus, no branch–and–bound algorithm is implemented.

3.2 The VRP

The VRP can be thought of as an m–TSP with additional side constraints. As in the m–TSP, the VRP typically requires the least–cost set of routes such that the number of vehicles, m, is between $\underline{m}$ and $\overline{m}$; each node in $V\backslash\{0\}$ is visited once by a single vehicle; and all routes start and end at the depot (node 0). The most common side constraint is a limit on the capacity of each vehicle. If each non–depot node represents a client with a known demand, then a vehicle may only visit a set of clients on a given route if the total demand of the set of clients is less than or equal to the vehicle's capacity. This is referred to as a *capacitated vehicle routing problem*, or CVRP. In most cases, a standard VRP is considered to have capacities. If there are actually none, then the capacities can simply be set to $+\infty$ so that any capacity restrictions are not binding. Many other side constraints have been studied as well. There are several very good survey articles that have been written about the vehicle routing problem. For example, the surveys of Bodin and Golden [8], Laporte and Nobert [51], Assad [3], Laporte [49] and Fisher [32] describe known formulations, heuristics, and exact solution methods for the VRP.

3.2.1 Exact Algorithms for the VRP

In a survey article written by Laporte [49], the author identifies three broad categories of exact algorithms for the VRP: direct tree search methods, dynamic programming, and integer linear programming. The numerous proposed algorithms for the VRP generally fit into one of these three areas. We review a few examples to illustrate the variety of exact algorithms within this categorization. For a more detailed survey, see Laporte [49] and Laporte and Nobert [51].

Direct tree search methods typically embed a non–LP based lower bounding procedure within a branch–and–bound scheme. One such method is the k–degree center tree and related algorithm. Developed by Christofides, Mingozzi and Toth [17], a k–degree center tree is a tree (that is, a subset of $n-1$ edges, T, such that T is a single connected component containing no cycles) where the degree of the depot node is k. The algorithm assumes that m is fixed and works best on small problems. The lower bound from the k–degree center tree is embedded in a branch–and–bound scheme which solves problems taken from the literature ranging from 10 to 25 nodes. An apparently better variation on this idea, is given in Fisher [34]. In his paper, the bound is based on finding a minimum K–tree, where K is the number of vehicles (m) and a K–tree is defined to be a set of $n+K-1$ edges that span the graph. Fisher adds the additional constraint that routes containing a single customer are not allowed. Thus, to obtain a good bound for the VRP, the optimum K–tree with $2K$ edges adjacent to the depot is needed. This bound appears to be much stronger and enables the algorithm to solve some 100 customer problems taken from the literature and from real vehicle routing applications. Another recent example is Fishetti, Toth, and Vigo's [31] branch–and–bound algorithm which utilizes two new bounding procedures at each node of the branch–and–bound search tree. The first procedure comes from solving assignment problems in a disjunctive lower bounding procedure involving infeasible arc subsets. The second lower bounding procedure uses a minimum cost flow relaxation and projection. Although this algorithm requires that m

be fixed, the authors are able to solve randomly generated problems with as many as 300 customers and 8 vehicles.

In his survey article, Laporte states that dynamic programming was first proposed as a method of solving VRPs in 1971 by Eilon, Watson–Gandy, and Christofides [30]. This method assumes that the number of vehicles (or routes) is fixed. Efficient use of dynamic programming requires a substantial reduction of the number of states. State–space relaxation, introduced by Christofides, Mingozzi and Toth [18] in 1981, seems to be the most successful means of using dynamic programming to generate lower bounds. Christofides [15] reports using this procedure to solve CVRPs with up to 50 nodes.

Many methods have been proposed that use integer linear programming, three of which merit discussion in this review. The first combines a set partitioning model with a column generation approach. The set partitioning model considers the set R of all feasible routes r and defines a_{ir} to be a 0 or 1 coefficient that takes the value 1 if and only if city i is on route r. The cost of a route is given by c_r^*. Thus, the model can be written as:

$$\text{Minimize} \quad \sum_{r \in R} c_r^* x_r \tag{3.8}$$

$$\text{subject to} \quad \sum_{r \in R} a_{ir} x_r = 1 \qquad i \in V \backslash \{0\} \tag{3.9}$$

$$x_r = 0 \text{ or } 1 \qquad \forall r \in R. \tag{3.10}$$

The difficulty with this formulation, however, is that the number of feasible routes is generally too large to be manageable. Thus, these routes, also referred to as columns since they imply the column $a_{\cdot r}$ in the formulation, are generated only as needed. The method used to generate feasible routes depends on the model and the algorithm being implemented. Various authors have proposed column generation methods to solve the

VRP incorporating different cutting plane and branch–and–bound approaches (see [27, 51]).

The second IP approach is the two–index vehicle flow formulation. This formulation uses a flow variable x indexed on two variables, i and j, to indicate whether or not a vehicle travels along the arc from node i to node j. The m–TSP formulation given in an earlier section is a two–index formulation. This model is used by Laporte, Nobert and Desrochers [54] in a branch–and–cut algorithm which uses the subtour elimination constraints as the cutting planes. Their algorithm appears to perform better on loosely constrained problem instances and solves instances with up to 60 nodes.

The third IP approach uses the three–index vehicle flow formulation. In this case, the flow variable x is indexed on three variables, k, i and j where k is a vehicle and i and j are nodes. So the binary variable x_{kij} indicates whether or not a vehicle k travels from node i to node j. Fisher and Jaikumar [35] use this method on the CVRP. They decompose the problem into two well known problems, the generalized assignment problem, or GAP, and the TSP. The method then iterates between solving a GAP master problem and a TSP subproblem. One advantage of their algorithm is that it produces a feasible solution even if it is not run to optimality. In fact, the authors did not run it to completion, but reported heuristic results for randomly generated instances with 50–199 nodes.

3.2.2 Heuristic Methods for the VRP

While there are many exact algorithms for the VRP in the literature, it is still more common for real–world applications to rely on heuristics for good solutions. There are numerous heuristics that have been used for the VRP, many of which have their roots in the TSP literature. Heuristics have proven to be quite adaptable to different problem variations, as well as being considerably faster than known exact techniques.

The savings/insertion approach was pioneered by Clarke and Wright [20] in 1964. This procedure adds nodes with the greatest savings to a tour. Improvement or exchange procedures such as the well–known branch exchange heuristic were developed by Lin [58] and Lin and Kernighan [59] for the traveling salesman problem and have been extended to the VRP by many authors, such as Christofides and Eilon [16]. This heuristic starts with a feasible solution and repeatedly alters the solution, each time improving the overall objective value and maintaining feasibility, until no more improvements can be found. In the mid–1970s, two heuristics based on very simple ideas were developed. The cluster first–route second method involves first clustering the nodes and/or arcs and then determining the most economical route for each grouping. The route first–cluster second method approaches the problem in the opposite direction. In this case, a large, usually infeasible route is created, and then partitioned into a number of smaller, feasible tours.

In the 1990s, the latest heuristics use meta–heuristics such as tabu search, genetic algorithms, and simulated annealing to generate high quality solutions for the VRP (see for example [11, 38]). The tabu search method performs a series of alterations to a solution, allowing the quality of the solution to sometimes deteriorate. A list of recent changes is maintained and kept as a tabu list, so that the algorithm does not keep repeating the same set of changes over and over again. A genetic search combines the best elements from a population of solutions to form a new population of solutions with potentially better objective values. Simulated annealing is an improvement procedure which also accepts with certain probability feasible solutions which increase the value of the objective function. These methods achieve improved performance from the introduction of randomness and/or the allowance within improvement procedures for some decline in solution quality in the hope that an overall better solution will be found further along in the process. In other words, these procedures allow the solution search to move away from locally optimal solutions so that a globally optimal solution may be discovered.

3.3 Time Window Problems

The *vehicle routing problem with time windows*, or VRPTW, refers to an $\mathcal{NP}$–hard vehicle routing problem in which the customers require vehicles to provide service within a specified time period, or window. As with any vehicle routing problem, service can refer to the pickup or delivery of goods or to any other task being modeled. In some cases, it is possible to have more than one time window for a single client; in most, though, the model assumes that there is no more than one, or exactly one, time window per client. A time window is typically given as $[a_i, b_i]$ for a given node i referring to client i. The value a_i is the earliest time that service may begin at the client; b_i is the latest time that service may begin. If $a_i = b_i$ for all nodes i, then the start times are fixed and the problem is referred to as a vehicle scheduling problem. If the difference between a_i and b_i is small, relative to the time horizon of the problem, then the time window is said to be tight. Depending on the problem specifications, the depot may or may not have a time window as well. If the depot does have a time window, typically this window $[a_0, b_0]$ will define a_0 as the earliest time a vehicle may leave the depot and b_0 as the latest time a vehicle may return to the depot.

Very little work had been done on solving VRPTWs prior to 1986. Since then, many of the exact and approximate methods developed for the VRP have been modified to include the more constrained time windows problem. For example, Desrochers, Desrosiers, and Solomon [25] use the set partitioning model with column generation to solve 100 customer instances of the VRPTW. These randomly generated test sets known as the Solomon instances are established benchmark problems from the literature on the VRPTW. Their method adds feasible columns as needed by using dynamic programming to solve shortest path problems with time windows and capacity constraints. They then use a branch–and–bound algorithm to solve the integer set partitioning problem. A more recent optimization method developed by Kohl, Desrosiers, Madsen, Solomon, and Soumis [46] apply a similar algorithm. They use a branch–and–cut algorithm, rather than branch–and–bound, with a new cutting plane based on a strengthened form of subtour elimination inequalities called *k–path cuts*. These cuts use the added information about vehicle capacities and time windows to tighten the inequality. Using this algorithm, they solve 100 customer instances of the problem faster than do other methods and solve several previously unsolved problems, as well. Heuristics for the VRP have been similarly adapted to the VRPTW.

3.4 Multiple Depot Problems

The *multiple depot vehicle routing problem*, or MDVRP, generalizes the VRP by permitting the vehicles in the model to be located at more than one depot. The objective of the problem is to design a set of routes using the vehicles located at each depot so that all of the customers are visited exactly once, each vehicle departs and returns to the same depot, and the total distance traveled by the fleet is minimized. Typically, there are also customer demands to be met and capacity restrictions on the vehicles. The number of vehicles stationed at each depot may be fixed or unbounded. If the number is fixed, then merely checking the feasibility of a model can be difficult; however, if there are an unlimited number of vehicles, then there exists a trivial feasible solution in which each customer is visited by a single vehicle.

Only a handful of papers have been written about the MDVRP (see, for example, [12, 14, 50, 53, 57]). We know of only two [53, 57] that present exact solution methods for the problem. In both of these papers, the difficulty of the problem increases with the number of depots. In 1984 Laporte, Nobert, and Arpin [53] developed a branch–and–bound algorithm that solves relaxed subproblems, adds upper bounds on variables, and branches on non–integer variables. They were able to solve some randomly generated symmetric instances of the MDVRP with as many as 25 nodes (including depots). In 1988, Laporte, Nobert, and Taillefer [57] solved some asymmetric multi–depot problems by performing a graph extension and then creating constrained assignment problems which they were able to solve through branch–and–bound. Using this method, they solved problem instances with up to 80 nodes, so long as the number of depots was small (2 or 3 depots).

While still few in number, more papers have been written concerning heuristic approaches to the MDVRP. A 1993 paper by Chao, Golden, and Wasil [14] provides a review of the previous heuristics in the operations research literature, and introduces a new heuristic, as well. The most important element in this new heuristic is the improvement procedure which allows total distance to increase with the hope that a solution with an overall decrease may be found further along in the improvement process. The authors apply their new heuristic on data sets taken from the literature and find that the new heuristic yields better solutions than were previously known.

Chapter 4

A Home Health Care Problem

The home health care industry in the United States is a growing service industry that must perform scheduling and/or routing tasks on a regular basis. A company came to us looking to improve the efficiency of home health care operations. They envision one company providing service for a large area and employing many health care providers with whom it can communicate remotely. By using technology to relay schedule information and patient reports, companies eliminate the need for employees to travel to a central office every day. Much of this vision is apparent in home health care companies today, though the service region is typically segmented into several small local problems which are then solved by hand. While this method may provide working solutions, the potential for savings involved in finding a better solution over a larger service area is considerable.

The following problem is posed in the context of a home health care problem. Much of the work done on this model is the result of a joint effort by the author and Dr. Eddie Cheng. Despite the fact that in this industry health care is provided by many qualified individuals, such as registered nurses, physical therapists and home health aides, for simplicity, in this thesis all employees are referred to as nurses. The only differentiation made is between salaried workers (full–time nurses) and non–salaried workers (part–time nurses). Salaried workers are paid for a full–time shift everyday, whether or not they are scheduled to work the entire time. They are paid overtime if they are required to work for longer than the standard shift length. Part–time nurses are paid by the hour. The differences in the nurses' qualifications are represented by a known binary relationship with each patient; a nurse–patient pair is thus, either a feasible match or an infeasible match. Accordingly, the nurse may be scheduled to visit a given patient, or not. Additionally, a company in this industry would like to satisfy not only a customer's need for health care, but also the customer's happiness by providing dependable service (that is, providing health care when the customer requests it). Thus, many home health care companies allow the customer to specify a time window during which he or she will be at home awaiting the requested care. In summary, the problem is to find an optimal schedule such that each nurse that is scheduled to work leaves from his or her home, visits a set of "feasible" patients within their time windows, takes a lunch break within the nurse's lunch time window, and returns home, all within the nurse's time window (the times during which the nurse may work) and within the known limit on the length of a shift. The optimal schedule minimizes cost by reducing the amount of overtime and part–time worked.

4.1 Previous work

This type of problem is not unique to the home health care industry. One example of the same type of problem is what Caseau and Koppstein [13] called the task/technician assignment problem. This problem involves assigning repairmen to sets of tasks. Each technician has a specified starting and ending location. The task must be executed within a time window and the technician must have the appropriate skills to perform the task. The goal is to maximize the amount of work by minimizing the amount of travel needed. The authors describe finding solutions with an expert system by dividing the problem into 2 subproblems: the matching of tasks to technicians and the scheduling of each technician's tour. This problem is essentially the same structure as the home health care problem. One difference is that in their problem each repairman–job pair has a match score rather than a binary relationship;

in this way, their problem permits less qualified matches. The authors' decision to approach this problem heuristically was based on their having many different levels on which to judge the quality of any solution, making it difficult to determine the meaning of optimal.

The home health care problem is, more generally, a vehicle routing problem with time windows, many depots, and compatibility information. It has many depots since each nurse's home acts as a depot housing a single vehicle (that is, a nurse) where a route must start and end. Although there is a significant amount of literature involving vehicle routing problems with time windows, most of the problems considered have a central depot, rather than multiple depots. Similarly, multiple depot problems in the literature tend not to consider time window constraints.

4.2 MIP Formulation

A difficulty in formulating the problem relates to the inclusion of lunch breaks in the model. The author has found only one reference in the literature to such constraints. Sansó, Desrochers, Desrosiers, Dumas, and Soumis [62] model lunch breaks as a special feature within a column generation approach. Their model is less flexible than the one proposed here since all breaks in their model had to be taken within the same time window and last the same amount of time. They then incorporate these conditions into the column generation procedure, rather than into a direct MIP model. In this thesis, two mixed integer linear programming problem formulations are developed, both of which incorporate lunch breaks into the model. One model uses the two–index vehicle flow representation, while the other uses the three–index vehicle flow representation. The lunch breaks are handled the same way in both cases – a lunch node is created for each nurse, made feasible for only the corresponding nurse, and given the appropriate time window during which the beginning of lunch must be scheduled. After testing both formulations, the three–index vehicle flow model proved to be better suited to our methods. This formulation tends to have more integer variables, fewer continuous variables, and far fewer constraints. Since this is the model to which all computational results refer, the three–index formulation of the problem is presented here. The two–index formulation of the model is provided in appendix B.

4.2.1 Notation

Graph theory terminology used in this chapter follows that of Cook, Cunningham, Pulleyblank and Schrijver [21]. The home health care problem can be defined on a simple, directed graph $G = (V, A)$. The set of nodes, V, consists of three disjoint sets: (1) a set N of nurses (or health care providers); (2) a set P of patients; and (3) a set L of lunches. The set of nurses is made up of the disjoint union of full-time nurses, F, and part-time nurses, H. Each nurse i has a time window $[a_i, b_i]$ in which to begin and end a shift. (Throughout this paper, times are recorded in minutes, where the start of the day is 0 and the end of the day is 1440.) Full-time nurses are salaried workers who get paid for some fixed length of time each day, regardless of whether or not they are scheduled to work. These workers only incur an additional cost (α) per amount of overtime that they are scheduled to work (overtime is taken to be the time past the standard shift length, SL). Part-time nurses incur a cost (β) proportional to the total time that they work. A maximum shift length, MSL, limits the total time any nurse may work. Each patient in the set P has a duration, d_i, which indicates the time it takes to provide health care at that patient, and a time window, $[a_i, b_i]$, where a_i is the earliest time service may begin at patient i and b_i is the latest time service may begin at patient i. Included in the set P are the original patients of the problem plus a dummy patient for each nurse, defined to be feasible only for its corresponding nurse. The reason for these dummy patients will become clear in the discussion of the formulation. The dummy patient corresponding to some nurse n is denoted by $DP(n)$. Each dummy patient's time window is defined to match the time window of its corresponding nurse. The set of lunches consists of a unique lunch node for each nurse, where each nurse's lunch has a duration and a time window. The lunch corresponding to some nurse n can be identified using the notation $L(n)$.

In order to be able to define the set of arcs for the graph, we must first consider the time window notation. Let t_{ij} denote the duration of service at node i plus the time it takes to travel from i to j for every $i, j \in V$. If i is a nurse node, then the duration of service at i is defined to be zero and t_{ij} represents only travel time. If either i or j is a lunch node, then define the value t_{ij} to be zero since any travel involved in a lunch break is considered to be part of the break and therefore must take place within the known duration of the lunch (denoted $d_{L(n)}$). An ordered pair of nodes (i, j) are said to be time window compatible if $a_i + t_{ij} \leq b_j$.

Thus, the set of arcs, A, is based on a notion of time window compatibility and feasibility. The feasibility restrictions exist as a binary relationship between each nurse and every other node. In other words, each nurse is either allowed or disallowed to visit a node in V. This known relationship is entirely contained in the set of feasibility lists for each nurse:

$$B_n = \{n\} \cup \{L(n)\} \cup \{j \in P : \text{nurse } n \text{ may visit patient } j\}, \forall n \in N.$$

By definition, there is only one lunch node and one dummy patient node that each nurse is permitted to visit. Given the feasibility lists, an ordered triple (n, i, j) is defined to be *feasible* if n is a nurse node and i and j are both nodes which n is permitted to visit and the ordered pair (i, j) is time window compatible. Thus, the set A of arcs can be defined over the set of feasible triples (n, i, j) in the graph. A given arc (n,i,j) starts at node i and ends at node j. Thus, there may be more than one arc connecting a pair of nodes. Note that the graph is not permitted to contain any loops (that is, no arcs that start and end at the same node) nor does it contain arcs between two nurse nodes or between two lunch nodes.

Several types of variables are needed to describe the model. Let x_{nij} be a binary variable defined on the set of arcs A. The interpretation of x_{nij} is that $x_{nij} = 1$ if nurse n travels from node i to node j and $x_{nij} = 0$ otherwise. The continuous variable y_i indicates the starting time at node i and is defined for all nodes in the graph. The variable z_n is defined for $n \in N$ and refers to the ending time of nurse n's shift. The variable s_n is defined only for $n \in F$ and represents the slack in a full–time nurse's shift.

For a set S of vertices in our directed graph, we define the following notation. The set $\delta(S)$ refers to the set of edges (i,j) such that $i \in S$ and $j \notin S$ and the set $\rho(S)$ is the set of edges (i,j) such that $i \notin S$ and $j \in S$.

4.2.2 Triple–Indexed Formulation

The mixed integer programming problem defined on this graph is as follows:

$$\text{Minimize } \alpha \sum_{n \in F} (z_n - y_n + s_n - d_{L(n)} - SL) + \beta \sum_{n \in H} (z_n - y_n) \tag{4.1}$$

subject to

$$x(\rho(L(n))) = 1, \quad n \in F \tag{4.2}$$

$$x(\rho(L(n))) = (1 - x_{n,DP(n),n}), \quad n \in H \tag{4.3}$$

$$x(\delta(n)) - x_{nnL(n)} = x(\rho(n)) = 1, \quad n \in N \tag{4.4}$$

$$x(\delta(i)) - \sum_{n \in N} x_{niL(n)} = x(\rho(i)) = 1, \quad i \in P \tag{4.5}$$

$$\sum_{j \in B_n} x_{nji} = \sum_{j \in B_n} x_{nij}, \quad n \in N, i \in P \cap B_n \tag{4.6}$$

$$x_{niL(n)} - x_{nni} - \sum_{j \in P \cap B_n} x_{nji} \leq 0, \quad n \in N, i \in P \cap B_n \tag{4.7}$$

$$y_i + t_{ij} - y_j \leq (b_i + t_{ij} - a_j)(1 - \sum_{n \in N, (n,i,j) \in A} x_{nij}), \quad i, j \in P, i \neq j \tag{4.8}$$

$$d_{L(n)} x_{nnL(n)} + y_i + t_{in} - z_n \leq (d_{L(n)} + b_i + t_{in} - a_n)(1 - x_{nin}),$$
$$n \in N, i \in P \cap B_n \tag{4.9}$$

$$y_{L(n)} + d_{L(n)} - y_i \leq (b_{L(n)} + d_{L(n)} - a_i)(1 - x_{niL(n)}),$$
$$n \in N, i \in P \cap B_n \tag{4.10}$$

$$y_{L(n)} + d_{L(n)} - z_n \leq (b_{L(n)} + d_{L(n)} - a_n)(1 - x_{nnL(n)}), \quad n \in N \tag{4.11}$$

$$y_n - y_{L(n)} \leq (b_n - \frac{1}{2} a_{L(n)})(2 - x_{nnDP(n)} - x_{nDP(n)L(n)}), \quad n \in N \tag{4.12}$$

$$y_i + t_{ij} - y_{L(n)} \leq (b_i + t_{ij} - \frac{1}{2} a_{L(n)})(2 - x_{nij} - x_{njL(n)}),$$
$$n \in N, i, j \in P \cap B_n, i \neq j \tag{4.13}$$

$$y_i + t_{in} - y_{L(n)} \leq (b_i + t_{in} - \frac{1}{2} a_{L(n)})(2 - x_{nin} - x_{nnL(n)}),$$
$$n \in N, i \in P \cap B_n, \tag{4.14}$$

$$y_n \leq y_{DP(n)}, \quad n \in N \tag{4.15}$$

$$y_n + d_{L(n)} \leq z_n, \quad n \in F, \tag{4.16}$$

$$y_n \leq z_n, \quad n \in H \tag{4.17}$$

$$z_n - y_n + s_n - d_{L(n)} - SL \geq 0, s_n \geq 0, \quad n \in F \tag{4.18}$$

$$z_n - y_n \leq MSL, \quad n \in N \tag{4.19}$$

$$a_n \leq y_n, z_n \leq b_n \quad n \in N \tag{4.20}$$

$$a_i \leq y_i \leq b_i, \quad i \in P \cup L \tag{4.21}$$

$$x_{nnDP(n)} = 1, \quad n \in N \tag{4.22}$$

$$x_e \in \{0, 1\}, \quad e \in A. \tag{4.23}$$

The objective function seeks to minimize the cost associated with the amount of overtime and part–time work that is scheduled. Taken together, constraints (4.2)–(4.7), (4.22) and (4.23) guarantee that a feasible solution will be made up of at most two types of components: (1) the first, see for example Figure 4.2.2(a), is a circuit containing in its "body" one nurse node and two or more patient nodes, plus a "tail" connecting a lunch node to a nurse or patient node of the circuit; and (2) a circuit, shown in Figure 4.2.2(b), containing only a part–time nurse and its corresponding dummy patient, with the corresponding lunch node forced to be an isolated node in the solution. The former refers to a complete route for the nurse in the component, and the latter indicates that the part–time nurse is not scheduled to work. Constraints (4.2)–(4.3) imply that every lunch node is either a member of exactly one component of the first type, or it is an isolated node in the solution. Due to constraint (4.3), a lunch node is adjacent to some node of a solution if and only if the corresponding nurse is scheduled to work. This interpretation is correct given the following observation: Since equation (4.22) ensures that each nurse visits his or her dummy patient first, it is easy to determine if that nurse is actually scheduled to visit any real patients by simply looking at the value of the arc from the dummy patient back to the nurse. If that arc has value 1, then the nurse does not visit any other patients. Thus, equation (4.3) forces a part–time nurse that does not visit any real patients to have a corresponding lunch node that is not visited. Full–time nurses are always scheduled to visit their lunch nodes since it is part of the regular shift length for which they are paid regardless of the hours that they actually work. Every non–lunch node is a member of exactly one circuit (that is, a "body" of some component).

The inequalities in (4.6) require for each node that the amount of flow entering the node from any nurse n must equal the flow out of that node from nurse n. Thus, the body must be composed of arcs which correspond to the same nurse n. The next

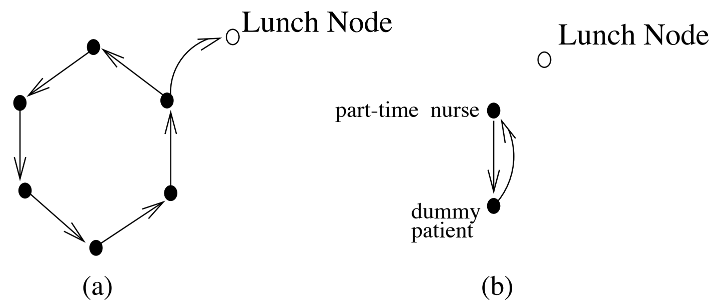

Figure 4.1 Two types of solution components:
(a) The solid nodes form the "body" of the main circuit containing one nurse node and five patient nodes, and the lunch node is connected to the body by a "tail." (b) The body is composed of only a part–time nurse and its dummy patient, and the corresponding lunch node is isolated.

constraint, (4.7), then forces the upper bound on any arc to a lunch node $k = L(n)$ from a feasible node i to be bounded by the amount of flow coming into node i from nurse n. Therefore, a lunch node can only be assigned to a route at node i which is being serviced by the feasible nurse n for that lunch. So, each connected component represents a feasible route for a specific nurse. These conditions restrict the solution set to routes that contain only one nurse, one dummy patient, any number of patients which that nurse is permitted to visit, and one lunch when the nurse is scheduled to visit more than just the dummy patient or the nurse is full–time.

Constraints (4.8)–(4.21) and (4.23) enforce the time window and route constraints. Clearly, (4.15)–(4.21) bound the variables y and z within the appropriate time window feasibility constraints. The constraints (4.8)–(4.14) are variations on the Miller–Tucker–Zemlin (MTZ) subtour elimination constraints [60] adapted for our model to include lunch nodes. All of these constraints are trivially true if any of the x variables on the right hand side of the constraint equal zero. Thus, they are equivalent to the following:

$$\begin{array}{rcl}
x_{nij} = 1 \Rightarrow & y_i + t_{ij} \leq y_j & n \in N, i,j \in P, i \neq j \\
x_{nin} = 1 \Rightarrow & d_{L(n)} x_{nnL(n)} + y_i + t_{in} \leq z_n, & n \in N, i \in P \cap B_n \\
x_{niL(n)} = 1 \Rightarrow & y_{L(n)} + d_{L(n)} \leq y_i, & n \in N, i \in P \cap B_n \\
x_{nnL(n)} = 1 \Rightarrow & y_{L(n)} + d_{L(n)} \leq z_n, & n \in N \\
x_{nnDP(n)} = x_{nDP(n)L(n)} = 1 \Rightarrow & y_n \leq y_{L(n)}, & n \in N \\
x_{nij} = x_{njL(n)} = 1 \Rightarrow & y_i + t_{ij} \leq y_{L(n)}, & n \in N, i \neq j \in P \cap B_n \\
x_{nin} = x_{nnL(n)} = 1 \Rightarrow & y_i + t_{in} \leq y_{L(n)}, & n \in N, i \in P \cap B_n.
\end{array}$$

Note that if a lunch node m is connected to some patient node j, then the start of lunch (y_m) must be at least the length of that lunch (d_m) before the start time at node j (y_j). In such a case, there must be some non–lunch node i such that the arc from i to j is traversed as well. Thus the start time of lunch y_m must also be at least as large as $y_i + t_{ij}$ since service at patient j may begin immediately after lunch. If a lunch node m is connected to its corresponding nurse node n, then the lunch must be taken after the last patient visited by that nurse and before the end of the nurse's shift (z_m). Note that if a nurse travels to a lunch node from some patient node i, then this implies that the lunch break is taken before providing service at patient i. Given these constraints, it is easy to show that the variables y and z have the correct interpretation in our model. (We also adapted the extensions to the MTZ constraints developed by Desrochers and Laporte [26] and found that the addition of these to our formulation generally did not improve performance of the solvers on the model.) And finally, the inequalities in (4.18)–(4.19) describe overtime for full–time nurses and limit the length of any nurse's shift to no longer than MSL, the maximum shift length.

This MIP formulation is very large. If n is the number of nurses and p is the number of patients, then the number of variables and the number of constraints is $O(np^2)$. The more constrained the problem is, though, the fewer variables are needed since in such a case there will be fewer feasible nurses between pairs of nodes.

4.3 Heuristics

Since the MIP formulation is so complex, a heuristic approach is most likely needed to handle large instances of the problem. The heuristic which we have chosen to implement is an intuitive two–phase algorithm. The first phase of the heuristic falls into the category of a parallel tour–building procedure since it typically builds several routes simultaneously. The second phase attempts to make improvements on the tours identified in phase one. A few important observations about this problem need to be made before describing the heuristic itself. First, note that the time windows are assumed to be "hard." In other words, if a nurse arrives at a patient's home prior to the earliest start time, a_i, then it must wait at that patient until the beginning of the time window before it may start servicing the patient. Similarly, service absolutely must begin by the latest start time, b_i. And finally, recall from the formulation that for patient j to follow patient i on some nurse's route, the starting times, y_i and y_j, at i and j, respectively, must satisfy the following conditions: $a_i \leq y_i \leq b_i, a_j \leq y_j \leq b_j, y_i + t_{ij} \leq y_j$.

In the first phase of our heuristic, feasible schedules, that is, collections of tours for each nurse, are generated using a randomized greedy algorithm. This algorithm takes as input the value of a random seed, as well as a number referring to the number of iterations to perform. The procedure begins with an ordered list of patients. The actual order of the list is varied from one iteration to the next. The list is first sorted by the patient time window lower bounds and then sectioned into intervals of size s.

The interval size s, determined through computational experimentation on our data sets, is taken to be $p, p/10, p/5, 11, 13, 14, 17$, or a random number between 10 and 25. Within each interval, the order of the patients is randomly determined. Once the patient order is set, the algorithm selects one patient from the list at a time and attempts to insert it into some nurse's schedule. In order to accomplish this goal, the method examines each feasible nurse for the selected patient and determines both the feasibility of visiting that patient next on the nurse's schedule and the feasibility of taking a lunch break next and then visiting that patient. This decision is randomized by varying the amount of time a nurse is permitted to wait at each patient before service may begin. This parameter was picked randomly from the set $\{0, 20, 40, 60, 80\}$ at each iteration. Thus, if the wait time parameter is set to zero for some iteration, then no wait time is permitted anywhere in the schedule. If both options are feasible, this parameter is also used to determine whether or not to take lunch prior to visiting the patient. If the difference in the two schedules' times is less than or equal to the wait time parameter, than the option including the lunch break is the one considered in the next step. We found that this encourages the heuristic to schedule the lunch breaks as early as possible in the routes. The next step is to insert the patient into the best possible nurse's schedule. At the end of each iteration of building a schedule, the overall best schedule is updated. A schedule is deemed better than the best schedule if (1) there is no best schedule, (2) the number of unscheduled patients in the new schedule is smaller than the corresponding number in the best schedule, (3) the number of scheduled patients is the same in both schedules, but the new schedule has less scheduled overtime and part–time, or (4) the number of patients scheduled and the sum of overtime and part–time are the same in both schedules, but the amount of full–time time or part–time scheduled is smaller in the new schedule than in the best schedule.

Once the best schedule is found with the randomized greedy algorithm, the second phase of our heuristic attempts to find improvements to this solution. In this next phase, a re–solve implies sending a problem with possibly fixed elements of the schedule and a known best solution back to the phase one algorithm so that unscheduled, or freed, patients can be scheduled via the randomized tour generator.

First, a form of local search is performed in the following manner:

- Select 2 nurses, i and j, such that at least one nurse is scheduled to work overtime or part–time.
- Free the patients on both nurse's routes while fixing all other patients in the schedule.
- Re–solve, both allowing and disallowing the nurses i and j to be scheduled.
- Keep the best known schedule created thus far.

The idea behind this procedure is to provide local fix–ups in small portions of the solution.

Next, an attempt is made to improve the current best solution by tightening the schedule. For each scheduled nurse, determine the first point in his/her route that the nurse is forced to wait. Fix all of the patients on the route prior to this point and free all of the rest. So, if a route has no waiting time, then all of the patients are fixed along that route. Note that other patients may be added at the end of the route prior to the nurse's return trip home. This reduced problem must now be re–solved.

At this point in the heuristic, the question is asked whether or not there are still unscheduled patients in the best known solution. If the answer is yes, then there are two procedures, insert and shuffle, which are performed in an attempt to schedule all of the patients. The insert procedure is a very basic attempt to insert

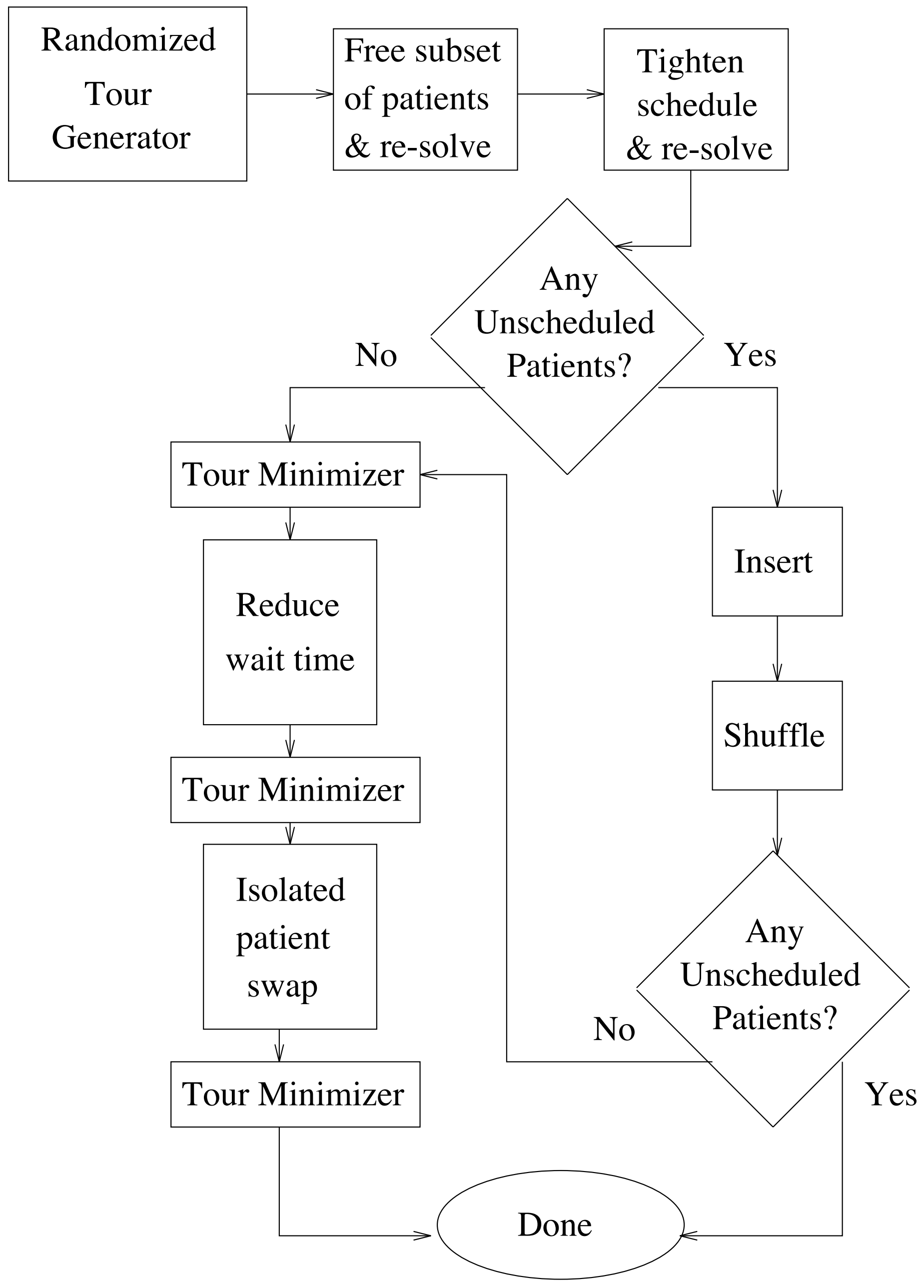

Figure 4.2 This flowchart describes the steps involved in the heuristic.

each unvisited patient into feasible nurse's routes. Typically this procedure is not especially beneficial, so it becomes necessary to shuffle other patients around to make room for the unvisited patient. The shuffle procedure in this heuristic considers each feasible nurse n for each unvisited patient, p_{lost}, until either all feasible nurses for p_{lost} have been considered or p_{lost} is placed in some route. For this nurse n, consider each patient p on n's route. If it is possible to insert p into some other nurse's route, then do the insertion, followed by a check to see if p_{lost} can now be inserted into nurse n's route. If p_{lost} was inserted into the schedule, then the shuffle procedure succeeded in this case and continues with the remaining unvisited patients. If p_{lost} was not inserted, then advance p to the next patient on the route (or if no other patients on the route, consider the next feasible nurse's route) and repeat.

Now, the question of whether all the patient have been scheduled is asked again. If unscheduled patients still exist, the algorithm terminates and reports the incomplete schedule. Otherwise, it continues with the tour improvement procedures that it performs if the schedule is complete without the insert and shuffle procedures.

The first of these is a procedure aimed at minimizing the time schedules for the given tours. The time schedule for each nurse's tour is made to be minimal by using the results on optimality of a time schedule by Florkiewicz and Kulej [36]. At the same time, a check is made to see if the lunch should be moved in the schedule. Recall that the first phase of the heuristic favors inserting lunches as early as possible. Sometimes a route can be greatly improved by simply moving the lunch. So, without changing the order in which a nurse visits assigned patients, the actual time may be reduced.

The second procedure seeks to reduce the maximum waiting time at a patient so long as that patient is visited by a part–time nurse. Given a schedule, the patient at which a nurse must wait the longest amount of time is determined. If that patient is visited by a part–time nurse, then reducing the waiting time by that nurse can reduce the overall amount of part–time labor required, so it is sent to the procedure. Similarly, if the patient is visited by a nurse who works overtime, then there exists the possibility of reducing the amount of overtime. If removing this patient from his or her assigned nurse reduces the amount of part–time or overtime in the nurse's tour, then the patient is removed. Then, if the patient can be inserted into another feasible nurse's schedule with a cost in terms of part–time and overtime which is less than the initial gain, the reduction is complete. If not, then the patient is returned to the original nurse's tour. If there is only one other patient on the nurse's tour, then this patient can be removed and tested for a better insertion place in an attempt to improve the overall solution, as well. This procedure can be called repeatedly, so long as the total amount of overtime and part–time labor in the best solution is reduced at each step. At the end of the loop to reduce the maximum waiting time, the tour minimizing procedure is run on the best solution.

The last procedure is labeled as an isolated patient swap. Oftentimes, the procedure to reduce wait times will contribute to the creation of tours in which nurses visit only one patient. This procedure takes such isolated patients and attempts to insert them into other feasible nurse's tours. If inserting the patient into the tour makes the new tour feasible and reduces the objective value, then the procedure ends. Otherwise, the new tour was made infeasible, and it is checked to see if moving one of the original patients onto the schedule of the nurse who was assigned to visit the isolated patient is a feasible option. If this swap is feasible and improves the overall solution, then it is performed. The entire process is repeated while improvements are found. At the end, the tour minimizing procedure is once again applied to the best solution.

4.3.1 Results from Heuristic

We tested our heuristic on two types of data sets. The first type of data we randomly generate using the following rules:

1. All lunch breaks have a length of 60 minutes.

2. The (x, y)–coordinates of the nurses' and patients' homes are supplied by the user. The traveling time between two points is an adjusted Euclidean distance. To be precise, the times are adjusted so that all of the traveling times are between 10 minutes and 60 minutes.

3. The service time required at each patient is chosen at random from 10 minutes to 30 minutes with 5 minutes as the step size.

4. The patient time windows are chosen at random from [0, 120] to [840, 1020] with 60 minutes as the step size.

5. To increase the chance of the existence of a feasible solution, a predetermined number of part-time nurses (usually one) have [0, 1440] as both the work time window and the lunch time window. Furthermore, they are compatible with every non–dummy patient.

6. The time windows for all other nurses are chosen at random from [0, 960] to [480, 1440] with 60 minutes as the step size.

7. The lunch time windows for these nurses depend on the corresponding nurse's time window. If $[a, b]$ is the nurse's time window, then the lunch time window for the nurse is defined to be $[a + 120, b - 120]$.

8. Given a nurse and a patient, the probability of their compatibility is 1/2 (unless the nurse is one of the part-time nurses already defined to be compatible with all non–dummy patients).

Case	Heur. Best	Cplex Opt	Cplex sec.	% Gap
1	689	652	219.69	5.7
2	967	896	1966.61	7.9
3	402	396	687.23	1.5
4	360	360	180.27	**0.0**
5	456	456	438.25	**0.0**
6	378	354	617.29	6.8
7	471	451	306.57	4.4
8	622	588	56.09	5.8
9	485	485	288.18	**0.0**
10	527	517	224.88	1.9
11	564	551	352.94	2.4
12	483	483	400.21	**0.0**
13	557	547	522.11	1.8
14	410	351	161.05	16.8
15	296	296	391.35	**0.0**
16	463	463	473.28	**0.0**
17	403	403	456.68	**0.0**
18	477	477	735.57	**0.0**
19	320	307	193.91	4.2
20	274	274	320.81	**0.0**
21	426	409	713.58	4.2
22	312	312	242.70	**0.0**
23	458	458	743.12	**0.0**
24	302	302	219.89	**0.0**
25	260	260	447.81	**0.0**
26	512	503	196.13	1.8
27	427	427	319.02	**0.0**
28	445	445	812.93	**0.0**
29	385	337	119.44	14.2
30	286	286	327.43	**0.0**
31	528	515	414.07	2.5
32	445	445	103.62	**0.0**
33	440	440	966.26	**0.0**
34	364	328	303.82	11.0
35	289	241	402.62	19.9
36	409	409	410.64	**0.0**
37	404	404	200.79	**0.0**
38	487	487	1229.71	**0.0**
39	323	279	379.69	15.8
40	267	267	1062.36	**0.0**

Table 4.1 Heuristic results on data sets with 2 full–time nurses, 2 part–time nurses, 10 patients

We ran our test data on a Sun UltraSparc I 167 MHz machine with 192 megabytes of memory. The largest problem instance of this type that CPLEX [22] was able to solve using the MIP solver of version 5.0 contained 2 full–time nurses, 2 part-time nurses, and 10 patients. With dummy patients and lunch nodes, this corresponds to a problem on a graph with 22 vertices. The next largest size tested on CPLEX contains 2 full–time nurses, 3 part–time nurses, and 14 patients, and was unable to solve with a node limit of $500,000$. We tested the heuristic on 40 different problem instances generated with 10 different seed values for 4 different input files. Since the heuristic takes a random seed and the number of iterations as input each time it is run, we used three different input files to generate three solutions for each problem instance. One file contained the value 10 for the iteration parameter and the other two files used the value 100. Each file contained a different random seed value. The best solution value out of the three runs is reported in Table 4.1 under "Heuristic Best." The optimal value and the time CPLEX required to determine it are reported next, and finally the percentage optimality gap between the heuristic value and the optimal value can be found in the last column. The time the heuristic took to run is less than a second in all cases. Upon closer examination of the five instances with an optimality gap greater than 10%, we found that our heuristic solutions could be made optimal by a sequence of patient moves from one schedule to another without further rearrangement of the tour. (The isolated patient swap considers only a special case of this type of move.) Any procedure which attempts to consider all such patient moves would have a high level of complexity due to the multi-depot, time window, and lunch-constrained nature of the problem. We tried to avoid this cumbersome and timely type of procedure by using the randomness inherent in our heuristic to combat this type of problem.

Unfortunately, it is not possible to evaluate the heuristic's performance on larger data sets, since we have no means to determine the optimal solution in such cases. We do, however, have three problem instances that were given to us by the Fortune 500 company which originally brought us this problem. These are generated instances, as well, but details on the rules used to generate them were not provided. We do know that this company developed its own heuristic for the problem as well, and their method was able to find a solution for instance A with some difficulty, could not determine if instance B is solvable; and solved instance C easily. From the results in Table 4.2, it appears that our heuristic did at least as well as the company's heuristic. Neither method could find a solution for instance B, but in our case, we could supply a partial solution that excludes only 6 out of 900 patients.

Case	num-nurses	num-patients	unvisited	time (sec)
A	30	96	0	1.0
B	229	900	6	78.0
C	294	900	0	100.0

Table 4.2 Heuristic results on outside company data sets

4.4 Exact Branch–and–Cut Procedure

Since only very small instances of the home health care problem can be solved by standard MIP solvers, we implement a branch–and–cut exact solution method in an attempt to determine optimal solutions for larger instances. This procedure begins with the LP relaxation of the entire formulation containing the smaller, though weaker, set of subtour elimination constraints. It then searches for violated cutting planes to add to the initial LP relaxation, as well as to the nodes of the branching tree. We consider subtour, blossom, and comb inequalities, originating in the TSP literature, as well as 2–path cuts from the more recent VRPTW literature. The main idea behind employing this method is to significantly reduce the number of nodes in the search tree by adding cutting planes to the LP, thus reducing the overall time required to determine the optimal solution.

4.4.1 TSP–type Inequalities

To detect valid inequalities that are based on results from the symmetric traveling salesman problem, we consider a graph obtained from an optimal solution, (x, y, z), to the LP relaxation. Let G be the graph induced by x with the removal of all lunch nodes and their adjacent edges. If this graph has a component that is a cycle of patients, then this cycle is a subtour. Although our formulation guarantees the exclusion of subtours in an integer solution, it is well-known that the constraints in our formulation produce a weaker LP relaxation than the Dantzig, Fulkerson, and Johnson (DFJ) subtour elimination constraints [26]. The DFJ subtour elimination constraints are written as $x(\delta(S)) \geq 2$ [23]. For our directed graph, this translates to the pair of constraints $x(\delta(S)) \geq 1$ and $x(\rho(S)) \geq 1$ for $S \subseteq P$. It is easy to see that $x(\delta(S)) = x(\rho(S))$ always holds in our problem, so that only one of these sets of constraints, say $x(\delta(S)) \geq 1$, is required to cut off a fractional optimal solution to the LP relaxation which violates the DFJ subtour elimination constraints. Since there are exponentially many of these constraints, only those that are violated are added to the LP relaxation.

Once a set S is found such that $x(\delta(S)) < 1$, the corresponding DFJ subtour elimination constraint can be added to the problem. By identifying all of the nurse nodes to a single depot node in the graph G, the separation problem for the subtour elimination constraints becomes a standard application of a maxflow algorithm. Once found, these subtour inequalities can potentially be strengthened by letting the right hand side equal the minimum number of nurses needed to visit the set S. Unfortunately, this is not a trivial value to determine. A lower bound on this value, however, can be determined based solely on the minimum number of nurses that, based on compatibility information, can cover the set S. This bound corresponds to the solution of the LP relaxation of an integer programming problem which minimizes the number of nurses subject to every patient having at least one nurse which may visit it. We, however, did not attempt to increase the right hand side of the subtour elimination constraints in this way since we designed the problems in our test cases so that at least one part time nurse was feasible for every patient, thus making the right hand side lower bound always equal to one.

The formulation of the blossom and comb inequalities in the context of the home health care problem varies only slightly from that of the TSP. *Blossom inequalities* are a subset of the more general *comb inequalities*. A *comb* is defined by a subset of the vertex–set H, known as the *handle*, and the subsets T_i $(i = 1, 2, \ldots, 2k+1)$ of the set of patient–nodes, known as the *teeth*, such that

$$x(\gamma(H)) + \sum_{i=1}^{2k+1} x(\gamma(T_i)) \leq |H| + \sum_{i=1}^{2k+1} (|T_i| - 1) - (k+1)$$

is a valid inequality [19, 40]. If every tooth is of size two, we have the blossom inequalities. If A is the set of edges corresponding to the teeth, the blossom inequalities can be written as:

$$x(\delta(H) \setminus A) + (|A| - x(A)) \geq 1.$$

To separate over the blossom inequalities, we mimic the technique of Padberg and Grötschel [61]. To modify this procedure to our problem, though, we may not simply identify all of the nurse nodes. The separation algorithm first requires finding the components of the graph induced by x. It is easy to see that there is a violated blossom inequality if and only if there is a violated blossom inequality in a component. A component that is a cycle is ignored since it must be a valid route for some nurse. For each of the non-cycle components, we determine whether there is a violated blossom inequality contained in it. Let H be the undirected graph obtained from a component by deleting the direction on all of the edges and by replacing multiple edges with a single edge whose weight is the sum of the weights of the edges that it replaces. Consequently, an edge in H may have a weight greater than one. In such instances, so long as there are no violated subtour inequalities in the x solution, one of the edge's end nodes must be a nurse node. For the TSP, the procedure of Padberg and Grötschel transforms H into H_1 by replacing every edge (v, w) with weight x_e by three edges (v, u_1) with weight x_e, (u_1, u_2) with weight $1 - x_e$ and (u_2, w) with weight x_e The new nodes u_1 and u_2 are known as *subdivision nodes.* Let T be the set of subdivision vertices. In the TSP, the separation problem for the blossom inequalities can be solved by finding a minimum T-cut in H_1. (A T–cut is a cut S such that $|S \cap T|$ is odd, and $|T|$ is even.) In our case, the only adjustment necessary is to replace the weight of (u_1, u_2) by ∞ if either of v or w is a nurse node. This modification corresponds exactly to handling those edges that may have weights greater than one in our model. Then, as in the TSP, the separation problem for the blossom inequalities reduces to finding a minimum T–cut in H_1. Thus, the inclusion of a subdivision edge (u_1, u_2) in the cut corresponds to the edge (v, w) being a tooth. Hence, all of the vertices contained in the set of teeth must be patient nodes, since otherwise the minimum T–cut would have an infinite value.

Finally, we implement a heuristic algorithm for finding violated comb inequalities. As in the TSP, if S is a set (in our case $S \subseteq P$) and S is tight with respect to our subtour inequalities (that is, $x(\delta(S)) = x(\rho(S)) = 1$), then we can contract S to a single node. Moreover, if there is a violated blossom inequality in the reduced graph, there is a violated comb inequality in the original graph. We use the heuristics of Grötschel and Holland [39] to choose such a set S.

4.4.2 2–path Inequalities

In addition to the TSP–related inequalities, we adapt a separation algorithm for finding 2–path cutting planes in the VRPTW to this nurse scheduling problem. The more general k–path inequality was introduced for the VRP by Laporte, Nobert, and Desrochers [54], and for the VRPTW by Kohl, Desrosiers, Madsen, Solomon, and Soumis [46]. Kohl et al. establish heuristic and exact routines for finding violated cuts in the case where $k = 2$ [46]. Since a 2–path inequality can easily be translated into the appropriate context for this problem, we base an alternate algorithm for finding these cutting planes on their work.

The k–path inequality is very similar in appearance to the subtour inequality except that the value one on the right hand side is replaced by a value $k(S)$: $x(\delta(S)) \geq k(S)$. For the VRPTW, $k(S)$ refers to the number of vehicles needed to service all customers in the set S (S may not include the depot). For the home health care problem under consideration, the value $k(S)$ for a subset of patients S must be interpreted as the number of nurses needed to visit all patients in the set. More specifically, a 2–path inequality refers to the instance where $k(S) \geq 2$, indicating that at least 2 routes are required to service the set S. Thus, a 1–path inequality, which clearly must be true for any subset not containing the depot, is actually a DFJ subtour constraint.

As in the VRPTW, calculating $k(S)$ for a set S of patients is very difficult. Such a calculation must take into consideration both time windows and nurse–patient feasibility restrictions. The problem becomes more manageable, though, if the focus is only to identify 2–path cuts. Since by definition $k(S)$ is an integer value, determining if $k(S) \geq 2$ is equivalent to finding that $k(S) > 1$. Two conditions must be checked in order to verify that more than one nurse is required to visit the set of patients S

(that is, $k(S) > 1$). The first check determines which nurses, if any, are feasible for every patient in the set. This can be accomplished in polynomial time. If there are no nurses which can visit the entire set, then clearly this set yields a valid 2–path inequality. Otherwise, it must be determined whether or not a feasible route which visits all of the patients in the set can be found for any of the feasible nurses for the set. If we disregard the scheduling of the lunch break, this amounts to solving a TSPTW for each of the feasible nurses. The TSPTW–feasibility problem is $\mathcal{NP}$–hard, but if the size of S is small, as in our case, then dynamic programming can be used to solve the problem relatively easily [29]. If the TSPTW is infeasible for all of these nurses, then we have a valid 2–path inequality. If the TSPTW is feasible for any of these nurses, then we may or may not have a valid 2–path cut. This rather difficult determination, which we do not address, depends on whether or not there exists a feasible schedule including the lunch break. Thus, we have a fast, though not polynomial, algorithm which, in certain instances, can determine if $k(S) > 1$.

To find a violated 2–path cut, we must find a subset of patients S such that $k(S) \geq 2$ is valid and the condition $x(\delta(S)) < 2$ is satisfied in the directed, fractional x solution. The method which we propose to find these cuts first identifies sets of patients which currently have a directed outward flow of less than two. Then we check to see if we can determine that the outward flow must be greater than or equal to two. If we can, then we have a violated 2–path cut. So long as we include lunch breaks in the model and do not attempt to determine the feasibility of the TSPTW with lunch breaks , this approach cannot guarantee that all violated 2–path cuts will be found.

Identifying the sets S of patients which have less than two paths entering them requires first building an undirected graph $H = (V, E)$ from the current x vector solution of the LP relaxation. The nodes of H are the nurse and patient nodes, including dummy patients, from the original graph. The graph's edges represent the cumulative flow in either direction between a pair of nodes. If there is no flow between a pair of nodes, then there is no edge connecting the nodes. Otherwise, each undirected edge ij has a capacity, or weight, equal to the positive value: $\sum_{n \in N}(x_{nij} + x_{nji})$. Note that in an undirected graph, the notation $\delta(R)$ for a set of nodes R is defined as the set of edges $\{e \in E : e$ has an end in R, and an end in $V \backslash R\}$. The weight of a set of edges, F, is denoted $w(F)$. Since the x solution satisfies the condition that the flow into a set of patients equals the flow out of the set, then clearly the value, or weight, of a cut $\delta(R)$, where R is a set of patients, is double the value of the corresponding directed cut in the original graph. Thus, identifying a set of patients S such that the condition $x(\delta(S)) < 2$ holds in the original solution amounts to finding a set in H satisfying the condition that $w(\delta(S)) < 4$.

If the graph H is built from an x solution which does not contain any violated subtour inequalities, then the minimum value of any cut in the graph is 2. Thus, our goal is to find every set S of patients in the graph H such that the value of the cut is less than double the minimum cut. A random contraction algorithm developed by Karger [42] is used to accomplish this task. In an implementation–oriented paper by Karger and Stein [43], the authors prove that using Karger's algorithm on an undirected graph with n vertices finds all cuts with weight within a multiplicative factor α of the minimum cut in $O(n^{2\alpha} log^3 n)$ time. We use a modified version of this algorithm with $\alpha = 2$ to generate the relevant sets S such that $S \subseteq P, |S| > 1$, and $w(\delta(S)) < 4$. The version used is not guaranteed to generate all of the relevant sets, but does run in less time. Further details on Karger's algorithm are given in the next chapter.

Algorithm 4.1 Heuristic for the 2–path separation problem

(1) Build undirected graph H from x vector.

(2) Use Karger's algorithm to find
$\Gamma = \{S : S \subseteq P, |S| > 1, w(\delta(S)) < 4\}$.

(3) For every $S \in \Gamma$, if $k(S) \geq 2$ then add S to list of valid inequalities.

4.4.3 Some Computational Experience

The maxflow code used in the subtour elimination procedure comes from an implementation written by R. Anderson and J. Setubal [2]. The implementation of Karger's algorithm was written by Sanjeeb Dash, a Rice University PhD student. All computation was done on a Sun UltraSparc I 167 MHz machine with 192 megabytes of memory with code written in the ANSI C programming language.

The exact solver requires several key pieces of information. First, the actual data defining the problem must be supplied. Second, one of two formulations must be selected. The primary formulation is the basic one found in the MIP formulation section. The other option adds additional constraints to the MIP so as to strengthen the formulation. These extra constraints extend the MTZ subtour elimination constraints as in Desrochers and Laporte [26]. Although this second option was tested, the results were generally far worse with the extra constraints. Since the stronger DFJ subtour elimination constraints are added to the model as cutting planes, strengthening the weaker constraints provides little value. Thus, all results presented here refer to the basic formulation. And finally, the third user–supplied information indicates whether or not a depth–first search or a best–bound breadth–first search is performed. The depth–first search recursively searches the nodes of the tree. The breadth–first approach maintains a heap of the nodes of the tree, and selection is based on the node with the lowest objective function value.

Several branching options are available to the exact solver, as well. To aid in the branching decision, the formulation was altered to incorporate a new binary variable q_{ni} that indicates whether or not a nurse n is assigned to visit patient i. In mathematical terms, this is equivalent to the constraint $q_{ni} = \sum_{j \in (B_n \setminus L(n))} x_{nij}$ for every $n \in N$ and $i \in (B_n \cap P)$. Thus, branching on a fractional q variable effects many x variables simultaneously. The branching strategy applied in all of the test cases, first branches on the most fractional q variable, if one exists. Otherwise, the most fractional x variable is selected as the next branching variable. The variable selection process can have a significant effect on the solution procedure. For example, the use of pseudo–cost variable selection by CPLEX, rather then branching on maximal infeasibilities, reduces the time and number of nodes by as much as a factor of thirty. Unfortunately, the approximate pseudo–cost methods which we applied were not successful. It is likely that more study of possible branching strategies in this setting could greatly improve the success of this approach.

				DFS		BFS	
Case	Opt	Best	% Gap	nodes	seconds	nodes	seconds
1	652	689	5.7	71,113	1834.63	80,966	5667.76
2	896	967	7.9	115,339	2730.05	135,016	9561.83
3	396	402	1.5	69,408	1787.56	78,166	4915.82
4	360	360	0.0	10,495	324.59	10,977	713.82
5	456	456	0.0	105,008	2351.27	122,791	6668.14
6	354	378	6.8	114,154	3029.59	126791	8884.03
7	451	471	4.4	48,262	1204.34	47,319	3592.58
8	588	622	5.8	13,385	392.42	18,704	981.24
9	485	485	0.0	26,008	622.40	32,917	1756.13
10	517	527	1.9	52,980	1237.95	59,311	3810.30
Totals				626,152	15,514.8	712,958	46,551.65

Table 4.3 Branch–and–bound (no cutting planes) on instances with 2 full–time/2 part–time nurses, 10 patients

Table 4.3 establishes the base line by which the effectiveness of adding cutting planes to the LP throughout a branch–and–bound scheme is judged. Results are obtained for ten data sets, each with two full–time nurses, two part–time nurses, and ten patients. The results in Table 4.3 correspond to an implementation which ignores the cutting plane procedures and finds the optimal solutions by applying a branch–and–bound algorithm. The column marked "Opt" refers to the value of the optimal solution to the problem instance. The "Best" column contains the best known solutions provided by the heuristic. These values become the starting upper bounds for the branch–and–bound and branch–and–cut approaches. The "Gap" column indicates the size of each optimality gap which ultimately must be eliminated to prove optimality. The results under the "DFS" heading indicate the number of nodes and the total amount of time required to solve the problem using a depth–first search tree. Similarly, the "BFS" results relate to the use of a best–bound breadth–first search tree. The depth–first search tree is faster in our implementation due to the need for less storage, as well as the efficiency of reoptimization methods. Thus, from this point forward only the results from the DFS approach are reported.

The success of the subtour inequalities within the DFS branch–and–cut approach is illustrated in Table 4.4. In this implementation of the branch–and–cut algorithm, separation routines for finding subtour, blossom, and comb inequalities are called at the root node of the tree. At all subsequent nodes of the search tree, the search for violated valid inequalities is restricted to subtour cuts. Table 4.4 reports the number of nodes in the tree, the total time in seconds required to find the solution, and the number of subtour ("Sub"), blossom, and comb inequalities found throughout the tree. The branch–and–cut approach using only subtour cuts clearly reduces the number of nodes in each of the ten instances. In addition, the solution time decreases in all but one of the data sets.

Case	Subtour Cuts		Cutting Planes		
	nodes	seconds	Sub	Blossom	Comb
1	38,487	1299.85	15		
2	67,742	2369.79	27		
3	47,726	1707.39	9		
4	9,223	381.74	5		1
5	59,616	1858.94	10		
6	69,671	2446.39	9		
7	23,701	844.82	12		
8	9,412	344.74	2		
9	17,664	583.67	6		
10	26,507	866.17	8		1
Totals	369,749	12,703.5	103		2

Table 4.4 DFS Branch–and–cut

Case	Subtour & Combs		Cutting Planes		
	nodes	seconds	Sub	Blossom	Comb
1	36,569	2869.83	16	900	5340
2	65,996	4105.46	29	489	4992
3	44,860	3995.86	9	1066	5442
4	7,428	1004.73	5	139	1300
5	54,339	3528.26	10	1008	5045
6	70,009	6452.75	10	976	6713
7	23,273	2147.39	12	464	3814
8	9,215	835.15	2	22	1421
9	16,857	1130.39	6	200	1049
10	26,508	1993.47	8	464	2436
Total	355,054	28,063.29	107	5,728	37,552

Table 4.5 DFS Branch–and–cut

The branch–and–cut algorithm used to obtain the results in Table 4.5 searches for subtour, blossom, and comb inequalities at every node of the search tree. Versions of the code which at any given node of the tree keep all cuts in the formulation, remove all cuts from the formulation, and keep only subtour cuts in the formulation were tested. The best results, shown in Table 4.5, correspond to the case which keeps only the subtour cuts in the formulation as the search tree is explored. Since this allows for the same blossom or comb inequality to be added to an LP relaxation in different parts of the search tree, the reported number of blossom and comb inequalities does not reflect the total number of different blossom and comb cuts found in the search tree.

	Subtour & Combs*		Cutting Planes		
Case	nodes	seconds	Sub	Blossom	Comb
1	39,871	1206.08	15	11	90
2	68,997	2119.45	27	18	75
3	47,674	1539.26	9	9	71
4	9,189	352.38	5	1	27
5	57,392	1552.49	10	26	65
6	67,054	2063.46	9	12	91
7	23,733	762.63	12	8	65
8	8,718	289.62	2	1	17
9	17,469	505.68	6	2	26
10	26064	736.21	8	8	40
Total	366,161	11,127.26	103	96	567

Table 4.6 DFS Branch–and–cut

Adding the blossom and comb inequalities to the subtour cuts does not have as significant an effect on the number of nodes as adding the subtour cuts to the non–cutting plane procedure did. Although the number of nodes typically decreases, the solution time uniformly increases. To reduce the overall time, a modified branch–and–cut approach is used to generate the results in Table 4.6. Rather than applying the cutting plane procedures at every node, the algorithm only searches for cuts every 100 nodes. Since the search tree for any given problem has a large number of nodes which do not provide any violated inequalities, this reduces the amount of unproductive time the algorithm spends in the cutting plane procedures. Additionally, adding only a couple of cuts to many LP relaxations is computationally more expensive than adding many cuts to only a couple of LP relaxations. Thus, applying cuts at set intervals of the tree reduces the solution time. Thus, the cut frequency within a branch–and–cut algorithm can be set at values greater than one. Of course, some of the value of adding cutting planes at every node is lost, so the number of nodes is not quite as small.

	DFS–Sub & 2–path		Cutting Plane s			
Case	nodes	seconds	Sub	Blossom	Comb	2–path
1	10,650	2014.54	11			552
2	14,842	1600.59	9			379
3	48,172	3451.48	7			50
4	9,187	850.17	5		2	198
5	55,976	3697.84	8			23
6	51,835	4787.49	6			327
7	19,233	2309.37	6			380
8	6,546	1120.26	2			527
9	10,615	1270.55	6			399
10	15,172	2078.33	7			501
Total	242,228	23,180.62	67		2	3,336

Table 4.7 DFS Branch–and–cut

The 2–path separation algorithm requires that the solution contains no violated subtour inequalities. It does not, however, need to have all blossom and comb cuts eliminated. So, the effectiveness of the 2–path cutting planes must be tested in conjunction with the subtour elimination cuts. These results are found in Tables 4.7 and 4.8. In both tables, the applied algorithm uses all cutting plane procedures at the

Case	DFS–Sub & 2–path*		Cuttin g Planes			
	nodes	seconds	Sub	Blossom	Comb	2–path
1	20,236	1140.64	12			196
2	22,467	846.43	8			95
3	49,362	1597.83	8			11
4	8,525	325.01	5		2	17
5	55,444	1512.07	8			7
6	60,569	2344.00	6			107
7	24,281	861.81	9			63
8	9,770	438.83	2			65
9	14,241	509.48	6			65
10	20,557	851.99	8			145
Total	285,452	10,428.09	72		2	771

Table 4.8 DFS Branch–and–cut

root node of the search tree and only subtour and 2–path cuts elsewhere. Results in Table 4.7 correspond to cutting at every node of the tree, while those in Table 4.8 refer to cutting at every 100th node. In both cases, all cuts are kept in the formulation. The results in Table 4.8 are the best so far.

Finally, Figure 4.3 shows the number of nodes in the search tree and the solution time for five different branch–and–cut scenarios using all four types of cuts. The number of nodes in the tree and the corresponding solution time for the branch–and–bound approach without any cutting planes is given by the "No Cuts" black line. In the other four methods, the term cuts refers to subtour, blossom, comb, and 2–path inequalities. Keeping all cuts refers to leaving the cuts in the LP relaxation. Removing combs means that comb inequalities are applied to the node at which they are found, and then they are removed from the formulation. And an asterisk (*) indicates that cutting was only applied to every 100th node of the search tree. The graphs clearly show that applying as many cuts as possible at every node of the search tree reduces the number of nodes. However, it also illustrates that in such cases the

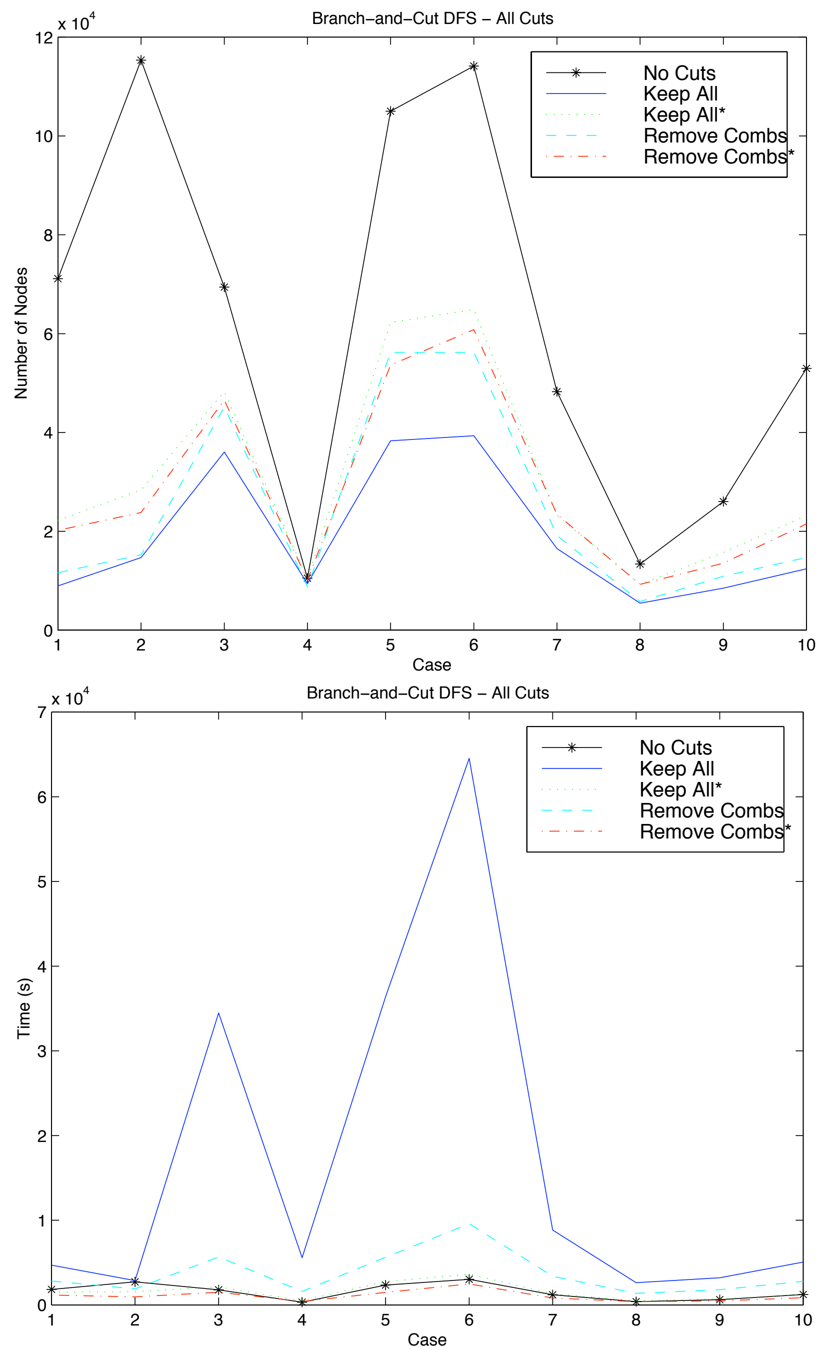

Figure 4.3 The top graph shows the number of nodes in the search tree for the ten data sets, given five different branch–and–cut scenarios. The bottom graph shows the time in seconds for the same set of instances.

cost in terms of time completely overshadows the savings in the number of nodes that must be explored. By not applying the cutting plane techniques at every node of the search tree, a compromise is reached that achieves small savings in both number of nodes and time.

Although the separation procedures and cutting planes described in this thesis are known to work well for the TSP and the VRPTW, they do not yield as effective results in our test sets. Although many violated inequalities are found over the entire branch–and–cut search tree, only a handful, if any, are found at any given node of the tree. The cutting plane procedures do succeed in reducing the number of nodes in the branch–and–cut tree, but oftentimes the cost in terms of time is too large to justify adding the cuts. When solution time increases, it is generally due to a combination of the time spent in the separation routines, as well as in the reoptimization of LP relaxations containing new cutting planes. To combat both of these factors, cutting algorithms can be applied at regular intervals within the search tree, rather than at every node. While this does achieve some savings in the solution process, it does not improve the approach enough to enable it to solve larger instances of the home health care problem.

4.5 Disregarding Lunch Breaks

Despite the necessity of having lunch breaks, the actual scheduling of such breaks is a convention that is rarely followed in practice. Instead, the nurses fit their meal breaks into their day whenever possible. Though this may seem less efficient, oftentimes nurses prefer the flexibility which this allows them.

Removing lunches from the model not only simplifies the formulation, but also improves the separation algorithm for 2–path cuts. The 2–path cutting plane routine requires determining whether or not a feasible route which visits a set S of patients exists for a given nurse. Without lunch breaks in the model, this can be determined definitively by solving a TSPTW. Thus, the same 2–path cutting plane algorithm as before generates more accurately the 2–path cuts in the simplified model.

Case	Opt	Best	% Gap	DFS nodes	DFS seconds
1	569	569	0.0	61,008	794.02
2	836	∞	∞	107604	1278.55
3	363	360	0.8	60,428	812.21
4	342	342	0.0	8,717	142.10
5	336	336	0.0	75,969	898.02
6	258	258	0.0	80,532	1135.35
7	393	393	0.0	36,987	493.85
8	468	468	0.0	9,655	160.09
9	365	365	0.0	18,009	242.63
10	472	444	6.3	46,648	568.80
Totals				505,557	6,525.62

Table 4.9 Branch–and–bound (no cutting planes) on instances without lunch breaks

The same set of ten data sets, with the lunches removed, are tested in the revised model. The heuristic can still be used to obtain upper bounds by simply making the lunch break have a duration of zero units of time. The heuristic did not find a solution for case 2, so the infinity value is used as the upper bound. The results of the depth–first search branch–and–bound algorithm without cutting planes are presented in Table 4.9.

Once again, the addition of subtour cuts within the branch–and–cut algorithm improves the overall results. The cumulative number of nodes over all ten cases is reduced by 44%. The time reduction, however, is much less significant: 11.5%. The individual results in Table 4.10 show the number of nodes, the time, and the number of cuts for each instance. As in the instances including lunch breaks, the branch–and–

Case	Subtour Cuts		Cutting Planes		
	nodes	seconds	Sub	Blossom	Comb
1	30529	607.67	15		
2	57484	1189.33	27		
3	38612	786.17	9		
4	8017	194.91	5		
5	42752	810.26	10		3
6	44479	929.66	9		
7	20472	440.42	12		
8	5948	138.74	2		
9	10874	228.31	6		1
10	23850	447.15	9		
Total	283,017	5,772.62	104		4

Table 4.10 Without Lunches: DFS Branch–and–cut

cut algorithm using subtour elimination cutting planes is the only version in which cutting at every node of the search tree proves to be efficient. Tables 4.11 and 4.12 give results using subtour and comb inequalities and subtour and 2–path inequalities, respectively, at every node. Though the number of nodes decreases for the latter, the total solution time for both is unreasonably large.

As illustrated in Tables 4.13, 4.14, and 4.15, using a cut frequency of 100 within the branch–and–cut code yields similar results regardless of which cutting planes are applied. The branch–and–cut approach with subtour and 2–path inequalities and a cut frequency of 100 is slightly more effective on these ten problems. The number of nodes per problem instance which must be explored is still very large, considering that this number will grow exponentially as the problem size increases. Despite the improvements over the non–cutting plane branch–and–bound approach, this branch–and–cut approach still proves ineffective on problems of a larger, more realistic size.

Case	Subtour & Combs nodes	Subtour & Combs seconds	Cutting Planes Sub	Cutting Planes Blossom	Cutting Planes Comb
1	28550	1978.44	15	1019	5162
2	65563	3153.56	27	656	5630
3	38276	2999.33	9	1242	5920
4	7063	802.35	5	119	1177
5	38377	2297.70	10	1088	5037
6	51073	4768.32	9	1394	8258
7	19392	1658.99	14	501	4083
8	5197	444.24	2	13	1051
9	11435	726.77	6	117	825
10	22804	1400.80	9	316	1972
Total	287,730	20,230.5	106	6,465	39,115

Table 4.11 Without Lunches: DFS Branch–and–cut

Case	Subtour & 2–path nodes	Subtour & 2–path seconds	Cutting Planes Sub	Cutting Planes Blossom	Cutting Planes Comb	Cutting Planes 2–path
1	7382	1300.04	12			545
2	14782	1398.42	9			382
3	37096	2004.96	6			43
4	8094	609.03	5			220
5	39643	2099.91	9			22
6	38341	2997.80	8			349
7	14127	1393.60	9			328
8	4212	670.79	2			507
9	6790	716.07	5			406
10	14121	1801.09	6			488
Total	184,588	14,991.71	71			3,290

Table 4.12 Without Lunches: DFS Branch–and–cut

Case	Subtour & Combs*		Cutting Planes		
	nodes	seconds	Sub	Blossom	Comb
1	31367	503.40	15	6	86
2	68380	1130.58	27	4	54
3	39203	665.84	9	15	64
4	8083	165.96	5	2	10
5	42407	643.46	10	11	80
6	47284	801.29	9	12	98
7	19824	347.70	12	4	57
8	5514	109.13	2		6
9	11396	204.02	6		9
10	25374	391.10	9	3	33
Total	298,832	4,962.48	104	57	497

Table 4.13 Without Lunches: DFS Branch–and–cut

Case	Subtour & 2–path*		Cutting Planes			
	nodes	seconds	Sub	Blossom	Comb	2–path
1	16255	566.76	13			164
2	21823	476.36	8			83
3	38327	656.47	8			12
4	8158	175.97	5			17
5	39905	596.09	9			9
6	44626	1083.71	8			126
7	17416	381.09	8			61
8	5590	134.85	2			43
9	9142	193.66	5			59
10	19129	516.55	6			138
Total	220,371	4,781.51	72			712

Table 4.14 Without Lunches: DFS Branch–and–cut

	All Cuts*		Cutting Planes			
Case	nodes	seconds	Sub	Blossom	Comb	2–path
1	15266	600.54	12		10	190
2	21558	475.24	8		1	87
3	39734	723.31	7	7	46	16
4	6655	134.02	4		9	9
5	40908	637.85	9	9	57	9
6	42423	1022.98	8	7	37	107
7	18056	426.18	9	1	28	66
8	5759	161.13	2	4	28	60
9	9507	224.97	5		6	79
10	19399	524.35	8	1	11	136
Total	219,265	4,930.57	72	29	233	759

Table 4.15 Without Lunches: DFS Branch–and–cut

Chapter 5

k–Path Cuts for the VRPTW

For the nurse scheduling and routing problem of chapter 4, we develop a separation routine for finding 2–path cuts based upon the method introduced by Kohl, Desrosiers, Madsen, Solomon, and Soumis [46] for the vehicle routing problem with time windows. To demonstrate that the new algorithm is an improvement over the previous method, in this chapter we test the new routine in its original context of the VRPTW. Additionally, the new separation algorithm is extended to search for the more general k–path cutting planes. Computationally, 2– and 3–path cuts are implemented within the solution procedure framework described by Kohl et al [46]. Thus, the effectiveness of the new algorithm is easily compared to the previous results.

As noted, the vehicle routing problem with time windows involves finding a set of routes starting and ending at a single depot that together visit a set of customers. Each customer must be visited once and within a certain time window. Each route is limited not only by time constraints, but also by the vehicle's capacity. Each customer has a demand, and the accumulative demand over all of the customers on a route may not exceed the vehicle's capacity. The objective can be to minimize the total distance traveled, the number of vehicles used, or the combination of both of these goals.

In the first section of this chapter, a mathematical formulation of the VRPTW is defined. Though this model is not solved directly, it does provide the appropriate context in which one should view the problem. The set partitioning model used in the solution procedure is presented in section 5.2, along with the column generation approach used to solve the LP–relaxation of the model. The next section explains the concept of valid inequalities and their application in the VRPTW. Section 5.4

introduces a new separation routine for finding 2–path cutting planes and section 5.5 presents a separation routine for finding k–path cutting planes. The benchmark problems used to test the solution procedure are described in section 5.6 and the implementation details are explained in section 5.7. Finally, the computational results are given in section 5.8.

5.1 Mathematical Formulation

The VRPTW is defined on a directed graph $G = (\mathcal{N}, \mathcal{A})$. The set of nodes, $\mathcal{N}$, contains the depot node and a set of clients, $\mathcal{C}$. The depot node is represented as 0 and $n+1$, and the clients are labeled from 1 to n. The set of arcs, $\mathcal{A}$, indicates the potential connections among the nodes. Node 0 is incident to outgoing arcs only, while node $n+1$ is incident only to incoming arcs. All routes must originate at node 0 and terminate at node $n+1$.

The set of vehicles $\mathcal{V}$ represents a homogeneous fleet, each with a capacity of q. The number of vehicles, $|\mathcal{V}|$, may be essentially free by allowing as many vehicles in the model as there are clients. Since there is a demand, d_i for each customer $i \in \mathcal{C}$, the length of a route is restricted by the vehicle capacity. Each arc (i, j) in the graph has a cost c_{ij} and a travel time t_{ij}. The travel time from i to j may include the service time at customer i, as well as a measure of the time it takes to travel the distance between the two clients. The service provided at a customer i must begin within a time window denoted as $[a_i, b_i], i \in \mathcal{C}$. The value a_i represents the earliest time at which service may begin for client i; similarly, b_i is the latest time at which service may begin. If a vehicle arrives at a client prior to the beginning of its time window, then it may wait at no cost at the client until service may begin. However, a vehicle may not arrive at a client after the end of the time window. These time restrictions are known as hard time window constraints since they do not allow any time window violations to occur. For the depot, all vehicles must leave node 0 within the time window $[a_0, b_0]$ and return within the time window $[a_{n+1}, b_{n+1}]$. Since the model does not penalize routes based on total time, we may assume without loss of generality that $a_0 = b_0 = a_{n+1} = 0$.

Graph $G = (\mathcal{N}, \mathcal{A})$

- $\mathcal{N}$ = set of nodes: $\{0, n+1\} \cup \mathcal{C}$ where $\{0\}$ and $\{n+1\}$ are the depot and $\mathcal{C} = \{1, 2, \ldots, n\}$ are clients
- $\mathcal{A}$ = set of arcs
- $\mathcal{V}$ = set of vehicles

Constants

- c_{ij} = cost of arc (i, j)
- t_{ij} = travel time along arc (i, j)
- d_i = demand at node $i \in C$
- $[a_i, b_i]$ = earliest and latest time service may begin at node i
- q = vehicle capacity

Variables

- S_i = time service begins at node i
- $x_{ij}^k = \begin{cases} 1 & \text{if vehicle } k \text{ drives from } i \text{ to } j \\ 0 & \text{otherwise} \end{cases}$

Figure 5.1 Notation for the VRPTW

The data is assumed to have certain characteristics. First, the constants q, d_i, c_{ij}, t_{ij}, a_i, and b_i are non–negative integers. In particular, t_{ij} must be strictly positive for reasons related to our solution approach. Our solution methods also require that the triangle inequality in cost and travel, that is $c_{ij} + c_{jk} \geq c_{ik}$, and $t_{ij} + t_{jk} \geq t_{ik}, \forall i, j, k \in \mathcal{N}$, is satisfied. In terms of the original problem, though, the arc costs are permitted to violate the triangle inequality. By adding a scalar to all of the arc costs $c_{ij}, i \in \mathcal{C}$, the costs can be transformed into an instance of the VRPTW that satisfies the triangle inequality in cost without changing the original optimal solution.

This increases the objective function value of any solution by a fixed amount, which is subsequently subtracted from the final solution. Unfortunately, the same cannot be done to the travel times t_{ij}. However, in most instances, especially when service time is included in the travel time, it is a fairly safe assumption that travel time will satisfy the triangle inequality.

The mathematical formulation of the VRPTW requires two types of decision variables. The continuous variables, S_i for $i \in \mathcal{N}$, indicate the time service is scheduled to begin at node i. As previously indicated, the starting time at the depot is assumed to be at time 0. Thus, it is known that $S_0 = 0$. The binary flow variables, x_{ij}^k for $(i, j) \in \mathcal{A}$, $k \in \mathcal{V}$, determine if vehicle k travels from i to j ($x_{ij}^k = 1$) or not ($x_{ij}^k = 0$).

The objective of the mixed integer linear programming formulation is to select the least–cost set of routes that satisfy all of the constraints. The routes must cover the entire set of clients while respecting the vehicle capacity restrictions and the time window constraints. The notation is summarized in Figure 5.1. The MIP formulation is as follows:

$$\text{Minimize} \quad \sum_{k\in\mathcal{V}} \sum_{ij\in\mathcal{A}} c_{ij} x_{ij}^k \tag{5.1}$$

$$\text{subject to} \quad \sum_{k\in\mathcal{V}} \sum_{j\in\mathcal{N}} x_{ij}^k = 1 \qquad \forall i \in \mathcal{C} \tag{5.2}$$

$$\sum_{i\in\mathcal{C}} d_i \sum_{j\in\mathcal{N}} x_{ij}^k \leq q \qquad \forall k \in \mathcal{V} \tag{5.3}$$

$$\sum_{j\in\mathcal{N}} x_{oj}^k \leq 1 \qquad \forall k \in \mathcal{V} \tag{5.4}$$

$$\sum_{i\in\mathcal{N}} x_{ih}^k - \sum_{j\in\mathcal{N}} x_{hj}^k = 0 \qquad \forall h \in \mathcal{C}, \forall k \in \mathcal{V} \tag{5.5}$$

$$\sum_{i\in\mathcal{N}} x_{i,n+1}^k \leq 1 \qquad \forall k \in \mathcal{V} \tag{5.6}$$

$$S_i + t_{ij} - S_j \leq M(1 - x_{ij}^k) \qquad \forall ij \in \mathcal{A}, \forall k \in \mathcal{V}, M \text{ large} \tag{5.7}$$

$$a_i \leq S_i \leq b_i \qquad \forall i \in \mathcal{N} \tag{5.8}$$

$$x_{ij}^k \in \{0, 1\} \qquad \forall ij \in \mathcal{A}, \forall k \in \mathcal{V} \tag{5.9}$$

The objective function (5.1) minimizes cost over the arcs of the vehicles' routes. Constraint set (5.2) ensures that each client is visited by one vehicle. Constraint set (5.3) forces each vehicle to visit only as many clients as its capacity allows. The constraints in (5.4), (5.5), and (5.6) are the flow conservation constraints. They, in combination with the binary constraints (5.9), restrict the solution to valid paths from node 0 to node $n+1$ for each vehicle. Constraint set (5.7) is the linearization of the set $x_{ij}^k(S_i + t_{ij} + S_j) \leq 0$. The scalar M may be any large number. Thus, if vehicle k travels from i to j, then the inequality $S_i + t_{ij} \leq S_j$ must hold; otherwise, the constraint is non–binding because it is trivially true. These constraints, combined with (5.8), guarantee that the S_i variables indicate valid starting times. An additional constraint providing a lower bound on the number of vehicles can easily be added, as well.

5.2 Optimization Method

Formulating the VRPTW as a set partitioning problem has proven to be an effective technique in solving these problems. While there does exist alternate solution approaches for solving the VRPTW (see for example [47] and [33]), no other method has succeeded in solving as many or as diverse of problems as the set partitioning and column generation approach. In particular, Desrochers, Desrosiers, and Solomon [25] successfully solve fifty out of eighty–seven of the VRPTW Solomon benchmark problems [63], including several having 100 customers. Most recently, Kohl et al. [46] succeeded in using this approach with the addition of cutting planes to solve seventy out of the same set of eighty–seven problems.

5.2.1 Set Partitioning Model

The VRPTW can be formulated as a set partitioning problem by defining a set R to be the set of feasible routes for the VRPTW. Then for each route $r \in R$, let a_{ir} be a constant integer equal to the number of times route r visits customer i. The cost of a route, c_r, is defined as the sum of the costs of each arc of the route. Thus, the set partitioning formulation of the VRPTW is stated as:

$$\text{Minimize} \quad \sum_{r \in R} c_r y_r \tag{5.10}$$

$$\text{subject to} \quad \sum_{r \in R} a_{ir} y_r = 1 \qquad \forall i \in C \tag{5.11}$$

$$y_r \geq 0, \ \text{integer} \quad \forall r \in R. \tag{5.12}$$

Clearly the set of feasible routes R is extremely large for all but very small–sized problems. Thus, the set partitioning problem is only the *master problem*, while a *subproblem* is solved to obtain those routes which are useful within the set partitioning model. Since each route is represented by the column of contants $a_{\cdot r}$, the process of generating new routes is called *column generation*. A route not contained in the set R is desirable if its reduced cost in the master problem is strictly negative, that is, if $c_r - \sum_{i \in C} \pi_i a_{ir} < 0$, where π_i is the dual variable associated with the constraint requiring that customer i is visited. The LP relaxation of the set partitioning problem is found by generating negative reduced cost columns as needed and adding them to the set R. The LP relaxation generally provides an excellent lower bound which is imbedded into a branch–and–bound scheme to solve the integer set partitioning problem. Bramel and Simchi–Levi [10] explain this very good behavior by demonstrating that the relative gap between fractional and integer solutions of the set partitioning formulation of the VRPTW decreases to zero as the number of customers increases.

Oftentimes, the set partitioning formulation is tightened by adding a simple constraint requiring a minimum number of vehicles (or routes) in the solution. A trivial lower bound on the number of vehicles is the total demand over all customers divided by the vehicle capacity, rounded up. The constraint is written as

$$\sum_{r \in R} y_r \geq \lceil \sum_{i \in C} d_i / q \rceil$$

where $\lceil \alpha \rceil$ is the smallest integer greater than or equal to α. This strengthened set partitioning formulation thus contains a constraint at the depot node limiting the total number of routes, as well as a constraint corresponding to each customer as in (5.11). The reduced cost associated with a route in this version of the problem is $c_r - \sum_{i \in (\mathcal{C} \cup \{0\})} \pi_i a_{ir}$, where π_i is defined as above for $i \in \mathcal{C}$ and π_0 is the dual variable associated with the lower bound constraint on the number of vehicles. From this point on, any reference to the set partitioning formulation used in this work is assumed to include (or permit) the use of this additional constraint.

5.2.2 Column Generation

The exact subproblem for the set partitioning model is known as the elementary shortest paths problem with time windows and capacity constraints, or ESPPTWCC. The cost of a path in the subproblem must correspond to the reduced cost of the path (or route) in the master problem. Thus, the arc cost of using arc (i, j) in the subproblem is $\hat{c}_{ij} = c_{ij} - \pi_i$. To generate the relevant columns, the ESPPTWCC seeks to find the shortest path from node 0 to node $n+1$ that does not visit any node more than once and does not violate any time window or capacity constraints. The model can be stated using almost the same data and variables as the VRPTW. The only exceptions are the modified costs of the arcs and the elimination of the index k since only one vehicle is considered:

$$\text{Minimize} \quad \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} \hat{c}_{ij} x_{ij} \tag{5.13}$$

$$
\begin{aligned}
\text{subject to} \quad & \sum_{i \in \mathcal{C}} d_i \sum_{j \in \mathcal{N}} x_{ij} \leq q && && (5.14)\\
& \sum_{j \in \mathcal{N}} x_{0j} = 1 && && (5.15)\\
& \sum_{i \in \mathcal{N}} x_{ih} - \sum_{j \in \mathcal{N}} x_{hj} = 0 && \forall h \in \mathcal{C} && (5.16)\\
& \sum_{i \in \mathcal{N}} x_{i,n+1} = 1 && && (5.17)\\
& S_i + t_{ij} - S_j \leq M(1 - x_{ij}) && \forall i \in \mathcal{N}, \forall j \in \mathcal{N} && (5.18)\\
& a_i \leq S_i \leq b_i && \forall i \in \mathcal{N} && (5.19)\\
& x_{ij} \in \{0, 1\} && \forall i \in \mathcal{N}, \forall j \in \mathcal{N} && (5.20)
\end{aligned}
$$

Unfortunately, there is no efficient solution method known to solve the ESPPTWCC. In fact, the ESPPTWCC is $\mathcal{NP}$–Hard in the strong sense [45]. Therefore, it is necessary to relax the problem by permitting cycles in the paths. This problem, known as the shortest paths problem with time windows and capacity constraints, or SPPTWCC, is also $\mathcal{NP}$–Hard, but is solvable by a pseudo–polynomial dynamic programming algorithm if $t_{ij} > 0$ for all $ij \in \mathcal{N}, i \neq j$ [28]. In the SPPTWCC, a non–elementary path must satisfy time and capacity constraints, but arcs may be used more than once and customers may appear on the same path more than once.

To model the SPPTWCC, the decision variables corresponding to flow and time are replaced by the variables x_{ij}^l and S_i^l. The index l is used to order the arcs: x_{ij}^l is 1 if the arc (i, j) is used as the lth arc, and 0 otherwise; S_i^l is the start of service at customer i if that customer is serviced as the lth customer. The value of S_0^0 is defined to be zero and l is defined to be in the set $\mathcal{L} = \{1, 2, \ldots |\mathcal{L}|\}$ where $|\mathcal{L}| = \lfloor \frac{b_{n+1}}{min_{ij}(t_{ij})} \rfloor$. (Note that $\lfloor \alpha \rfloor$ is the largest integer not larger than α.) Thus, the size of the set $\mathcal{L}$ is equal to a trivial upper bound on the number of arcs in any route. The mathematical model of the SPPTWCC can now be stated in a format very similar to that of the elementary paths problem:

$$\text{Minimize} \quad \sum_{l\in\mathcal{L}}\sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}} \hat{c}_{ij}x_{ij}^l \tag{5.21}$$

$$\text{subject to} \quad \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}} x_{ij}^1 = 1 \tag{5.22}$$

$$\sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}} x_{ij}^l - \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}} x_{ij}^{l-1} \leq 0 \qquad \forall l \in \mathcal{L} - \{1\} \tag{5.23}$$

$$\sum_{i\in\mathcal{C}} d_i \sum_{l\in\mathcal{L}}\sum_{j\in\mathcal{N}} x_{ij}^l \leq q \tag{5.24}$$

$$\sum_{j\in\mathcal{N}} x_{0j}^1 = 1 \tag{5.25}$$

$$\sum_{i\in\mathcal{N}} x_{ih}^{l-1} - \sum_{j\in\mathcal{N}} x_{hj}^l = 0 \qquad \forall h \in \mathcal{C}, \forall l \in \mathcal{L} - \{1\} \tag{5.26}$$

$$\sum_{l\in\mathcal{L}}\sum_{i\in\mathcal{N}} x_{i,n+1}^l = 1 \tag{5.27}$$

$$S_i^{l-1} + t_{ij} - S_j^l \leq M(1 - x_{ij}^l) \qquad \forall i \in \mathcal{N}, \forall j \in \mathcal{N}, \forall l \in \mathcal{L} - \{1\} \tag{5.28}$$

$$a_i \leq S_i^l \leq b_i \qquad \forall i \in \mathcal{N}, \forall l \in \mathcal{L} - \{1\} \tag{5.29}$$

$$x_{ij}^l \in \{0, 1\} \qquad \forall i \in \mathcal{N}, \forall j \in \mathcal{N}, \forall l \in \mathcal{L} \tag{5.30}$$

Constraint (5.22) ensures that the first arc is used only once, while the constraints in (5.23) state that the lth arc can only be used if the $(l-1)$th arc is used. The combination of these constraints gives a correct numbering of the arcs. The rest of the constraints (5.24) – (5.30) correspond in function to constraints (5.14) – (5.20) in the ESSPTWCC. These constraints are only altered to maintain the validity of the ordering of the arcs provided by the index l. This model is not used explicitly to solve the SPPTWCC; rather, the problem is solved by using a dynamic programming algorithm.

A Dynamic Programming Algorithm for the SPPTWCC

The dynamic programming approach developed by Desrochers [24] is commonly used to solve shortest paths problems with resource constraints, or SPR's. The SPPTWCC is an SPR for which there are two resources: time and capacity. In his paper, Desrochers describes in detail several variants of the algorithm, including a *reaching* and a *pulling* version. For an excellent summary of Desrocher's reaching algorithm specifically for the SPPTWCC, including a section analyzing the algorithm's complexity with regards to different data structures, the reader is referred to Kohl [45]. Since the reaching algorithm is implemented in this thesis, we will briefly describe the algorithm by presenting the main points identified by Kohl [45].

The basic algorithm assumes that times and capacity are integer values and that time is increasing on the arcs (that is, $t_{ij} > 0 \; \forall ij \in \mathcal{N}, i \neq j$). The algorithm then exploits a discretization of time and capacity through the use of the *states* denoted by the triple (i, t, d). Given a state (i, t, d), i denotes the last serviced node (or customer) on the path, t denotes the current time, and d denotes the accumulated demand serviced. The cost associated with a state is given by $c(i, t, d)$. The forward dynamic programming algorithm begins with the initial state $(0, 0, 0)$ and its cost, $c(0, 0, 0)$, equal to zero. The recurrence equation is given by:

$$c(j, t, d) = \min_i \left[\hat{c}_{ij} + c(i, t', d') \mid t' + t_{ij} = t \text{ and } d' + d_j = d\right].$$

The optimal solution's objective function value is given by:

$$\min_{t \leq b_n, d \leq q} c(n+1, t, d).$$

The actual solution is found by *backtracking* through the states.

The total number of states is bounded by $\Omega = \sum_{i \in \mathcal{N}} (b_i - a_i)(q + 1)$. Even though Ω may be very large, in most cases some states will be infeasible and others will be *dominated* by some other state and thus unnecessary to consider. The determination of dominance will be discussed after the overall algorithm is presented. To exploit the fact that not all states will necessarily need to be considered, the algorithm keeps track

Algorithm 5.1 SPPTWCC

Step 0: Initialization – Labels := $\{(0,0,0)\}$, $c(0,0,0) = 0$

```
Repeat
    Step 1: Find label to be treated: (i', t', d'):=Newlabel(Labels)

    if (i' < n + 1) then begin
        Step 2: Treat label
            for j := 1 to n + 1
                if (j ≠ i' and t' + t_ij ≤ b_j and d' + d_j ≤ q) then begin
                    t_new = max(t' + t_ij, a_j)
                    d_new = d' + d_j
                    if (ĉ_ij + c(i', t', d') < c(j, t_new, d_new)) then begin
                        Insertlabel(Labels, {j, t_new, d_new})
                        c(j, t_new, d_new) := c(i', t', d') + ĉ_ij
                    end
                end
    end
Until i' = n + 1
```

Step 3: Stop

of a set of labels corresponding to the states which are necessary to consider. The labels are treated in order of increasing time. When a label is treated it is extended to all other nodes of the network, if the extension is feasible and it decreases the cost. The number of labels starts at one and increases during the algorithm, but never beyond the value of Ω. The structure of the algorithm is described in Algorithm 5.1.

In step 1 of the algorithm the next label to be treated is selected from the set of labels. The function *Newlabel* returns a label (i', t', d') with the minimal value of t' among all the labels such that $i' \neq n + 1$. If no such label exists, then it returns a label with $i' = n + 1$. After selecting a label with $i' < n + 1$, the second step is to treat the label by extending it to as many nodes as possible. The label (i', t', d')

can be extended to the node j (where $j \neq i$) if the path associated with the label can visit node j as the next node on the path without violating any time or capacity constraints. If a feasible extension with a lower cost is found, then the function *Insertlabel* inserts the new label onto the list of labels. If the label already exists, the old label is replaced by the new one. If a label does not exist, it is said to have infinite cost. Note that since the states are considered in order of non–decreasing time, once a state is considered, it is permanently treated. The algorithm is called a reaching algorithm because the algorithm treats all of the successors of one state in a single execution of step 2.

Dominance

The amount of computational work involved in implementing Algorithm 5.1 may be greatly reduced by applying a dominance criterion which identifies labels which need not be considered. For instance, consider the two labels (i, t, d) and (i, t', d') such that $c(i, t, d) \leq c(i, t', d')$, $t \leq t'$, and $d \leq d'$. Note that both labels correspond to paths ending at node i and that any feasible extension of the second label is also feasible for the first label. And since the cost of the first label is less than or equal to that of the second, the first label (i, t, d) will yield at least as good of a path as the second label (i, t', d'). Therefore, the second label can be discarded and its possible extensions ignored. This relationship is contained in the following formal definition.

> **Definition:** (i, t, d) *dominates* (i, t', d') if and only if $c(i, t, d) \leq c(i, t', d')$ and $t \leq t'$ and $d \leq d'$.

If equality holds in all three relations, then only one of the labels may be discarded, even though the definition would imply that each dominates the other. It is possible that no label will dominate another label, but computational experiments show that in practice dominance will occur among labels, allowing for a reduction in the total number to be considered [45].

The dominance criterion is applied in step 2 of the algorithm. It must be determined whether the new label is dominated by any existing labels, and (if this is not the case), whether the new label dominates any existing label. Thus, it may be necessary to compare the new label with all other labels. This is done by storing the set of labels as $|\mathcal{N}|$ linked lists (one list of labels for each node) sorted in order of time first and then quantity. Using such a data structure, only one pass through the list corresponding to the relevant node is required to make all of the necessary dominance comparisons. Until the correct position for the new label has been found, a check to see if the new label is dominated by the existing labels must be done. Assuming that the new label is not discarded and that the correct position is found, then the subsequent labels in the list must be checked to see if any of them are dominated by the new label. Thus, the tests for dominance are easily incorporated into the Insertlabel function. The overall complexity of the linked list implementation of the algorithm is $O(\Omega^2)$.

2–cycle elimination

The non–elementary paths in the solution space of the SPPTWCC are those paths that contain a cycle. A 2–cycle, which takes the form $[\cdots i - j - i \cdots]$, is the most common form of cycle found in such paths. To reduce the number of non–elementary paths that are generated, the reaching algorithm can be modified to eliminate these 2–cycles. Eliminating 2–cycles from shortest paths algorithms is a standard procedure that was first presented by Houck, Picard, Queyranne, and Vemuganti [41]. In our problem, 2–cycle elimination is equivalent to adding the following constraint to the SPPTWCC model:

$$x_{ij}^{l-1} + x_{ji}^{l} \leq 1 \quad \forall i \in \mathcal{N}, \forall j \in \mathcal{N}, \forall l \in \mathcal{L} \tag{5.31}$$

To describe the modified algorithm, the 4–dimensional label $(i, t, d, pred)$ is required. The *pred* component refers to the predecessor node of the label, that is the node that precedes i in the path that corresponds to the label. To extend the notion of dominance to the 4–dimensional label, the following definitions are needed:

Definitions:

A. The label $(i, t, d, pred)$ is *strongly dominant* if it is not dominated by any other label and at least one of the following two conditions are satisfied:

1. $t + t_{i,pred} > b_{pred}$
2. $d + d_{pred} > q$

B. The label $(i, t, d, pred)$ is *semi–strongly dominant* if it is not dominated by any other label and neither condition 1 or 2 is satisfied.

C. The label $(i, t, d, pred)$ is *weakly dominant* if it is only dominated by semi–strongly dominant labels $(i, t', d', pred')$ for which $pred'$ is the same and $pred' \neq pred$.

A strongly dominant label corresponds to a path that cannot feasibly be extended to its predecessor. In other words, an extension of such a label will never yield a 2–cycle. A semi–strongly dominant label corresponds to a path that could feasibly be extended to its predecessor if 2–cycles were permitted. A weakly dominant label keeps track of the best path not having the same predecessor as the best path. Thus, since we do not want to extend a semi–strongly dominant label to its predecessor, the weakly dominant label is extended only to the predecessor of the semi–strongly dominant label. In summary, for each state (i, t, d) there is either

1. no label, or
2. one strongly dominant label, or
3. one semi–strongly dominant label and possibly one weakly dominant label.

Since the use of the 4–dimensional labels increases the number of labels by at most two, the computational complexity of the algorithm does not change. Since labels are no longer necessarily discarded if they are dominated by some other label, the Insertlabel function in step 2 becomes more complicated. If an existing (or old) label dominates the new label, then the new label can be discarded in the following cases:

- if the old label is strongly dominant,
- if the old label is weakly dominant,
- if the old label and the new label have the same predecessor,
- if the new label cannot be extended to the predecessor of the old label,
- if the new label is dominated by two (or more) labels with different predecessors.

If, on the other hand, the new label dominates an existing label, then the old label is either discarded or its status is changed to weakly dominant.

5.3 Valid Inequalities

After solving the LP relaxation of the VRPTW using the set partitioning formulation with column generation, it is possible to search for violated valid inequalities. As explained in chapter 2 of this thesis, adding violated valid inequalities, or cutting planes, to the LP relaxation can tighten the relaxation and improve the bound. The goal of the branch–and–cut method is to tighten the LP bound as much as possible before branching.

5.3.1 Incorporating Valid Inequalities in Column Generation

All inequalities which we consider for the VRPTW can be stated in the form:

$$\sum_{k \in \mathcal{V}} \sum_{(i,j) \in \mathcal{A}} \alpha_{ij}^k x_{ij}^k \geq \gamma.$$

Since the vehicles are all identical, it is natural to simplify the valid inequality by aggregating over the vehicles (that is, $x_{ij} = \sum_{k \in \mathcal{V}} x_{ij}^k$). This gives us the simpler form:

$$\sum_{(i,j) \in \mathcal{A}} \alpha_{ij} x_{ij} \geq \gamma. \tag{5.32}$$

If x^* represents the aggregated flow over all of the arcs in the current LP solution, then the separation problem consists of finding α_{ij} and γ such that (5.32) is valid and $\sum_{(i,j) \in \mathcal{A}} \alpha_{ij} x_{ij}^* < \gamma$. Once found, the violated valid inequality must be included in the master problem by adding an extra row to the linear program. In order to rewrite (5.32) in the appropriate context for the set partitioning formulation, consider the following definitions. Let f_{ijr} be the number of times arc $(i,j) \in \mathcal{A}$ is used in route r. Then $x_{ij} = \sum_{r \in R} y_r f_{ijr}$. Substituting for x_{ij}, the valid inequality can be rewritten as $\sum_{r \in R} y_r \sum_{(i,j) \in \mathcal{A}} \alpha_{ij} f_{ijr} \geq \gamma$. Finally, letting $b_r = \sum_{(i,j) \in \mathcal{A}} \alpha_{ij} f_{ijr}$ the cutting plane can be added to the master problem in the form

$$\sum_{r \in R} b_r y_r \geq \gamma.$$

Adding a constraint of this form to the master problem changes the costs of the subproblem, but not the structure. For example, the new reduced costs for the SPPTWCC after adding a single constraint to the master problem can be calculated using the formula $\hat{c}_{ij} = c_{ij} - \pi_i - \alpha_{ij}\mu$ where μ is the dual variable associated with the added constraint.

5.3.2 Known Valid Inequalities for the VRPTW

Derived from the literature on the TSP, the most commonly known valid inequalities for the VRPTW are the subtour, comb, and blossom inequalities. Adding subtour inequalities of the form $x(\delta(S)) \geq 1$ for $S \subseteq \mathcal{C}$ is a very effective means of tightening the LP relaxation. This statement is supported by both the work presented in Kohl et al. [46] and the computational work presented in this thesis. Although a form of the comb inequality, as well as the more specific blossom inequality, is valid for the VRPTW, our testing of the addition of these constraints yielded very little, if any, improvement in the LP relaxation. The slightly altered form of the comb inequality that is valid for this problem can be given as $\sum_{i=1}^{|T|} x(\delta(T_i)) + x(\delta(H)) \geq 1 + 3|T|$ where $|T|$ is the number of teeth, $T_i \subseteq \mathcal{C}$ is the set of nodes in each tooth i, and $H \subseteq \mathcal{N}$ is the set of nodes in the handle. The separation routines for the subtour and comb inequalities discussed in the previous chapter are easily modified to apply to the VRPTW. For further details, the reader is referred back to section 4.4 and Cook, Cunningham, Pulleyblank, and Schrijver [21].

Another known valid inequality for the VRPTW is the k–path cut. As in chapter 4, the inequality is written as $x(\delta(S)) \geq k(S)$ for $S \subseteq \mathcal{C}$, where $k(S)$ is the fewest number of vehicles needed to provide service to the clients contained in the set S. This valid inequality is called a k–path inequality because it requires that at least k paths enter (and leave) the set S in any feasible integer solution. The concept of the k–path inequality for the VRPTW is introduced in Kohl et al. [46], but only the cases where $k = 1$, which is equivalent to the subtour inequality, and where $k = 2$, called a 2–path inequality, are determined to be computationally tractable.

Two different separation routines are presented in Kohl et al. [46] to identify violated 2–path inequalities. The first is an heuristic approach. The authors show that if the network is acyclic, then their heuristic actually finds all violated 2–path inequalities. Otherwise, the heuristic fails to consider those sets S such that $x(\delta(S)) < 2$, but $x(\delta(S \setminus \{r\})) \geq 2, \forall r \in S$. The second method is an exact procedure which will always find all of the 2–path cutting planes, but at a much higher computational cost. In fact, computational results are only provided using the heuristic separation routine.

5.4 A New Separation Routine for 2–Path Cuts

In this section, we present a new separation algorithm for finding violated 2–path inequalities in the VRPTW. Our method is as computationally efficient as that of Kohl et al. [46], but with the added advantage that it is possible, though not guaranteed, that the routine may generate all of the violated 2–path inequalities. Although our process for finding these 2–path cuts in the VRPTW is almost exactly the same as the one presented for the home health care problem in chapter 4, for completeness, we present the algorithm here in its entirety.

Given a solution x^* to the LP relaxation of the VRPTW, the set $S \subseteq \mathcal{C}$ is a 2–path cut if the following two conditions are satisfied:

1. $k(S) \geq 2 \Rightarrow S$ represents a valid 2–path inequality, and
2. $x^*(\delta(S)) < 2 \Rightarrow$ the current solution violates the 2–path inequality.

In other words, the goal is to identify sets of clients that require at least two vehicles to be serviced, but in the current fractional solution are serviced by strictly less than two vehicles.

To determine if $k(S) > 1$ (or equivalently, $k(S) \geq 2$), we apply the same 2–step approach as that of Kohl et al [46]. The first step is to check if there is sufficient vehicle capacity on a single vehicle to visit all of the clients in set S. If not, then clearly more than one vehicle is needed and it becomes unnecessary to consider step 2. The check on capacity is performed in linear time by taking the sum of demands for the clients in S. The second step is more complicated, as it requires determining if there exists a feasible route that leaves from the depot, visits every client in S once, and returns to the depot. This is exactly a TSPTW–feasibility problem. Despite the fact that this is an $\mathcal{NP}$–hard problem, so long as the size of S is relatively small, the problem can be solved by dynamic programming (see Dumas et al. [29]). Thus, if it is concluded that the TSPTW is infeasible for a set S, then clearly more than one vehicle is required for the set S, implying that the corresponding 2–path inequality is valid. Thus, we have a fast, though not polynomial, algorithm that determines if $k(S) > 1$.

Identifying the sets S of clients which currently have less than two paths visiting them requires first building an undirected graph $G' = (\mathcal{N}, E)$ from the current solution x^* of the LP relaxation. The graph's edges, E, represent the cumulative flow in either direction on all vehicles between a pair of nodes. If there is no flow between a pair of nodes, then there is no edge connecting the nodes. Otherwise, each undirected edge ij has a capacity, or weight, equal to the positive value: $\sum_{k \in \mathcal{V}} (x_{ij}^k + x_{ji}^k)$. Recall that in an undirected graph, the notation $\delta(R)$ for a set of nodes R is defined as the set of edges $\{e \in E : e$ has an end in R, and an end in $V \backslash R\}$. The weight of a set of edges, F, is denoted $w(F)$. Since the x^* solution satisfies the condition that the flow into a set of clients equals the flow out of the set, then clearly the value, or weight, of a cut $\delta(R)$, where R is a set of clients, is double the value of the corresponding directed cut in the original graph. Thus, identifying a set of clients S such that the condition $x^*(\delta(S)) < 2$ holds in the original solution amounts to finding a set in G' satisfying the condition that $w(\delta(S)) < 4$.

If the graph G' is built from an x^* solution which does not contain any violated subtour inequalities, then the minimum value of any cut in the graph is 2. Thus, our goal is to find every set S of clients in the graph G' such that the value of the cut is less than double the minimum cut. A random contraction algorithm developed by Karger [42] is used to accomplish this task. In an implementation–oriented paper by Karger and Stein [43], the authors prove that using Karger's algorithm on an undirected graph with n vertices finds all cuts with weight within a multiplicative factor α of the minimum cut in $O(n^{2\alpha}log^3n)$ time. We use this algorithm with $\alpha = 2$ to generate the relevant sets S such that $S \subseteq P, |S| > 1$, and $w(\delta(S)) < 4$. Note that if $S_1 \subset S_2$ and both sets must be visited by at least 2 vehicles, then the smaller set is, in a sense, the minimal set. It is minimal since it is possible that once the valid inequality from this set is added to the formulation, the larger set may no longer violate a 2–path inequality. Thus, each set S generated by Karger's algorithm must be checked to determine if $k(S) \geq 2$, unless the set S contains another set in Γ that has already been determined as valid.

Algorithm 5.2 2–path Separation Routine

(1) Build undirected graph G' from x^* vector.

(2) Use Karger's algorithm with $\alpha = 2$ to find $\Gamma = \{S : S \subseteq \mathcal{C}, |S| > 1, w(\delta(S)) < 4\}$.

(3) Sort the sets in Γ from smallest to largest.

(4) For each $S \in \Gamma$, if S does not contain a known valid set $S_1 \in \Gamma$ corresponding to a 2–path inequality and $k(S) \geq 2$ then add S to list of valid inequalities.

The implementation of Karger's algorithm that is used in this work was written by a fellow Rice Ph.D. student Sanjeeb Dash. The implementation is based on Karger and Stein's [43] paper. The premise of the algorithm is rooted in the simple notion of edge contraction. Given a graph of size n, the random process of selecting an edge with probability proportional to its weight and then contracting it, can be repeated until only 2 vertices remain in the graph. The remaining vertices define a cut in the graph. This is called the *Random Contraction Algorithm.* As edges are contracted, the probability that the relevant cut survives a given contraction decreases as the graph gets smaller. Karger and Stein observe that the algorithm can be improved further by recursively calling itself on partially contracted graphs. By stopping the contraction algorithm early, and then considering different ways to contract the graph from that point, the probability of the overall success of the algorithm increases. In the version written by Sanjeeb Dash, once a graph is reduced to a small number of nodes, we used the value 4, then all possible cuts are enumerated. For most applications though, running the random contraction algorithm the necessary number of iterations to achieve the goal of probabilistically generating all of the desired sets is too costly. Thus, Karger's algorithm is implemented as a heuristic by reducing the number of calls to the random contraction algorithm. In our code, we found that computationally, setting the number of iterations at $3n^2$ consistently worked well with our data.

Since Karger's algorithm generates the relevant sets in polynomial time, it is reasonable to expect the overall separation routine to be relatively fast, so long as the TSPTW problem can be solved efficiently. The exact dynamic programming approach to solving the TSPTW is clearly capable of very quickly solving problems, so long as the size of S is small (less than 10–15 nodes), or the time windows are tight [29]. If the time windows are tight, much larger instances may be solved quickly, as well. It is not always necessary, though, to find the optimal solution for the TSPTW. If a feasible solution is known to exist, then that is sufficient information to establish that $k(S) = 1$. Thus, one may use a heuristic to find a feasible solution for the TSPTW. If the heuristic fails to find a solution, then the optimization method is needed to either prove that the problem is infeasible or find a feasible solution. The heuristic used in this work is a very simple insertion heuristic for the TSPTW. While using the heuristic prior to the dynamic programming algorithm is optional, preliminary tests indicate significant savings in terms of time are achieved by doing so. In summary, the following steps are taken to determine if $k(S) > 1$:

1. If $\sum_{i \in S} d_i > q$, then $k(S) > 1$; otherwise goto step 2.

2. If TSPTW heuristic finds a feasible solution, then $k(S) = 1$; otherwise goto step 3.

3. If TSPTW dynamic programming algorithm determines the problem is infeasible, then $k(S) > 1$; otherwise $k(S) = 1$.

5.5 A Separation Routine for k–Path Cuts

While the validity of the k–path inequality for $k \geq 3$ is clear, prior to this work, the implementation of a separation routine for these inequalities appeared to be unreasonable. However, by modifying the new 2–path cutting plane approach outlined in section 5.4, a promising separation routine for k–path inequalities where $k \geq 3$ emerges.

Clearly, Karger's algorithm can be applied to generate the relevant sets S for any size k. By setting $\alpha = k$, sets S for which $x(\delta(S)) < k$ are found. Thus, the only other modification of the cutting plane algorithm must be to the routine for checking if $k(S) \geq k$. Once again, the sum of the demands associated with the clients in set S is found in linear time. It is then necessary to determine if $k - 1$ vehicles have a total capacity large enough to handle the sum of the demands. If so, then it must be determined if there exists a set of no more than $k - 1$ routes which cover the set S. This is no longer a TSPTW–feasibility problem, but rather a VRPTW–feasibility problem. In this smaller VRPTW instance, the objective is to minimize the number of vehicles required, rather than the total distance traveled. Thus, an optimization method for the VRPTW must be called from within the cutting plane procedure on a smaller instance of the VRPTW with a modified objective function. If the problem is determined to be infeasible, or the minimum number of vehicles required is greater than or equal to k, then the set corresponds to a valid k–path inequality.

One obvious choice for an optimization method for the smaller VRPTW is the set partitioning and column generation method being used to solve the initial problem. Since such an approach may not always provide a fast solution to the smaller VRPTW, the separation routine can be transformed into a heuristic by enforcing a limit on the amount of time spent on solving any single smaller VRPTW. If a solution to the smaller instance is not determined within such a time limit, then the set is discarded, and the next set is considered.

Algorithm 5.3 k–Path Separation Routine

(1) Build undirected graph G' from x^* vector.

(2) Use Karger's algorithm with $\alpha = k$ to find $\Gamma = \{S : S \subseteq \mathcal{C}, |S| \geq k, w(\delta(S)) < 2 * k\}$.

(3) Sort the sets in Γ from smallest to largest.

(4) For each $S \in \Gamma$, if S does not contain a known valid set $S_1 \in \Gamma$ corresponding to a valid k–path inequality and $k(S) \geq k$ can be determined, then add S to the list of valid inequalities.

As in the 2–path case, the optimization method for solving the smaller VRPTW is actually only necessary if a heuristic is unable to generate a feasible solution for the instance. Thus, the heuristic TSPTW solution method is applied first, since a feasible solution to the TSPTW clearly indicates that the corresponding k–path inequality where $k \geq 2$ is not valid. This will typically work well on the very small sets contained in Γ. For a set S which can not be visited by a single vehicle, a heuristic for the VRPTW is needed. We use two basic heuristics that are described in detail by Solomon [63]. The first is the nearest neighbor algorithm with the parameters $(\delta_1, \delta_2, \delta_3)$ defined as $(0.4, 0.4, 0.2)$. The second heuristic is called *Insertion–Criterion (i)*, abbreviated as I1. For this heuristic, the following sets of parameters $(\mu, \delta, \alpha_1, \alpha_2)$ are used: $(1,1,0,1), (1,1,1,0), (1,2,1,0)$, and $(1,2,0,1)$. If any of these heuristics generates a feasible solution to the smaller VRPTW with $k-1$ or fewer vehicles, then $k(S) < k$ and the current set is not a valid k–path inequality. While testing the optimization method for the smaller VRPTW, we found that computationally, solving the VRPTW instance with the objective of minimizing the number of vehicles was far more time consuming than using the objective of minimizing total distance traveled. Therefore, we optimized with the latter objective first, as an additional heuristic before finally having to call the optimization algorithm with the modified objective function to determine if the set actually yields a k–path cut.

5.6 Solomon Benchmark Problems

Our computational study is done on the set of well–known Solomon problems [63]. These data sets have become an accepted standard benchmark for testing exact and heuristic solution procedures for the VRPTW. Despite some criticism for not being realistic, they do represent a reasonable variation of both geography and time window width.

There are two problem sets within the Solomon problems. Solutions to problems in set 1 tend to be limited to approximately 5–10 customers per route, due to time and capacity constraints. Alternately, in set 2, solutions may assign more than 30 customers to a route. Within each of these two sets, there are three types of problems. The types are differentiated based on how the coordinates for each customer are determined. The r–problems refer to a set of customers for which location is determined randomly. The c–problems contain customers that are located in clusters. And finally, the rc–problems assign the location of some of the customers randomly, and the others in clusters. Each problem has 100 customers, though smaller instances are made by taking only the first 25 or 50 customers. Within each of the three types of problems, the coordinates for the customers are the same, but the width of the time windows is varied. The c–problems are considered to be the easiest and least realistic type. Problem set 2 is generally considered to contain more difficult optimization problems, and for this reason these problems have not yet been considered by optimization methods. For this study, we consider only those problems in set 1.

The customer locations in the Solomon problems are defined as (x, y)–coordinates, but the exact distance formula, needed to calculate cost and travel time, is not specified. Solutions in the literature have therefore varied with the different conventions. For our computational study, we follow the method of calculation used in [46] and [45]. Cost, or equivalently distance, and travel time are calculated with one decimal point and truncation. Thus, for customers i and j with coordinates (x_i, y_i) and (x_j, y_j), the cost is calculated as

$$c_{ij} = \frac{\lfloor 10\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \rfloor}{10}.$$

A small scalar of 0.1 can than be added to all of the costs to ensure that the triangle inequality in cost holds. This makes the objective function value of any solution increase by $\frac{|\mathcal{C}|}{10}$. All of the final objective function values are reported after subtracting this value from the inflated objective. Travel time is calculated as $t_{ij} = c_{ij} + st_i$ where st_i is the service time of customer i. In the computer program, c_{ij} and t_{ij} are multiplied by 10 and stored as integers. Thus, to ensure the appropriate integer data in the SPPTWCC subproblem, the time window parameters must be multiplied by 10 as well.

5.7 Implementation Issues

Computational experiments are performed on a 300 MHz Pentium II with 256 megabytes of memory and all times are reported not in CPU time, but in total elapsed time. All of the algorithms are implemented in the ANSI C programming language and compiled with the –O (optimization) option. The CPLEX callable library [22], version 5.0 is used to solve all of the linear programming problems. The primal simplex algorithm is used to solve the LPs within the set partitioning and column generation scheme, and the dual simplex algorithm is used to re–optimize after adding cutting planes and before generating more columns. As is the standard in the vehicle routing literature, solutions which the LP solver claims to be optimal are assumed to be optimal and are not verified with exact arithmetic.

As a preprocessing technique, standard time window reductions [25] are applied at the outset. These reductions serve to tighten the time window at a node as much as possible based on the earliest and latest possible arrival and departure times from all other nodes.

In the column generation phase of the algorithm, a limit of at most 20 columns is placed on the number of columns to be added from each subproblem solved. The limit is necessary since adding too many new columns to an LP will make the problem more difficult to optimize. Generated columns are required to have a reduced cost of less than -10^{-4}.

The cutting strategy checks for subtour cuts first by calling a maxflow algorithm. If none exist, then the 2–path separation routine is called. If no 2–path cuts are found, then the 3–path separation routine is called, and so on in this way until whatever value of k is specified by the user. The largest value of k for which we present complete results is three. Some experiments applying a global cutting strategy at nodes within the branching tree were performed, but the results indicate that the additional time required to do so is not accompanied by a significantly smaller search tree. Thus, in most of the computational results, cutting is only applied at the top of the branching tree. We also experimented with adding blossom and comb inequalities, but discovered that very few were found and even fewer improved the lower bound. None of these results are provided since they did not enhance the algorithm in any way.

If an integer solution is not obtained after solving the LP relaxation and generating cuts, then branching is applied. We apply the same branching rules as in Kohl et al. [46]. At each node in the search tree, branching is performed on the number of vehicles, if fractional; otherwise, branching is performed on the value of a single arc ($x_{ij} = \sum_{k \in \mathcal{V}} x_{ij}^k$). The branching arc is selected as the arc maximizing $\{c_{ij}, \min(x_{ij}, 1 - x_{ij})\}$. In terms of the set partitioning model, if $x_{ij} = 0$, then the cost of each route which uses arc (i, j) is penalized, and the arc is eliminated from the subproblem network. If $x_{ij} = 1$, then the cost of each route visiting i or j without using arc (i, j) is penalized, and the arc (i, j) is made to be the only feasible way of leaving node i or visiting node j in the subproblem. For more details on branching within the set partitioning and column generation model, see [25] and [45].

The proposed optimization method discussed thus far, also lends itself to a parallel implementation in terms of evaluating the sets inside the 2– and k–path separation routine and the processing of nodes in the branching tree. We used the software package TreadMarks [1], a *distributed shared memory* (DSM) system, to run a parallel version of the code on a network of 300 MHz Pentium II machines, each with 256 megabytes of memory. Results are provided for the sequential version (run on one machine), as well as the parallel version run on four, eight, sixteen, and thirty–two machines.

5.8 Numerical Results

The sequential version of the algorithm using only subtour and 2–path cutting planes prior to branching succeeds in solving 73 out of 87 of the Solomon benchmark problems. This is four more than the number solved using the standard settings for the heuristic 2–path separation routine implemented by Kohl et al. [46]. Three of these, specifically $r103.100, r106.100$, and $r112.50$, were previously unsolved in the literature. The fourth, $r111.50$, was solved by Kohl et al [46] by modifying a parameter specifically for that problem, whereas the new algorithm solves the problem with the standard settings unchanged. Tables 5.1, 5.2, and 5.3, report the statistics for each problem set using the 2–path routine. The column designated LB(1) refers to the value of the LP relaxation prior to the addition of valid inequalities. The value of the lower bound after all possible valid inequalities have been inserted into the LP is denoted as LB(2). The optimal integer objective value is given as IP(opt). In those instances where cutting planes are added to the LP, the "gap cl." column indicates the proportion of the gap between IP(opt) and LB(1) closed by the insertion of valid inequalities. The number of nodes in the search tree is provided in the "nodes" column, and the number of valid inequalities added to the LP is given in the "vi" column. The total number of columns in the final LP is provided in the "ncols" column. Finally, the computation time in total elapsed seconds is displayed in the last column. For those problems corresponding to blank rows, the code is unable to find the optimal solution within a time limit of 50,000 seconds. Those problems that were previously unsolved in the literature that our algorithm solved optimally are printed in boldface.

Problem	LB(1)	LB(2)	IP(opt)	gap cl.	nodes	vi	ncols	time(s)
r101.25	617.100	617.100	617.100		1	0	84	0.36
r101.50	1043.367	1044.000	1044.000	1.000	1	1	292	1.67
r101.100	1631.150	1634.000	1637.700	0.435	9	1	1360	37.51
r102.25	546.333	547.100	547.100	1.000	1	2	152	0.43
r102.50	909.000	909.000	909.000		1	0	418	1.43
r102.100	1466.600	1466.600	1466.600		1	0	1320	34.92
r103.25	454.600	454.600	454.600		1	0	218	0.47
r103.50	765.950	767.300	772.900	0.194	49	2	2051	47.37
r103.100	**1206.312**	**1206.376**	**1208.700**	**0.027**	**39**	**2**	**4860**	**741.60**
r104.25	416.900	416.900	416.900		1	0	277	1.02
r104.50	616.500	620.758	625.400	0.478	74	12	3382	489.87
r104.100					1	0		
r105.25	530.500	530.500	530.500		1	0	124	0.17
r105.50	892.120	893.650	899.300	0.213	8	2	609	5.51
r105.100	1346.142	1349.316	1355.300	0.347	71	7	2892	251.37
r106.25	457.300	465.400	465.400	1.000	1	1	236	0.54
r106.50	791.367	793.000	793.000	1.000	1	5	499	4.19
r106.100	**1226.440**	**1227.404**	**1234.600**	**0.118**	**760**	**4**	**10781**	**13975.23**
r107.25	422.925	424.300	424.300	1.000	1	2	278	0.89
r107.50	704.438	705.880	711.100	0.216	42	3	2262	67.07
r107.100								
r108.25	396.139	396.720	397.300	0.500	2	2	305	2.21
r108.50								
r108.100					1	0		
r109.25	441.300	441.300	441.300		1	0	192	0.25
r109.50	775.096	776.231	786.800	0.097	140	8	1955	91.88
r109.100					1	0		
r110.25	437.300	437.938	444.100	0.094	16	4	603	4.13
r110.50	692.577	694.150	697.000	0.356	3	3	666	12.16
r110.100					1	0		
r111.25	423.787	428.050	428.800	0.850	2	5	362	1.94
r111.50	691.812	692.642	707.200	0.054	199	6	3955	477.35
r111.100					1	0		
r112.25	384.200	385.391	393.000	0.135	7	14	483	6.30
r112.50	**607.219**	**612.389**	**630.200**	**0.225**	**2895**	**15**	**18052**	**43675.00**
r112.100					1	0		
TOTAL TIME:								59932.84

Table 5.1 r–problems: Subtour and 2–path cutting planes added at root node of search tree.

Problem	LB(1)	LB(2)	IP(opt)	gap cl.	nodes	vi	numcols	time (s)
c101.25	191.300	191.300	191.300		1	0	499	0.57
c101.50	362.400	362.400	362.400		1	0	956	1.52
c101.100	827.300	827.300	827.300		1	0	2165	10.64
c102.25	190.300	190.300	190.300		1	0	726	1.83
c102.50	361.400	361.400	361.400		1	0	1150	6.88
c102.100	827.300	827.300	827.300		1	0	2770	64.55
c103.25	190.300	190.300	190.300		1	0	616	3.34
c103.50	361.400	361.400	361.400		1	0	1192	27.00
c103.100	826.300	826.300	826.300		1	0	2873	208.92
c104.25	186.900	186.900	186.900		1	0	570	5.41
c104.50	357.250	357.300	358.000	0.067	4	1	2043	169.22
c104.100	822.900	822.900	822.900		1	0	2736	622.96
c105.25	191.300	191.300	191.300		1	0	445	0.58
c105.50	362.400	362.400	362.400		1	0	1222	2.53
c105.100	827.300	827.300	827.300		1	0	2631	18.39
c106.25	191.300	191.300	191.300		1	0	363	0.27
c106.50	362.400	362.400	362.400		1	0	744	1.30
c106.100	827.300	827.300	827.300		1	0	2069	18.31
c107.25	191.300	191.300	191.300		1	0	472	0.50
c107.50	362.400	362.400	362.400		1	0	1176	3.58
c107.100	827.300	827.300	827.300		1	0	2338	22.50
c108.25	191.300	191.300	191.300		1	0	380	0.69
c108.50	362.400	362.400	362.400		1	0	1142	5.43
c108.100	827.300	827.300	827.300		1	0	2507	38.50
c109.25	191.300	191.300	191.300		1	0	383	1.20
c109.50	362.400	362.400	362.400		1	0	999	9.53
c109.100	825.640	827.300	827.300	1.000	1	3	2003	64.51
TOTAL TIME:								1310.66

Table 5.2 *c*–problems: Subtour and 2–path cutting planes added at root node of search tree.

Problem	LB(1)	LB(2)	IP(opt)	gap cl.	nodes	vi	ncols	time(s)
rc101.25	406.625	461.100	461.100	1.000	1	2	237	0.42
rc101.50	850.021	944.000	944.000	1.000	1	16	604	6.98
rc101.100	1584.094	1616.921	1619.800	0.919	7	23	1568	136.23
rc102.25	351.800	351.800	351.800		1	0	252	0.33
rc102.50	719.902	813.037	822.500	0.908	303	21	5123	671.32
rc102.100					1	0		
rc103.25	332.050	332.050	332.800		3	0	630	1.83
rc103.50	643.133	710.112	710.900	0.988	4	15	1403	44.55
rc103.100					1	0		
rc104.25	305.825	305.825	306.600		5	0	997	3.88
rc104.50	543.750	543.750	545.800		17	0	2855	58.89
rc104.100					1	0		
rc105.25	410.950	410.950	411.300		2	0	413	0.78
rc105.50	754.443	853.675	855.300	0.984	15	17	1432	38.83
rc105.100	1471.160	1509.733	1513.700	0.907	36	16	3288	502.27
rc106.25	342.829	343.200	345.500	0.139	9	1	699	2.09
rc106.50	664.433	716.501	723.200	0.886	7	14	941	29.61
rc106.100					1	0		
rc107.25	298.300	298.300	298.300		1	0	344	0.74
rc107.50	591.476	631.588	642.700	0.783	71	7	2843	139.08
rc107.100					1	0		
rc108.25	293.791	294.500	294.500	1.000	1	4	427	1.62
rc108.50	538.957	587.120	598.100	0.814	8	7	1579	56.80
rc108.100					1	0		
TOTAL TIME:								1696.25

Table 5.3 *rc*–problems: Subtour and 2–path cutting planes added at root node of search tree.

For the r–problems in Table 5.1, all of the 25–customer problems, all but one of the 50–customer problems, and almost half of the 100–customer problems are solved. The easier c–problems in Table 5.2 are all solved and only two of them use valid inequalities in the solution procedure. For the rc–problems in Table 5.3, all of the 25– and 50–customer problems, and two of the 100–customer problems are solved. The proportion of the integrality gap closed by the insertion of valid inequalities is very large, generally larger than 75%, for most of these type of problems. Although it is difficult to compare the computational times reported in Kohl et al. [46] to these results, it appears that the new 2–path separation routine is comparable, if not better, in terms of total time. In order to test this conclusion, we implemented the heuristic 2–path separation routine of Kohl et al. [46] within our code and found that our routine yielded equivalent or faster run times in most instances, as well as the same or better LB(2) values.

To measure the effectiveness of the cutting plane procedure, we attempted to solve the problems without generating any valid inequalities. Since all but one of the problems in Tables 5.1–5.3 finished within $20,000$ seconds, we use this as our time limit for these tests. All of the c–problems are solvable with simply the set partitioning and column generation approach within a branch–and–bound scheme. Given these results, it is clear that adding valid inequalities for these problems is generally unneccesary. Thus, further results on our cutting plane approach for the c–problems are not presented. For the r and rc problem sets, all of the 25–customer problems could be solved without cutting planes. Out of all of the 50– and 100–customer problems for sets r and rc, though, only 14 out of 24 and 5 out of 16 problems, respectively, could be solved without adding cuts. These results, which are similar to those of Kohl et al. [46], indicate that the 2–path cutting planes are most effective on the rc–problems.

Tables 5.4 and 5.5 report the same statistics for the problems when the 3–path cutting plane algorithm is applied as well. In these instances, the time limit on solving each of the smaller VRPTW problems within the 3–path cutting plane routine is set at 600 seconds. The value of LB(2) is improved with the addition of 3–path cuts in six of the r–problems (r101.100, r103.50, r104.50, r105.100, r106.100, and r111.50) and seven of the rc–problems (rc101.100, rc102.50, rc103.50, rc105.100, rc106.50, rc107.50 and rc108.50). Clearly, in those problems which do not yield any 3–path cuts, the additional time spent in the cutting plane portion of the code provides no benefits. Unfortunately, for the thirteen problems that do find 3–path cuts, the improvement of the lower bound prior to branching does not yield similar improvements in time. However, it is clear that the 3–path separation routine can be applied without an unreasonable cost in terms of time. In fact, it is likely that the overall routine would improve with better heuristics and optimization methods for the smaller VRPTW. To our knowledge, there has not been a focus on trying to solve small VRPTWs optimally. The k–path separation routine, though, provides motivation for future research in this area. As it is, out of all of the problems that are solved in Tables 5.4–5.5, only one smaller VRPTW was not solved within the 600 second time limit.

Several other variations on the k–path algorithm are tested as well. The first of which reduces the time limit on the smaller VRPTWs from 600 to 60 seconds. In most cases, the reduced time limit does not alter the algorithm's performance at all. There are three exceptions, though, which can be found in Table 5.6. In two cases, r103.50 and r104.50, a single smaller VRPTW was unable to be solved in each, causing a 3–path cut not to be found and reducing the total number of valid inequalities added to the formulation, the total time spent in the cutting plane routines, and the value of LB(2). In problem rc108.50 though, the failure to solve sixteen smaller VRPTWs

Problem	LB(1)	LB(2)	IP(opt)	gap cl.	nodes	vi	ncols	time(s)
r101.25	617.100	617.100	617.100		1	0	84	0.31
r101.50	1043.367	1044.000	1044.000	1.000	1	1	292	1.66
r101.100	1631.150	1635.550	1637.700	0.672	7	7	1165	117.07
r102.25	546.333	547.100	547.100	1.000	1	2	152	0.42
r102.50	909.000	909.000	909.000		1	0	418	1.44
r102.100	1466.600	1466.600	1466.600		1	0	1320	34.93
r103.25	454.600	454.600	454.600		1	0	218	0.45
r103.50	765.950	767.473	772.900	0.219	27	5	1862	329.49
r103.100	**1206.312**	**1206.376**	**1208.700**	**0.027**	**39**	**2**	**4860**	**838.29**
r104.25	416.900	416.900	416.900		1	0	277	1.08
r104.50	616.500	620.779	625.400	0.481	55	13	2862	822.42
r104.100					1	0		
r105.25	530.500	530.500	530.500		1	0	124	0.15
r105.50	892.120	893.650	899.300	0.213	8	2	609	14.94
r105.100	1346.142	1350.175	1355.300	0.440	72	8	2689	354.62
r106.25	457.300	465.400	465.400	1.000	1	1	236	0.56
r106.50	791.367	793.000	793.000	1.000	1	5	499	4.18
r106.100	**1226.440**	**1227.425**	**1234.600**	**0.121**	**728**	**5**	**10509**	**14398.17**
r107.25	422.925	424.300	424.300	1.000	1	2	278	0.87
r107.50	704.438	705.880	711.100	0.216	42	3	2262	82.45
r107.100								
r108.25	396.139	396.720	397.300	0.500	2	2	305	4.50
r108.50								
r108.100					1	0		
r109.25	441.300	441.300	441.300		1	0	192	0.24
r109.50	775.096	776.231	786.800	0.097	140	8	1955	95.92
r109.100					1	0		
r110.25	437.300	437.938	444.100	0.094	16	4	603	5.95
r110.50	692.577	694.150	697.000	0.356	3	3	666	30.34
r110.100					1	0		
r111.25	423.787	428.050	428.800	0.850	2	5	362	3.60
r111.50	691.812	693.337	707.200	0.099	205	15	4261	930.51
r111.100					1	0		
r112.25	384.200	385.391	393.000	0.135	7	14	483	25.38
r112.50	**607.219**	**612.389**	**630.200**	**0.225**	**2895**	**15**	**18052**	**43559.24**
r112.100					1	0		
TOTAL TIME:								61659.18

Table 5.4 Subtour, 2–path and 3–path cutting planes added at root node of search tree

Problem	LB(1)	LB(2)	IP(opt)	gap cl.	nodes	vi	ncols	time(s)
rc101.25	406.625	461.100	461.100	1.000	1	2	237	0.45
rc101.50	850.021	944.000	944.000	1.000	1	16	604	6.97
rc101.100	1584.094	1617.339	1619.800	0.931	9	29	1546	364.60
rc102.25	351.800	351.800	351.800		1	0	252	0.33
rc102.50	719.902	814.249	822.500	0.920	259	32	5015	1162.20
rc102.100					1	0		
rc103.25	332.050	332.050	332.800		3	0	630	4.20
rc103.50	643.133	710.667	710.900	0.997	3	21	1442	291.77
rc103.100					1	0		
rc104.25	305.825	305.825	306.600		5	0	997	16.11
rc104.50	543.750	543.750	545.800		17	0	2855	449.87
rc104.100					1	0		
rc105.25	410.950	410.950	411.300		2	0	413	2.09
rc105.50	754.443	853.675	855.300	0.984	15	17	1432	85.70
rc105.100	1471.160	1509.800	1513.700	0.908	25	18	3111	640.49
rc106.25	342.829	343.200	345.500	0.139	9	1	699	2.42
rc106.50	664.433	717.930	723.200	0.910	15	22	1327	288.03
rc106.100					1	0		
rc107.25	298.300	298.300	298.300		1	0	344	0.72
rc107.50	591.476	632.336	642.700	0.798	63	10	3011	862.48
rc107.100					1	0		
rc108.25	293.791	294.500	294.500	1.000	1	4	427	1.86
rc108.50	538.957	595.982	598.100	0.964	8	19	1983	1778.69
rc108.100					1	0		
TOTAL TIME:								5958.98

Table 5.5 Subtour, 2–path and 3–path cutting planes added at root node of search tree

actually allowed the algorithm to find a different set of 3–path cuts which lead to an improvement in the value of LB(2) and an increase in the amount of time.

Problem	LB(1)	LB(2)	IP(opt)	gap cl.	nodes	vi	ncols	time(s)
r103.50	765.950	767.300	772.900	0.194	49	2	2051	114.87
r104.50	616.500	620.758	625.400	0.478	74	12	3382	680.34
rc108.50	538.957	596.867	598.100	0.979	8	21	2257	2982.89

Table 5.6 Results using a 60 second time limit on the smaller VRPTWs.

Another variation of the k–path algorithm tests the premise that it is better to optimize the smaller VRPTW with an objective function that minimizes total distance prior to solving the actual problem that minimizes the number of vehicles. As initial results predicted, the performance of this version of the algorithm is significantly worse. In fact, the same set of solved problems in Table 5.5 required $39,851.55$ seconds in this version. This is more than six and a half times as long as the time it takes when using the first optimization problem as a heuristic. This clearly demonstrates the importance of being able to quickly find feasible solutions to the smaller VRPTW instances.

As was mentioned in the implementation section, it is possible to search for global cuts within the branch–and–bound search tree. To limit the cost in terms of time, we only consider adding subtour and 2–path inequalities inside the branching tree. Initial testing of the algorithm, though, made it very clear that cutting at every node of the search tree costs too much time relative to the added value of the cuts that are found. Thus, a strategy that applies the cutting routines at every node of the tree having a depth divisible by five is adopted. The results using this cut frequency of five with $k = 2$ and $k = 3$ on the sets r and rc are provided in Tables 5.7 – 5.10. Notice that Tables 5.7 and 5.9 have smaller total times, but that the algorithm in this form cannot solve problem $r112.50$. A closer analysis shows that although the number of nodes necessary to solve problems does decrease in many cases, the overall times are much greater due to the calling of the cutting plane routine within the search tree.

Two additional problems, $r107.100$ and $r108.50$, were able to be solved by running the parallel version of the 2–path code on four processors. These results are displayed in Table 5.11. The overall time limit in these parallel versions of the code is 30,000 seconds, rather than 50,000 as previously used. The times for the r– and rc–problems using one, four, and eight processors for both 2–path and 3–path cutting plane algorithms are given in Tables 5.12 and 5.13. In all of these instances, both the k–path separation routines and the branching sections of the code involve parallelization. For the separation routines, Karger's algorithm is called on one machine to generate the relevant sets S contained in Γ. Then each set S is sent to an available machine to determine (if possible) whether $k(S) \geq k$. The effectiveness of performing this operation in parallel varies for the different problems. In general, the more cutting planes found, the greater the benefit. In those instances with few or no violated valid inequalities, the extra communication involved in parallelization of the separation routine can actually add time to the solution process. It is also relevant to note that the sequential and parallel versions of the computer code generate slightly different cutting planes since the LP solutions found with CPLEX are not necessarily the same. The cutting plane results for the parallel version, though, are similar to the sequential version's results presented in Tables 5.1 to 5.5.

The amount of computation time spent in the cutting plane routine often increases or remains the same for a problem run on eight machines versus four machines. Though there are clear benefits from parallization of the cutting plane method when using four processors, adding too many machines can actually slow down the computation. For example, if the same algorithm is performed on sixteen processors, the communication and wait time in the parallelized cutting plane routine becomes so

Problem	LB(1)	LB(2)	IP(opt)	gap cl.	nodes	vi	ncols	time(s)
r101.25	617.100	617.100	617.100		1	0	84	0.34
r101.50	1043.367	1044.000	1044.000	1.000	1	1	292	1.67
r101.100	1631.150	1634.000	1637.700	0.435	9	1	1360	51.05
r102.25	546.333	547.100	547.100	1.000	1	2	152	0.44
r102.50	909.000	909.000	909.000		1	0	418	1.37
r102.100	1466.600	1466.600	1466.600		1	0	1320	34.94
r103.25	454.600	454.600	454.600		1	0	218	0.45
r103.50	765.950	767.300	772.900	0.194	31	7	1865	54.22
r103.100	**1206.312**	**1206.376**	**1208.700**	**0.027**	**38**	**3**	**4858**	**928.58**
r104.25	416.900	416.900	416.900		1	0	277	0.98
r104.50	616.500	620.758	625.400	0.478	54	21	3097	645.72
r104.100					1	0		
r105.25	530.500	530.500	530.500		1	0	124	0.17
r105.50	892.120	893.650	899.300	0.213	8	2	609	6.80
r105.100	1346.142	1349.316	1355.300	0.347	55	14	2510	428.05
r106.25	457.300	465.400	465.400	1.000	1	1	236	0.55
r106.50	791.367	793.000	793.000	1.000	1	5	499	4.07
r106.100	**1226.440**	**1227.404**	**1234.600**	**0.118**	**1117**	**25**	**10376**	**20204.58**
r107.25	422.925	424.300	424.300	1.000	1	2	278	0.86
r107.50	704.438	705.880	711.100	0.216	38	7	2078	81.37
r107.100					1	0		
r108.25	396.139	396.720	397.300	0.500	2	2	305	2.35
r108.50					1	0		
r108.100					1	0		
r109.25	441.300	441.300	441.300		1	0	192	0.27
r109.50	775.096	776.231	786.800	0.097	102	15	1990	154.48
r109.100					1	0		
r110.25	437.300	437.938	444.100	0.094	16	4	603	4.48
r110.50	692.577	694.150	697.000	0.356	3	3	666	13.52
r110.100					1	0		
r111.25	423.787	428.050	428.800	0.850	2	5	362	2.14
r111.50	691.812	692.642	707.200	0.054	156	21	4317	913.86
r111.100					1	0		
r112.25	384.200	385.391	393.000	0.135	7	14	483	6.49
r112.50								
r112.100					1	0		
TOTAL TIME:								23543.8

Table 5.7 Subtour and 2–path cutting planes added at root; cuts of $k \leq 2$ added at cut frequency of 5 at nodes of search tree

Problem	LB(1)	LB(2)	IP(opt)	gap cl.	nodes	vi	ncols	time(s)
rc101.25	406.625	461.100	461.100	1.000	1	2	237	0.94
rc101.50	850.021	944.000	944.000	1.000	1	16	604	6.98
rc101.100	1584.094	1616.921	1619.800	0.919	7	23	1568	158.49
rc102.25	351.800	351.800	351.800		1	0	252	0.37
rc102.50	719.902	813.037	822.500	0.908	283	26	5136	807.69
rc102.100					1	0		
rc103.25	332.050	332.050	332.800		3	0	630	1.98
rc103.50	643.133	710.112	710.900	0.988	4	15	1403	46.73
rc103.100					1	0		
rc104.25	305.825	305.825	306.600		5	0	997	4.08
rc104.50	543.750	543.750	545.800		17	0	2855	60.39
rc104.100					1	0		
rc105.25	410.950	410.950	411.300		2	0	413	0.94
rc105.50	754.443	853.675	855.300	0.984	15	17	1432	40.43
rc105.100	1471.160	1509.733	1513.700	0.907	31	18	3260	630.57
rc106.25	342.829	343.200	345.500	0.139	9	1	699	2.20
rc106.50	664.433	716.501	723.200	0.886	7	14	941	31.39
rc106.100					1	0		
rc107.25	298.300	298.300	298.300		1	0	344	0.75
rc107.50	591.476	631.588	642.700	0.783	71	11	3591	212.27
rc107.100					1	0		
rc108.25	293.791	294.500	294.500	1.000	1	4	427	1.91
rc108.50	538.957	587.120	598.100	0.814	8	7	1579	59.16
rc108.100					1	0		
TOTAL TIME:								2067.27

Table 5.8 Subtour and 2–path cutting planes added at root; cuts of $k \leq 2$ added at cut frequency of 5 at nodes of search tree

Problem	LB(1)	LB(2)	IP(opt)	gap cl.	nodes	vi	ncols	time(s)
r101.25	617.100	617.100	617.100		1	0	84	0.35
r101.50	1043.367	1044.000	1044.000	1.000	1	1	292	1.65
r101.100	1631.150	1635.550	1637.700	0.672	7	7	1165	130.08
r102.25	546.333	547.100	547.100	1.000	1	2	152	0.42
r102.50	909.000	909.000	909.000		1	0	418	1.31
r102.100	1466.600	1466.600	1466.600		1	0	1320	32.66
r103.25	454.600	454.600	454.600		1	0	218	0.41
r103.50	765.950	767.473	772.900	0.219	27	5	1862	323.28
r103.100	**1206.312**	**1206.376**	**1208.700**	**0.027**	**38**	**3**	**4858**	**990.28**
r104.25	416.900	416.900	416.900		1	0	277	0.93
r104.50	616.500	620.779	625.400	0.481	64	22	2681	868.63
r104.100					1	0		
r105.25	530.500	530.500	530.500		1	0	124	0.16
r105.50	892.120	893.650	899.300	0.213	8	2	609	16.29
r105.100	1346.142	1350.175	1355.300	0.440	65	19	2435	663.33
r106.25	457.300	465.400	465.400	1.000	1	1	236	0.53
r106.50	791.367	793.000	793.000	1.000	1	5	499	4.20
r106.100	**1226.440**	**1227.425**	**1234.600**	**0.121**	**685**	**13**	**10118**	**18876.57**
r107.25	422.925	424.300	424.300	1.000	1	2	278	0.83
r107.50	704.438	705.880	711.100	0.216	36	9	2039	94.22
r107.100					1	0		
r108.25	396.139	396.720	397.300	0.500	2	2	305	4.53
r108.50					1	0		
r108.100					1	0		
r109.25	441.300	441.300	441.300		1	0	192	0.25
r109.50	775.096	776.231	786.800	0.097	102	15	1990	153.67
r109.100					1	0		
r110.25	437.300	437.938	444.100	0.094	16	4	603	6.23
r110.50	692.577	694.150	697.000	0.356	3	3	666	31.04
r110.100					1	0		
r111.25	423.787	428.050	428.800	0.850	2	5	362	3.72
r111.50	691.812	693.337	707.200	0.099	155	25	4431	1162.24
r111.100					1	0		
r112.25	384.200	385.391	393.000	0.135	7	14	483	24.94
r112.50								
r112.100					1	0		
TOTAL TIME:								23392.75

Table 5.9 Subtour, 2–path, and 3–path cutting planes added at root; cuts of $k \leq 2$ added at cut frequency of 5 at nodes of search tree

Problem	LB(1)	LB(2)	IP(opt)	gap cl.	nodes	vi	ncols	time(s)
rc101.25	406.625	461.100	461.100	1.000	1	2	237	0.60
rc101.50	850.021	944.000	944.000	1.000	1	16	604	6.99
rc101.100	1584.094	1617.339	1619.800	0.931	9	29	1546	386.16
rc102.25	351.800	351.800	351.800		1	0	252	0.32
rc102.50	719.902	814.249	822.500	0.920	232	35	4792	1060.88
rc102.100					1	0		
rc103.25	332.050	332.050	332.800		3	0	630	4.27
rc103.50	643.133	710.667	710.900	0.997	3	21	1442	285.35
rc103.100					1	0		
rc104.25	305.825	305.825	306.600		5	0	997	15.81
rc104.50	543.750	543.750	545.800		17	0	2855	434.11
rc104.100					1	0		
rc105.25	410.950	410.950	411.300		2	0	413	2.25
rc105.50	754.443	853.675	855.300	0.984	15	17	1432	86.48
rc105.100	1471.160	1509.800	1513.700	0.908	26	21	3015	749.86
rc106.25	342.829	343.200	345.500	0.139	9	1	699	2.50
rc106.50	664.433	717.930	723.200	0.910	15	22	1327	281.75
rc106.100					1	0		
rc107.25	298.300	298.300	298.300		1	0	344	0.89
rc107.50	591.476	632.336	642.700	0.798	81	16	3864	968.94
rc107.100					1	0		
rc108.25	293.791	294.500	294.500	1.000	1	4	427	1.79
rc108.50	538.957	595.982	598.100	0.964	8	19	1983	1711.45
rc108.100					1	0		
TOTAL TIME:								6000.4

Table 5.10 Subtour, 2–path, and 3–path cutting planes added at root; cuts of $k \leq 2$ added at cut frequency of 5 at nodes of search tree

Problem	LB(1)	LB(2)	IP(opt)	gap cl.	nodes	vi	ncols	time
r107.100	**1051.844**	**1052.927**	**1064.600**	**0.085**	**2613**	**4**	**16,892**	**18,124.09**
r108.50	**588.926**	**595.624**	**617.700**	**0.233**	**2451**	**20**	**26,730**	**31,516.58**

Table 5.11 Subtour and 2–path cutting planes added at root node of search tree. Cutting and branching performed on 4 processors.

large that the elapsed time increases beyond a reasonable level. Unfortunately, the current version of TreadMarks only permits sequential operations on one processor or parallel operations on all of the available processors. Thus, the code implemented on sixteen processors performs all of the cutting plane routines on a single processor, and then branches using all of the available machines. Using the 3–path cutting routine, the results for the 50– and 100–customer problems in sets r and rc are displayed in Tables 5.14 and 5.15. The savings in terms of time are more significant for the r–problem set, since these problems require the most branching. In fact, an additional problem, $r109.100$ is solved to optimality in just under the time limit of $30,000$ seconds. For the solved instances in the rc–problem set, the addition of violated valid inequalities reduces the size of the required tree so much so that little time was saved by adding additional machines to the branching process.

For the eleven unsolved problems that remain, we ran the algorithm on sixteen processors. This version sets $k = 6$, the time limit on the smaller VRPTWs is 600 seconds, and the overall time limit is 30,000 seconds. To find an initial upper bound on the objective function value, the simple heuristic for the smaller VRPTW is called on the larger problem. The results are displayed in Table 5.16. The LB(3) column indicates the lower bound achieved on a single machine prior to termination. In other words, the lower bound is not strict since a different machine may be working on a node from the heap that has a lower value. It does, however, give a sense of how much the branch–and–bound tree has accomplished. The UB value refers to the best known

Problem	Time (s) for k=2			Time (s) for k=3		
	Number of Processors			Number of Processors		
	1	4	8	1	4	8
r101.25	0.36	0.32	0.32	0.31	0.29	0.33
r101.50	1.67	1.78	1.89	1.66	1.79	1.81
r101.100	37.51	36.45	37.31	117.07	197.60	201.59
r102.25	0.43	0.52	0.54	0.42	0.52	0.53
r102.50	1.43	1.45	1.45	1.44	1.43	1.49
r102.100	34.92	35.55	35.37	34.93	35.63	35.88
r103.25	0.47	0.45	0.45	0.45	0.44	0.46
r103.50	47.37	25.93	24.63	329.49	341.04	309.12
r103.100	**741.60**	**452.06**	**449.81**	**838.29**	**517.53**	**507.08**
r104.25	1.02	1.00	1.04	1.08	1.00	1.02
r104.50	489.87	227.94	190.93	822.42	529.75	605.36
r104.100						
r105.25	0.17	0.15	0.16	0.15	0.15	0.16
r105.50	5.51	5.29	5.58	14.94	17.24	19.24
r105.100	251.37	126.87	114.42	354.62	319.15	299.82
r106.25	0.54	0.69	0.73	0.56	0.73	0.72
r106.50	4.19	4.46	4.57	4.18	4.39	4.48
r106.100	**13975.23**	**2987.87**	**1556.16**	**14398.17**	**3909.46**	**1981.35**
r107.25	0.89	1.02	1.03	0.87	1.01	1.04
r107.50	67.07	36.65	40.52	82.45	59.17	58.92
r107.100		**18124.09**	**8703.21**		**18435.36**	**8293.81**
r108.25	2.21	3.10	3.54	4.50	8.79	9.50
r108.50		**31516.58**	**13468.16**		**30893.75**	**16714.84**
r108.100						
r109.25	0.25	0.25	0.26	0.24	0.24	0.25
r109.50	91.88	40.44	31.86	95.92	53.97	47.48
r109.100						
r110.25	4.13	4.18	4.67	5.95	9.05	9.53
r110.50	12.16	13.52	14.60	30.34	34.76	35.23
r110.100						
r111.25	1.94	2.64	2.81	3.60	7.08	7.51
r111.50	477.35	133.05	87.27	930.51	270.93	279.24
r111.100						
r112.25	6.30	8.74	8.24	25.38	31.49	19.10
r112.50	**43675.00**	**8271.28**	**3456.71**	**43559.24**	**7626.78**	**4085.28**
r112.100						

Table 5.12 r–problems: TreadMarks summary of times.

Problem	Time (s) for k=2			Time (s) for k=3		
	Number of Processors			Number of Processors		
	1	4	8	1	4	8
rc101.25	0.42	0.81	0.83	0.45	0.79	0.84
rc101.50	6.98	7.92	8.35	6.97	7.89	8.10
rc101.100	136.23	131.76	130.88	364.60	587.32	646.36
rc102.25	0.33	0.36	0.38	0.33	0.37	0.38
rc102.50	671.32	185.57	96.00	1162.20	388.50	282.32
rc102.100						
rc103.25	1.83	2.04	2.17	4.20	7.87	8.18
rc103.50	44.55	53.47	60.41	291.77	199.02	143.64
rc103.100						
rc104.25	3.88	3.58	3.66	16.11	12.43	11.33
rc104.50	58.89	39.79	38.37	449.87	166.20	131.09
rc104.100						
rc105.25	0.78	1.00	1.05	2.09	5.30	5.94
rc105.50	38.83	38.87	33.85	85.70	62.71	54.87
rc105.100	502.27	337.30	303.59	640.49	488.10	711.32
rc106.25	2.09	2.04	2.01	2.42	6.67	7.06
rc106.50	29.61	27.59	28.78	288.03	205.76	176.41
rc106.100						
rc107.25	0.74	0.77	0.81	0.72	0.78	0.79
rc107.50	139.08	83.18	54.33	862.48	369.13	117.22
rc107.100						
rc108.25	1.62	1.72	1.73	1.86	1.95	1.75
rc108.50	56.80	65.74	62.56	1778.69	2050.17	2870.54
rc108.100						

Table 5.13 *rc*–problems: TreadMarks summary of times.

Problem	LB(1)	LB(2)	IP(opt)	gap cl.	nodes	vi	ncols	time(s)
r101.50	1043.367	1044.000	1044.000	1.000	1	1		1.98
r101.100	1631.150	1635.550	1637.700	0.672	7	7	1020	118.64
r102.50	909.000	909.000	909.000		1	0		1.46
r102.100	1466.600	1466.600	1466.600		1	0		35.65
r103.50	765.950	767.473	772.900	0.219	38	5	902	309.05
r103.100	**1206.312**	**1206.376**	**1208.700**	**0.027**	**132**	**2**	**3328**	**588.98**
r104.50	616.500	620.779	625.400	0.481	98	13	1478	597.47
r104.100					1	0		
r105.50	892.120	893.650	899.300	0.213	9	2	462	16.37
r105.100	1346.142	1350.175	1355.300	0.440	120	8	1659	226.63
r106.50	791.367	793.000	793.000	1.000	1	5		4.16
r106.100	**1226.440**	**1227.425**	**1234.600**	**0.121**	**738**	**5**	**5184**	**1000.39**
r107.50	704.438	705.880	711.100	0.216	111	3	1369	51.20
r107.100	**1051.844**	**1052.927**	**1064.600**	**0.085**	**2541**	**4**	**10500**	**4309.28**
r108.50	**588.926**	**595.738**	**617.700**	**0.237**	**2214**	**20**	**14942**	**6046.61**
r108.100					1	0		
r109.50	775.096	776.231	786.800	0.097	104	8	870	32.03
r109.100	**1130.587**	**1133.166**	**1146.900**	**0.158**	**11201**	**18**	**17230**	**29712.87**
r110.50	692.577	694.150	697.000	0.356	3	3	672	31.77
r110.100					1	0		
r111.50	691.812	693.337	707.200	0.099	186	15	1856	168.63
r111.100					1	0		
r112.50	**607.219**	**612.389**	**630.200**	**0.225**	**2896**	**15**	**7496**	**2239.50**
r112.100					1	0		
TOTAL TIME:								45492.67

Table 5.14 Subtour, 2–path, and 3–path cutting planes added at root node of search tree without parallelization. Branching performed on 16 processors

Problem	LB(1)	LB(2)	IP(opt)	gap cl.	nodes	vi	ncols	time(s)
rc101.50	850.021	944.000	944.000	1.000	1	16		7.35
rc101.100	1584.094	1617.339	1619.800	0.931	38	29	1544	384.94
rc102.50	719.902	814.249	822.500	0.920	259	32	2372	300.33
rc102.100					1	0		
rc103.50	643.133	710.667	710.900	0.997	4	21	1274	326.94
rc103.100					1	0		
rc104.50	543.750	543.750	545.800		16	0	1259	459.49
rc104.100					1	0		
rc105.50	754.443	853.675	855.300	0.984	16	17	1002	111.61
rc105.100	1471.160	1509.800	1513.700	0.908	24	18	2023	536.56
rc106.50	664.433	717.930	723.200	0.910	21	22	937	290.56
rc106.100					1	0		
rc107.50	591.476	632.336	642.700	0.798	84	10	1520	793.85
rc107.100					1	0		
rc108.50	538.957	595.982	598.100	0.964	8	19	1483	1834.50
rc108.100					1	0		
TOTAL TIME:								5046.13

Table 5.15 Subtour, 2–path, and 3–path cutting planes added at root node of search tree without parallelization. Branching performed on 16 processors

upper bound. The total number of cuts is given in the column "Number of cuts" by value of k (that is, subtour/2–path/3–path/ $\ldots$ /6–path). Note that only two 4–path cutting planes, one 5–path cutting plane, and no 6–path cutting planes were found in the eleven problems. For two problems, $rc104.100$ and $rc108.100$, the cutting plane procedure was not able to finish within the $30,000$ second time limit. The check to determine if $k(S) \geq k$ failed for 70 sets in $rc104.100$ and in 50 sets for $rc108.100$. Each of these failures are caused because the optimization of the smaller VRPTW was not completed within the 600 second time limit. So as to reduce the time spent in the cutting plane routine, we tried running the code on these problems with the time limit on the smaller VRPTW reduced to 60 seconds. With this new setting, the cutting plane routines reached completion in both cases. For $rc104.100$, the number of incomplete optimizations for the smaller VRPTW increased only slightly to 71, and the values LB(2)=1098.505, LB(3)=1109.4, and UB=1574.10 were generated. For

$rc108.100$, the number of sets for which $k(S) \geq k$ could not be determined increased considerably to 276, and the values LB(2)=1085.781, LB(3)=1093.3, and UB=1617.60 were achieved.

Problem	LB(1)	LB(2)	LB(3)	UB	Number of cuts	nodes
rc102.100	1403.646	1437.264	1456.0	1941.7	0/42/21/1/1/0	12330
rc103.100	1218.495	1243.233	1254.1	1258.0	0/26/69/1/0/0	1488
rc104.100	1094.333	1102.324			2/17/14/0/0/0	0
rc106.100	1308.781	1332.824	1356.1	1878.2	1/27/36/0/0/0	17904
rc107.100	1170.689	1180.995	1201.2	1770.1	1/17/10/0/0/0	10457
rc108.100	1063.011	1082.464			10/12/11/0/0/0	0
r104.100	949.103	951.135	962.3	1434.3	2/8/1/0/0/0	10184
r108.100	907.162	909.472	919.4	1329.2	2/8/1/0/0/0	8905
r110.100	1048.482	1049.939	1064.7	1596.0	1/8/0/0/0/0	21704
r111.100	1032.028	1032.075	1047.3	1049.3	0/4/0/0/0/0	17904
r112.100	919.192	922.337	935.1	1301.0	5/29/3/0/0/0	3279

Table 5.16 Results on the 11 unsolved problems using 16 processors.

In a final attempt to solve the remaining unsolved problems, the code is run with $k = 6$ on 32 machines, the maximum number of computers available, with an increased time limit. Optimal solutions for four more problems are displayed in Table 5.17. Due to a technical issue in the TreadMarks software, the problem instance $r110.100$ cannot be run to completion using 32 processors. Thus the results in table 5.17 for $r110.100$ are obtained using 16 processors, rather than 32 processors. Thus, our algorithm, which is composed of a new separation algorithm for k–path cuts and a parallelized branching tree, succeeds in solving 80 out of 87 of the Solomon benchmark problems. Ten of these problems were previously unsolved in the literature.

Problem	LB(1)	LB(2)	IP(opt)	gap cl.	nodes	vi	time
rc102.100	**1403.646**	**1439.547**	**1457.4**	**0.668**	**26632**	**65**	**42150.49**
rc103.100	**1218.495**	**1242.491**	**1258.0**	**0.607**	**9236**	**77**	**84438.90**
r111.100	**1032.028**	**1032.075**	**1048.7**	**0.003**	**28503**	**4**	**41879.97**
***r110.100**	**1048.482**	**1049.939**	**1068.0**	**0.075**	**90448**	**9**	**457024.92**

Table 5.17 All cuts up to k=6 are added at root node of search tree. Branching performed on 32 processors for the first three problems. Branching performed on 16 processors for problem instance r110.

Chapter 6

A Less-Than-Truckload Trucking Application

Less-than-truckload trucking represents a portion of the motor carrier industry in which the shipments to be sent on trucks do not completely fill a 45,000 lb. capacity tractor-trailer [9]. Typically, the freight in each shipment weighs under 10,000 lbs., with a vast majority falling under 1000 lbs. Since each shipment does not fill a truck, significant savings can be achieved by consolidating shipments into loads at regional terminals and transporting these loads from terminal to terminal. Companies in this industry encounter variations of the traditional vehicle routing problem, as defined in the literature review, both in the gathering and the distribution of shipments at regional terminals. The application discussed in this chapter, however, differs from the classical VRP in that it considers strategic–level decisions for the LTL carriers. In particular, this thesis considers a routing problem called the *strategic load planning problem*, or SLP. A solution to this problem determines how to route the flow of consolidated loads from origin terminals to destination terminals cost effectively and allowing for certain service standards. The daily decisions of how specific shipments are collected and delivered to the origin terminal and distributed from the destination terminals are handled in completely separate problems from the SLP.

The work presented in this chapter is a result of work done by the author in conjunction with two Rice University Ph.D. students, Cassandra McZeal and Erica Klampfl, and under the direction of Dr. Bruce Hoppe of the SABRE Group. Our approach to building a network for the SLP takes advantage of the close relationship that exists between this problem and many classic network design problems. The final solution procedure, in the form of a software system, builds a network to solve the strategic load planning problem provided that the appropriate data concerning the LTL carrier's needs and past routing decisions are available. The empirical results, based on real data from trucking companies, indicates that our system does a very credible job of building an efficient network.

The first section of this chapter defines the MIP formulation of the strategic load planning problem. A mathematical model is provided for clarity, but is not essential to the methods discussed in the rest of the chapter. Section 6.2 discusses the implemented solution strategy, including both a network–building procedure and a final–stage improvement heuristic. In section 6.3, results obtained from the data are provided. Finally, section 6.4 draws some conclusions and briefly examines some possibilities for future work.

6.1 Problem Formulation

6.1.1 Simplifying Assumptions

In order to model the strategic load planning problem, certain simplifying assumptions are made. First, it is assumed that all freight moves via trucks. No airplanes, boats or trains are available to ship the loads. In addition, the fleet is homogeneous. In other words, there exists one type of trailer with one fixed capacity. Also, an unlimited supply of company–owned trailers are available at each terminal for transporting the loads. In order to facilitate higher–level decisions beyond the scope of this particular problem, the number of trailers used to ship loads is allowed to be fractional. Driver feasibility is not considered when creating the load plan since the goal is only to model the movement of the freight. For example, *direct service* may be provided between two terminals whose distance far exceeds the number of miles one driver could traverse nonstop, so long as the freight remains in the same, unopened trailer throughout the journey. Additionally, it is assumed that a driver will be available at any terminal to take a load wherever the load plan dictates.

6.1.2 Problem Description

The strategic load planning problem is based on data obtained from an analysis of a company's past and projected freight distribution needs. This analysis yields a set of loads, referred to as *commodities*, each with an origin and destination terminal and a quantity. These commodities are aggregated, meaning that a given origin/destination pair uniquely identifies a commodity. A commodity with zero freight simply indicates that no freight is shipped from that commodity's origin to its destination. Thus, commodities represent the expected shipment needs of the company. If the origin–to–destination path for a given commodity contains intermediate terminals, then the commmodity's freight is said to be "handled" at these terminals. Each terminal has a capacity in terms of the quantity of freight handled per day, as well as a handling cost per unit of freight and a handling time.

The arcs of the overall network are defined by the set of potential direct services between terminals. A potential direct service is an ordered pair of terminals and refers to the possibility of routing freight from one terminal to another with no intermediate stops at other terminals. Each direct service has a unit cost per amount of freight routed along the arc and a minimum frequency requirement. If freight is routed along a direct service, then the minimum frequency refers to the minimum number of trailers that the company must send along that arc throughout a day. If no freight is routed along the direct service, then the minimum frequency restriction does not apply. Finally, the company may also set certain service standards for each origin/destination pair. These standards refer to the maximum time permitted to transport a commodity from its origin to its destination. These restrictions limit the lengths of the routes.

The solution to the strategic load planning problem must provide two key pieces of information. First, it must reveal the status of each potential direct service. This indicates that either freight is routed along a given direct service (status in) or it is not (status out). Second, a *load plan* providing the path for each commodity from its origin to its destination is required. This load plan must be tree-based so that given a current terminal location and a destination, the next terminal on the commodity's path is uniquely defined. The set of terminals, along with the status–in directs, define the solution network. Other restrictions on the solution include that there be enough trailers traveling along each status–in direct service to carry the amount of freight being sent along that arc, as well as to meet the minimum frequency requirement. A terminal may not exceed its handling capacity for freight and must have zero net change in trailers, which may require the movement of empty trailers. The path for each commodity may not exceed the service standard. Finally, the goal is to minimize the sum of the cost of sending trailers along direct services and the cost of handling freight at the terminals.

6.1.3 Mathematical Model

The SLP can be formulated as a mixed integer programming problem in the following manner. The set of terminals for the strategic load planning problem is denoted by T. For each t in T, there is a handling cost, HC_t, and a handling time, HT_t. Also for each terminal, there is a capacity on the number of trailers of freight that can be handled in a day, CAP_t. The set of commodities to be shipped is denoted by K, and each element k has an origin terminal denoted $O(k)$ and a destination terminal denoted $D(k)$ (recall, commodities are unique origin/destination pairs). The set $D \subseteq T$ is defined as the set of destination terminals for the set of commodities K. That is, $D = \bigcup_{k \in K} D(k)$. The parameter Q_k is the number of trailers, possibly fractional, needed to ship commodity k. The amount of time advertised to ship commodity k from its origin terminal to its destination terminal is the service standard, ST_k. The set of potential direct services is denoted by A. For each ij in A, TC_{ij} is the cost of sending one trailer from terminal i to terminal j with transit time TT_{ij}. The final parameter, MF_{ij}, refers to the minimum number of trailers that must be sent along arc ij throughout a day, if freight is allowed to move along ij.

The decision variable f_{ijk} represents the assignment of $ij \in A$ to the freight path for $k \in K$. The variable f_{ijk} is 1 if direct ij is used to ship commodity k to its destination terminal $D(k)$. These variables define the freight path for each commodity. The decision variable n_{ijd} is 1 if the next transfer at i is j for a commodity whose final destination is d. These variables determine a load plan that depends only on the destination terminal for a commodity. The variable v_{ij} is the number of trailers sent on direct ij. The strategic load planning problem is stated as:

$$\text{Min} \quad \sum_{ij \in A} TC_{ij} v_{ij} + \sum_{k \in K, O(k), D(k) \neq j} \; \sum_{ij \in A} HC_j Q_k f_{ijk} \tag{6.1}$$

$$\text{s.t.} \quad \sum_{ij \in A} f_{ijk} - \sum_{ji \in A} f_{jik} = \begin{cases} -1 & i = D(k) \\ 0 & i \neq O(k), D(k) \end{cases} \qquad i \in T, \quad k \in K \tag{6.2}$$

$$\sum_{k \in K, O(k), D(k) \neq t} (Q_k \sum_{it \in A} f_{itk}) \leq CAP_t \qquad t \in T \tag{6.3}$$

$$\sum_{ij \in A} (TT_{ij} + HT_j) f_{ijk} \leq ST_k + HT_{D(k)} \qquad k \in K \tag{6.4}$$

$$f_{ijk} \leq n_{ijD(k)} \qquad ij \in A, \quad k \in K \tag{6.5}$$

$$\sum_{tj \in A} n_{tjd} \leq 1 \qquad d \in D, \quad t \in T, t \neq d \tag{6.6}$$

$$n_{djd} = 0 \qquad d \in D, \quad dj \in A \tag{6.7}$$

$$\sum_{k \in K} Q_k f_{ijk} \leq v_{ij} \qquad ij \in A \tag{6.8}$$

$$\sum_{it \in A} v_{it} - \sum_{ti \in A} v_{ti} = 0 \qquad t \in T \qquad (6.9)$$

$$v_{ij} \geq MF_{ij} n_{ijd} \qquad ij \in A, \quad d \in D \qquad (6.10)$$

$$n_{ijD(k)}, f_{ijk} \in \{0, 1\} \qquad ij \in A, k \in K \qquad (6.11)$$

The objective is to minimize total cost. The costs incurred are the transit costs per trailer on each direct service provided, in addition to the cost of handling freight at each of the intermediate handling terminals on a commodity's freight path. The first set of constraints in (6.2) are the flow conservation constraints for each commodity k in the network. The flow of commodity k at the destination terminal $D(k)$ is -1 since no flow leaves the terminal and one unit of flow arrives. The reverse is true for the origin terminal (which has a flow of $+1$), though it does not need to be explicitly stated in the formulation. The second set of constraints is the terminal capacity constraints. The constraints specify that each terminal handle no more than its capacity. The service standard set of constraints in (6.4) ensures that the handling time and transit time for a commodity's freight path to its destination terminal is no more than the advertised time for arrival at the destination. The fourth set of constraints (6.5) link the freight path variables to the load path variables so that if the next destination from i is not j for a commodity destined for $D(k)$, then f_{ijk} is forced to be zero.

Constraints (6.6) and (6.7) force the load plan to be tree-based and rooted in the destination terminals. Constraint set (6.6) allows no more than one next transfer from any terminal t for a specific destination d. The next set eliminates any next transfers from a commodity's destination terminal. Constraints (6.8)-(6.10) are related to the trailers. Constraint set (6.8) requires that the numbre of trailers sent along an arc is large enough to carry the amount of freight being shipped along the arc. The set (6.9) ensures that the trailers balance at each terminal, and (6.10) specifies that the number of trailers shipping commodities on a direct service satisfies the minimum frequency condition. Finally, the freight path and load plan decision variables are binary.

Clearly, the MIP formulation of the problem can become very large. The number of integer variables alone is on the order of $|A| * |K|$. Since both the number of directs and the number of commodities are bounded by the square of the number of terminals, the number of integer variables is on the order of $|T|^4$. Given the size and difficulty of the integer programming formulation for any realistic data set, it is reasonable, if not necessary, to use an approximate solution method.

6.2 Solution Strategy

The solution strategy can be divided into three major phases. The first phase involves pruning the overall network of possible direct services. Our goal in this phase is to eliminate the more unlikely direct services while also maintaining feasibility. The second stage is an iterative network building phase. A modified version of Balakrishnan, Magnanti, and Wong's [5] dual–ascent procedure for uncapacitated network design is used to build the network at each iteration. Also, at each iteration we modify the network cost vectors in order to heuristically encourage solutions that adhere more closely to the requirements of the strategic load planning problem. In the last stage of the algorithm, we apply an add/drop procedure to the final solution from the iterative phase. The add/drop procedure systematically adds and drops direct services in order to find local improvements in the solution.

6.2.1 Pruning Techniques

To understand the necessity of pruning the overall network, consider an instance with 50 terminals, a complete, directed graph such that each arc is a potential direct service, and the requirement that some quantity of freight travel between every pair of terminals. A complete graph, in this case, implies a network with 2,450 arcs. The number of commodities is the same. Thus, the number of network flow variables is well over 6 million. Other than eliminating the commodities with zero freight, the number of commodities cannot be reduced. However, it is possible to eliminate arcs as a means of reducing the problem size. To ensure the feasibility of the problem, we require the existence of a known feasible solution which we call the *historical load plan*, or HLP. The HLP is a complete load plan, supplied as part of the original data, and represents a solution previously used by the company. By not eliminating any arcs belonging to the HLP, the known solution remains feasible and, thus, a valid upper bound on the optimal solution.

By pruning arcs which are unlikely to be in the optimal load plan, the problem size can be reduced significantly. The two factors considered in the process of pruning an arc are (1) the type of terminals at an arc's endpoints and (2) the amount of freight that would be sent along that arc if all commodities were sent along direct services from their origins to their destinations. A terminal's type is either *end–of–line (EOL)* or *breakbulk*. An end–of–line terminal is one which is not permitted to handle any freight. That is, freight may either originate or terminate at the terminal, but it may not be an intermediate terminal on a commodity's path. A breakbulk terminal is permitted to handle intermediate transfers of freight. Recall that freight originating and terminating at a terminal is not considered to be handled. Given this knowledge about the terminals, it is known exactly how much freight could possibly travel on an arc from one EOL to another — only the amount of freight belonging to the commodity corresponding to that direct service. If this commodity's quantity of freight is less than some low freight parameter, and the arc is not in the HLP, then the arc is eliminated from the network. In general, as the low freight parameter increases, so does the number of arcs that are pruned.

Given additional data, further reductions are possible. If a company can provide a *historical load summary* relating how many loaded trailers are used over a certain time frame, this information can be used to prune less frequently used direct services. The real–world data is based on the use of *pups*, which translate into half of a trailer, and *vans*, which translate into one trailer in the model. If the number of trailers previously sent along an arc is less than the predefined parameter and is not in the HLP, then the arc is pruned. Also in this pruning technique, as this parameter indicating the minimum trailers permitted increases, in general the number of pruned arcs increases. While there does exist the risk that optimal solutions are eliminated by removing arcs using these two pruning techniques, so long as the values of these parameters are relatively small, the likelihood of eliminating good solutions is small. As the parameters increase, the tradeoff between a more manageable–sized network and the quality of solutions found may become more pronounced.

6.2.2 The Uncapacitated Network Design Problem

An *uncapacitated network design* (UND) problem consists of a set of nodes, a set of uncapacitated arcs, and a required flow of one unit that must be routed between specified pairs of nodes. Each arc has a fixed cost associated with using the arc and a per unit cost for the amount of flow along the arc. The optimization problem is to select the subset of arcs along which to route the required flow that minimizes the total cost. While this is an $\mathcal{NP}$–hard problem, there exists several approximate solution procedures which have succeeded in finding good solutions for large scale problems of this type. A survey of network design problems and solution methods can be found in a paper written by Balakrishnan, Magnanti, and Mirchandani [4].

The general case of the UND problem has directed arcs ij in the set of arcs A and is defined over the set of nodes V. There is a set of commodities K which define the sets of pairs which must have a flow from the origin to the destination of any given commodity k (denoted $O(k)$ and $D(k)$, respectively). Unit costs are commodity dependent; thus, c_{ij}^k is the per unit cost of routing commodity k along arc ij. Fixed costs, however, are incurred if any commodity is routed along an arc. Thus, F_{ij} denotes the fixed cost of using the arc ij for any commodity. Given these cost parameters, only two types of variables are needed. The first variable is the flow variable x_{ij}^k, which is a variable indicating the fraction of commodity $k \in K$ which is routed along arc $ij \in A$. The second variable is the binary variable y_{ij} indicating whether the arc $ij \in A$ is used ($y_{ij} = 1$), or not ($y_{ij} = 0$). Thus, the primal UND problem can be written as follows:

$$\text{Minimize} \quad \sum_{ij \in A} \sum_{k \in K} c_{ij}^k x_{ij}^k + \sum_{ij \in A} F_{ij} y_{ij} \tag{6.12}$$

$$\text{subject to} \quad \sum_{j \in V} x_{ji}^k - \sum_{j \in V} x_{ij}^k = \begin{cases} 1 & \text{if } i = D(k) \\ 0 & \text{if } i \neq O(k), D(k) \end{cases} \qquad k \in K, i \in V \tag{6.13}$$

$$x_{ij}^k \leq y_{ij} \qquad ij \in A, k \in K \tag{6.14}$$

$$x_{ij}^k \geq 0 \qquad ij \in A, k \in K \tag{6.15}$$

$$y_{ij} \in \{0, 1\} \qquad ij \in A. \tag{6.16}$$

Note that in this formulation the flow variables x_{ij}^k are permitted to be free variables even though they will always take on binary values (that is, either none of a commodity or all of a commodity is routed along an arc). This is due to the unit demand assumed in the network flow constraints.

Given a strategic load planning problem, we define a *related UND problem* by ignoring the capacity restrictions, service requirements, and empty trailer balancing constraints and by approximating the minimum frequency restrictions with the fixed cost parameter of the UND model. More specifically, the related UND model is constructed from an SLP problem containing the following data. First, the amount of freight associated with a commodity $k \in K$ is represented by $frt(k)$, or Q_k in the MIP formulation. Similarly, for each arc ij in the SLP network, transit cost on the arc is defined to be $transcost(ij)$, or TC_{ij}, and the minimum frequency on the arc is defined to be $minfreq(ij)$, or MF_{ij}. For each terminal i in the set of terminals, $handcost(i)$, or HC_i, is the handling cost at terminal i.

In order to build the related UND problem, the set of nodes V must be built from the set of terminals, T. Let each EOL terminal j be represented by a single node j in the set V, and let every breakbulk terminal i be split into two nodes, i_1 and i_2 in V. The set of arcs (or direct services) from the SLP problem can all be translated into the UND model by forcing any arc that ended at a breakbulk terminal i in the SLP model to end at the corresponding node i_1 in the UND model. Similarly, any arc that originated at a breakbulk terminal i in the SLP model, must originate at the corresponding node i_2 in the UND model. In addition to these arcs, there must be a single arc from i_1 to i_2 for all breakbulk terminals i which were split into two nodes. These additional arcs are referred to as the *split breakbulk arcs*, and they represent the actual handling at a terminal of any commodity routed along that arc.

Since the UND model requires a unit demand for each commodity, the commodity–dependent demands of the SLP problem must be transformed to unit demands by scaling the costs for each commodity, as is done in Balakrishnan, Magnanti, and Wong [5] for their instances taken from the less–than–truckload, or LTL, industry. In other words, the value of c_{ij}^k for commodity $k \in K$ and for any arc $ij \in A$ which is not a split breakbulk arc is defined to be $transcost(ij) * frt(k)$. (Note that every arc $ij \in A$ which is not a split breakbulk arc corresponds to a unique arc in the SLP

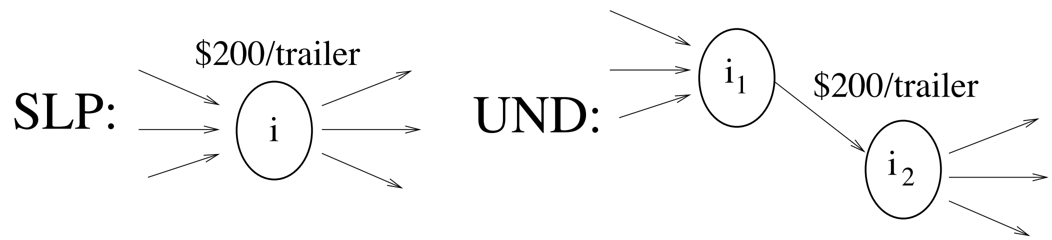

Figure 6.1 In the SLP, breakbulk terminal i is split into nodes i_1 and i_2 in the UND. The handling cost of terminal i becomes the cost of using the split breakbulk arc (i_1, i_2).

model. It is the transit cost of this original arc that $transcost(ij)$ refers to.) The value of $c^k_{i_1 i_2}$ for commodity $k \in K$ and the split breakbulk arc $i_1 i_2$ obtained from terminal i is assigned the value of $handcost(i) * frt(k)$. Finally, the fixed cost F_{ij} of an arc $ij \in A$ which is not a split breakbulk arc is $minfreq(ij) * transcost(ij)$. For the split breakbulk arcs, $F_{ij} = 0$. Thus, we now have a related UND problem which we can solve heuristically using a known algorithm.

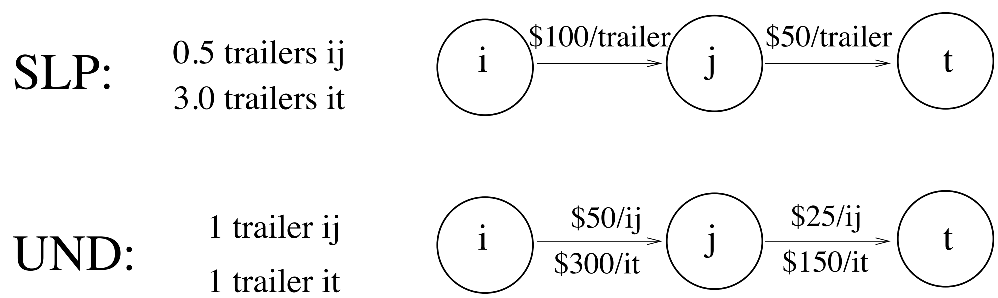

Figure 6.2 The transformation of the cost structure of 2 commodities (one from i to j, the other from i to t) in the SLP to the cost structure in the UND is illustrated above.

The method we use to solve the UND is essentially the dual–ascent procedure described in Balakrishnan, Magnanti and Wong [5]. This approach begins by first relaxing the binary constraints on the y variables in the primal UND problem. Then the dual is formed using the dual variables v_i^k for the flow conservation constraints and the dual variables w_{ij}^k for the second set of constraints (known as the forcing constraints). Thus, the dual UND problem can be written as follows:

$$\begin{aligned}
\text{Minimize} \quad & \sum_{k \in K} v_{D(k)}^k & & (6.17)\\
\text{subject to} \quad & v_j^k - v_i^k - w_{ij}^k \leq c_{ij}^k & ij \in A, k \in K \quad & (6.18)\\
& \sum_{k \in K} w_{ij}^k \leq F_{ij} & ij \in A, k \in K \quad & (6.19)\\
& w_{ij}^k \geq 0 & ij \in A, k \in K \quad & (6.20)
\end{aligned}$$

Notice that when w is fixed, the remaining problem is the dual of a shortest paths problem. As a result, the solution v_i^k is the shortest path from $O(k)$ to i given that the cost on the arcs is the value of $c_{ij}^k + w_{ij}^k$. The dual-ascent algorithm takes advantage of this observation by initially setting $w = 0$ (which is always feasible) and then finding the best values for v using a shortest paths algorithm. From this point, the values of w are iteratively increased with the v values being updated accordingly, until the objective of the dual can no longer be increased by simply increasing a component of w. This process in Balakrishnan, Magnanti, and Wong [5] is referred to as the unrestricted labeling method.

Within the labeling algorithm, we found that it is possible to eliminate several variables and thus improve efficiency. All of the variable eliminations are based on the differences between EOL and breakbulk terminals. Since an EOL terminal can never be an intermediate stop on a commodity's path, certain arcs do not make sense given a particular commodity. For example, if a commodity's origin and destination terminals are both breakbulks, then that commodity can never be routed along an arc that has an EOL terminal as one of its endpoints. Similarly, for a commodity with both EOL origin and destination terminals, no EOL terminal other than the commodity's origin or destination may appear in that commodity's path. Thus, variables w_{ij}^k where the arc ij and the commodity k are incompatible may be eliminated. Also, variables v_i^k such that i is a terminal that can never appear on commodity k's path may also be eliminated. These observations are made in the paper by Balakrishnan, Magnanti, and Wong [5], but the implementation details are not specified. These variable eliminations are carried out in our code, as well.

Once the labeling algorithm is completed, the information from the last iterate is used to construct a primal UND solution. This method, though, does not necessarily yield a tree-based UND solution. To acquire a tree-based solution, it is necessary to build a graph including only the arcs in the newly constructed primal solution. Then costs are assigned to the arcs based on their original commodity–independent transit cost and the handling cost at terminals for the split breakbulk arcs. A subgraph is then built for each terminal that is a destination for some commodity. This subgraph contains only the relevant arcs for the given destination. The direction of each of these arcs is reversed and the single–source shortest paths problem is solved with the destination as the source. This destination–rooted tree then provides the paths from every other node to the destination, given the arcs' original directions. This guarantees that the solution will be tree–based. The corresponding primal UND objective value for the tree–based solution is then computed. To complete the load plan which must have a next transfer for every location/destination pair, we simply fill in unassigned pairs with the value from the historical load plan. Finally, we perform the arc–exclusion test from Balakrishnan, Magnanti, and Wong [5] which is based on the difference between the objective values of the best known primal and dual UND solutions. If any arcs are excluded, then we repeat the algorithm using the smaller network. The dual–ascent procedure terminates when no more arcs can be excluded using the arc–exclusion test, as illustrated by the flowchart in Figure 6.3.

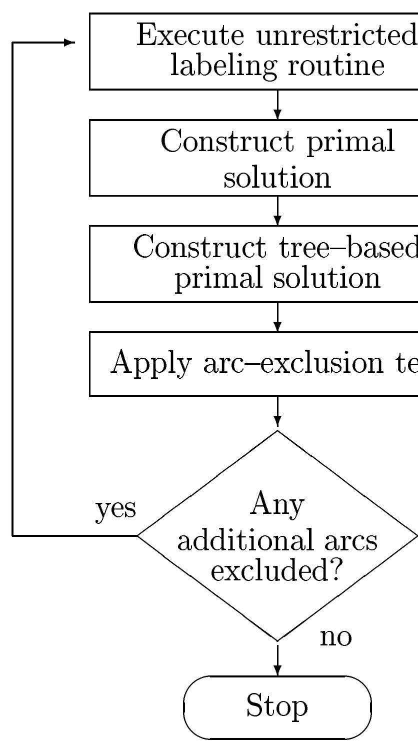

Figure 6.3 Flowchart for the dual–ascent procedure

6.2.3 Calculating the SLP Objective Function Value

The UND solution provides a starting load plan which must be evaluated in terms of the SLP problem. This requires the calculation of the appropriate objective function value. The SLP objective function is the sum of the transit costs of the vehicles used to ship the freight and the handling costs of routing the freight through intermediate (neither origin nor destination) terminals on the freight's path. Determining the latter is more straightforward than determining the former, since a load plan uniquely defines the route from origin to destination for each commodity. Thus, the handling costs are computed for each commodity by a direct examination of the terminals along each route.

Calculating the transit costs incurred shipping each commodity, however, cannot be determined so readily from the load plan. The load plan determines how much freight must be sent from one terminal to any other terminal, but it does not specify how to satisfy minimum frequency constraints or flow conservation of trailers. Fortunately, given the load plan variables n_{ijd} for all $ij \in A, \;\; d \in D$, our mathematical model decomposes into the following minimum cost network flow problem that deals with the minimum frequency and trailer balancing concerns in conjunction with satisfying freight distribution:

$$\text{Minimize} \quad \sum_{ij \in A} TC_{ij} v_{ij} \tag{6.21}$$

$$\text{subject to} \quad \sum_{it \in A} v_{it} - \sum_{ti \in A} v_{ti} = 0 \qquad t \in T \tag{6.22}$$

$$v_{ij} \geq max \;\; (MF_{ij} n_{ijd} \,, \sum_{k \in K, D(k)=d} Q_k f_{ijk}) \qquad ij \in A, \;\; d \in D \tag{6.23}$$

This minimum cost network flow problem specifies that each arc (i, j) has a flow of trailers, represented by the variable v_{ij}, that is at least enough to transport the freight assigned to the direct and enough to satisfy minimum frequency if the direct is status in. The solution to this problem provides all of the information necessary for a solution to the SLP problem that is not already provided by the UND load plan. This problem is solved using the CPLEX 5.0 callable library routine CPXhybnetopt [22]. The objective function value of this optimization corresponds to the final piece of the SLP objective function, the transit cost of the vehicles.

6.2.4 SLP Heuristics

After creating an SLP solution based upon the load plan obtained by solving a UND problem, the current solution may contain certain infeasibilities and/or bad characteristics. In such a case, one would like to improve the approximation of the SLP problem by the UND. To do so, it is possible to modify the cost vectors of the UND problem. Thus, the tree-based solutions constructed within the dual–ascent algorithm are more likely to be good, feasible solutions to the SLP problem. For further refinements to the approximation, this process is then performed in an iterative procedure.

One load plan trait that is avoided by most LTL carriers is the excessive handling of freight caused by too many intermediate terminals on a commodity's path. For example, in the historical solution for the primary data set, all but one commodity are handled at no more than two terminals. Even though this is not a feasibility constraint for the SLP, this is an important requirement for most companies. In practice, handling a commodity at more than two handling points makes it difficult to meet service requirements under real–world conditions. Even though the results from the mathematical model may appear to reduce costs by handling the commodities at several different terminals, the likelihood of unforeseen delays at terminals is too large to risk more than two such points along a freight path. The first SLP heuristic identifies those commodities that are handled more than two times and then multiplies the per unit cost of every split breakbulk arc for such commodities by a positive constant greater than one. Recall from section 6.2.2 that split breakbulk arcs represent the actual handling cost at a terminal of any commodity routed along that arc. By not increasing the cost on every arc of a commodity and only increasing the cost on the split breakbulk arc, the algorithm is less likely to include as many of the more expensive split breakbulk arcs in its next solution.

Another SLP heuristic is concerned with the feasibility restriction relating to service standards. A service restriction is violated if the total time from origin to destination is greater than the commodity's service time plus a service tolerance parameter, which is set to 60 minutes in the model. Unfortunately, the UND model uses the cost function, not time, in its calculations. Thus, the implemented heuristic must rely on the premise that a path using fewer arcs is likely to take less time than a longer path. This is primarily due to the fact that shorter paths imply fewer intermediate terminals, which implies less handling time is incurred. So, if service for a commodity is violated, the cost of every arc for that commodity is multiplied by two. The cost on every arc is increased for this commodity so that the transit costs for this commodity become more important relative to the fixed cost of adding more arcs to the network. Thus, the algorithm becomes more likely to select a more direct path with fewer arcs.

An SLP heuristic is also required to address the minimum frequency requirement of the SLP. Recall that the minimum frequency is the minimum number of trailers that must travel along a direct service that is status in. Each time the objective is computed, though, the minimum frequency requirements are satisfied by the definition of the minimum cost network flow problem. Once the number of vehicles sent along each arc of the network is determined, the approximation of the minimum frequency condition as a fixed cost in the UND model may be modified. For example, in the implemented heuristic, if the number of loaded trailers on a direct is at least minimum frequency, the fixed cost for that direct is set to zero in the UND model. Allowing the arcs that meet minimum frequency to be less expensive improves the approximation of the SLP.

The last requirement for the SLP to be feasible is that each terminal may not exceed its capacity. For our purposes, this is the least important characteristic of a solution. If a terminal is over its capacity, the cost of every commodity for that terminal is multiplied by two. Since the cost for the terminal that exceeds its capacity is increased for every commodity, fewer commodities are likely to be routed through this terminal, and thus the terminal is more likely to meet its capacity requirements.

An iterative loop is implemented to search for a "better" solution. The most important considerations are to reduce the number of service violations, the number of commodities that are over–handled (that is, commodities that are handled at more than two intermediate breakbulk terminals), and the objective function value. Given a current solution and a best solution, the best solution is replaced by the current load plan if

- the current number of service violations is less than the best solution's number of service violations; or
- the number of service violations in both solutions is the same and at least one of the following is true:
    1. the current number of commodities over–handled is not more than the number of commodities over–handled in the HLP, and the current solution's objective function value is less than the best solution's objective function value, or

2. the number of commodities over–handled in the current solution is greater than the number over–handled in the HLP, but less than the number over–handled in the best solution; or

- the number of service violations and the number of commodities over–handled is the same in both the current and best solutions, and the current objective function value is less than the best objective function value.

Since we are continuously multiplying the arc costs in the UND problem, causing them to increase, there is a danger that these costs will attain values that are computationally too large. Therefore, we calculate the upper bound on the number of iterations based on the maximum multiplier for any arc in a single iteration and how many times an arc can be multiplied by this value before overflow occurs. If the same solution is generated in successive iterations or the iteration limit is reached, the loop stops, and the best solution is reported.

6.2.5 Add/Drop Heuristic

After finding solutions to the strategic load planning problem, a simple add/drop heuristic is applied to these solutions in an attempt to improve their overall quality. The heuristic uses the original SLP commodity–independent cost structure for all of the arcs. It begins by considering only the arcs belonging to the inputted load plan. Given this subset of arcs, the heuristic constructs a tree–based load plan as is done in the dual–ascent procedure. Using this load–plan, each origin/destination pair is a potential add or drop candidate. If the direct corresponding to the origin/destination pair is in the network and the freight shipped along it is less than the minimum frequency, then it is a drop candidate. On the other hand, if the direct corresponding to a pair is not in the network and the commodity associated with the origin/destination pair is handled more than twice or has at least minimum frequency amount of freight, then it is considered an add candidate. Fixing a load plan after an arc has been added or dropped simply requires building a new tree–based solution based on the new set of arcs. If the adding or dropping of the arc improves the solution, then the change is kept, otherwise we revert back to the previous solution. An improved solution is one in which either

- the new objective value is lower than the best objective value found so far and neither handling or service violations are worse than those in the original tree–based solution, or
- the new objective value is as good as the best objective value found so far and at least one of the number of handling or service violations has improved and neither has worsened.

This process is repeated until an iteration without improvement is found.

6.3 Results

The computational runs are performed on a Sun UltraSparc I 167 MHz machine with 192 megabytes of memory running Solaris 2.6. The overall structure of the algorithm is outlined in Figure 6.4. In order to fine–tune the algorithm to adapt to different data sets, parameters are introduced that may be varied prior to the start of any search for a feasible solution to the strategic load planning problem. Modifying these parameters is another way to make the mathematical model adhere to real–world considerations. Since the data used in this thesis contain proprietary information, though, exact values for many of the parameters may not be given.

There are two pruning parameters. The low freight parameter refers to the pruning of EOL to EOL arcs. Since this procedure only considers removing a subset of arcs,

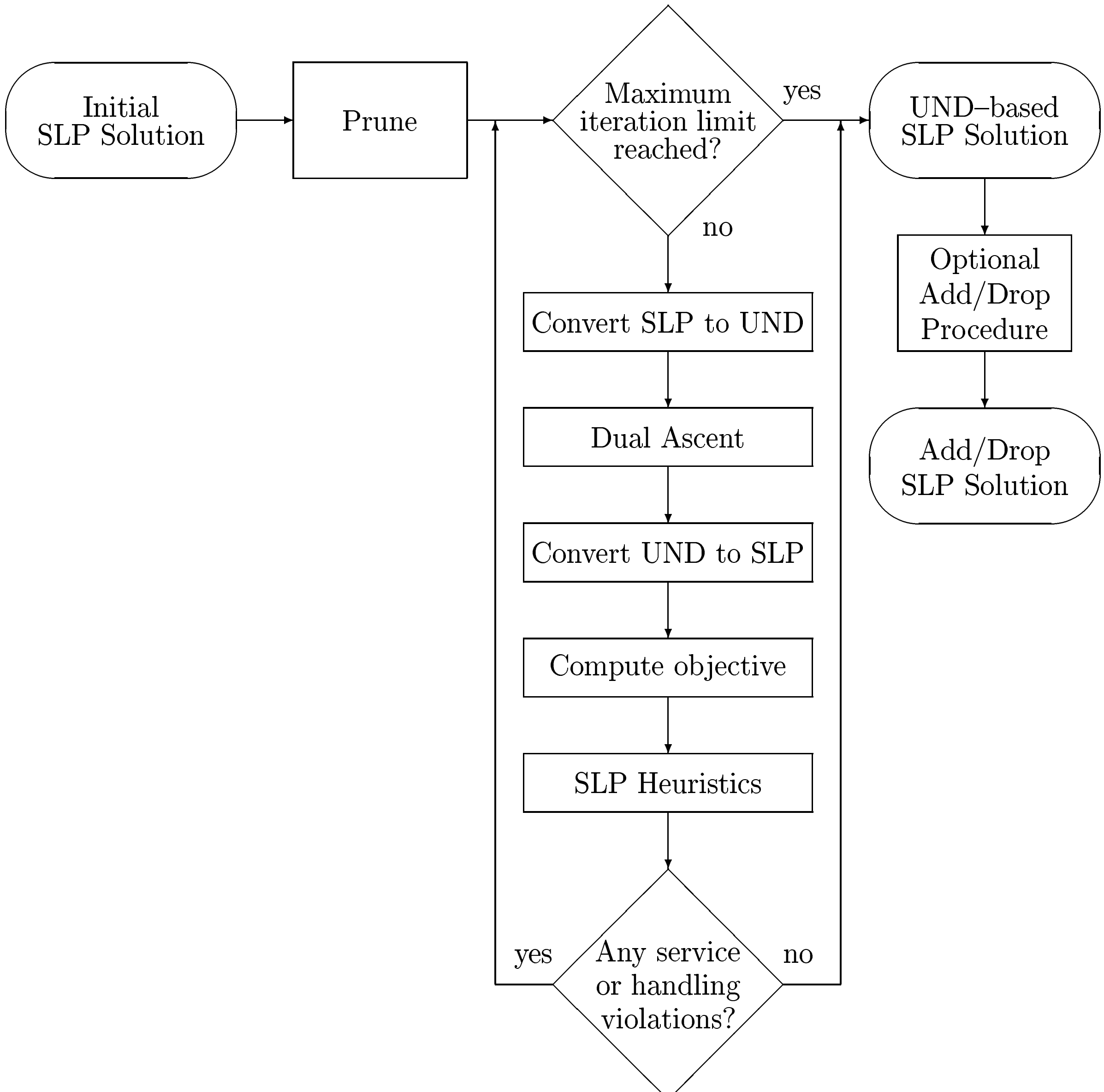

Figure 6.4 Flowchart for the complete algorithm

varying the value of the parameter has a limited effect. The pruning parameter used with the historical load summary,on the other hand, can potentially affect all of the arcs except for those in the HLP. Clearly, this technique is only relevant in instances where a load summary is provided.

The original data assume a fixed minimal number of hours for handling at any terminal. This throughput time at each terminal assumes that there is one time each day (around midnight) when the majority of the freight to be transferred arrives and is handled in one very busy time window. Chances are high that a path with more than two transfer points will fall outside this window for at least one of the transfer points. Thus, a constant parameter is used to artificially inflate all handling times, in this way promoting more directs and fewer handling points in the network.

The original data also assumes 28,403 lbs/van and 14,202 lbs/pup when computing the number of trailer–volumes of freight for each commodity. The actual trailer capacities are closer to 35,000 lbs/van and 20,000 lbs/pup. Increasing the amount of freight that can be loaded onto each tractor–trailer effectively increases handling costs relative to transit costs. This is easily seen since handling costs are a function of the amount of freight, but transit costs are a function of the number of trailers used. In the experments, a parameter is used to adjust the freight density of each type of trailer.

6.3.1 SEFL Data Set

The primary data for this research comes from Southeastern Freight Lines (SEFL) headquartered in Lexington, SC. Four types of data were provided to us by SEFL. The first is terminal data, which includes the code for each terminal, the handling cost, the handling time, and the capacity. The capacity data, in this case, is less of a restriction and more of a suggestion as to how much freight is acceptable to transfer through each location. The SEFL network made available to us contains 48 terminals. The second set of data contains information about the possible direct services in the network. The data includes the origin terminal, the destination terminal, the cost per trailer, the transit time, and the minimum frequency for each direct. The third set of data describes the commodities. We are given the origin terminal, destination terminal, trailer volumes per day, and the service standard advertised for each commodity. The trailer volumes per day is given as the number of trailers, possibly fractional, that the freight will fill. All freight is assumed to originate at 9:00 p.m. and is due by 8:00 a.m., local time. Therefore, the service times vary in small amounts depending on changes in time zones and in large amounts depending on the type of service (one–, two–, or three–day service). The fourth and final type of data provided is a historical load summary relating the number of loaded trailers sent along each arc over a three month period. As was previously mentioned, this permits the use of an intelligent pruning technique.

In addition to this data, which defines the SLP problem, SEFL supplied a historical load plan, or HLP, representing what they considered to be a very good solution in their network. The HLP is used inside our algorithm to maintain feasibility and is used outside the algorithm as a means of evaluating the quality of our solutions. In order to have a basis for comparison between the historical load plan and the various tests which we performed, we select one set of parameters corresponding to our original data set which we use to evaluate all of the solutions. The parameters correspond to (1) not changing the freight density (Frt. D. = 1.0), and (2) not increasing the handling time at a terminal beyond its initial setting (Handtime=0). In this framework, the HLP has no service violations, a very small number of handling violations (defined as a commodity being handled more than two times), and a small number of capacity violations. Actual values are not given here due to confidentiality restrictions. All of the results are reported relative to these actual values, regardless of the individual test parameters.

In general, all of the SLP heuristics worked as anticipated, but with varying degrees of success. The cost modifications implemented to improve the number of commodities meeting the service standard worked most efficiently. It worked well alone, as well as in conjunction with the other heuristics. As for the heuristic to handle commodities which were routed through more than two intermediate terminals, we found that this heuristic worked better when combined with the service heuristic. In other words, if only the handling heuristic is called at each iteration, we found that the number of commodities which were over–handled improved; but, if both the handling and the service heuristics were called at each iteration, the handling improved even faster (and with no negative consequences for the service heuristic). Therefore, both heuristics are applied at each iteration throughout the testing process.

In all of the preliminary testing, the heuristic to reduce the number of capacity violations at terminals succeeds in reducing the capacity violations, but at great cost in terms of the objective function. Recall that this heuristic increases the cost of using a terminal that is over its capacity for every commodity, with no regard to the degree to which the constraint is violated. Rather than modifying the heuristic, we chose to simply disregard capacity violations in the model. This decision is based on two major factors. First, and most importantly, SEFL expressed interest in a non–capacitated SLP model which could be used to better understand capacity needs at terminal locations. Second, the nature of the capacity data that was supplied reflects more of a soft constraint, than a strict requirement. Without more specific information, the capacities that were supplied are better thought of as guidelines. Since the solutions obtained without considering capacity violations tended to have a comparable number of capacity violations to the HLP, it is likely that the solutions are within an acceptable range. Since comparison of solutions in terms of capacity violations is not essential to SEFL and difficult within the constraints of the data set, numerical results for capacity violations are not given. The only form of capacity restriction left in the model is the definition of an EOL terminal. Thus, each terminal either has no capacity for handling freight (an EOL), or it has unlimited capacity (a breakbulk).

Three basic settings are modified to obtain the final results. The first setting concerns the number of arcs not in the HLP that are pruned. The EOL–to–EOL pruning parameter remains fixed at a reasonable value throughout all of the computations. The frequency parameter pertaining to the historical load summary, though, is used to prune varying proportions of those arcs not yet pruned and not in the HLP. As for the freight density parameter, the values 1.0 (corresponding to no change in the original data), 1.1 and 1.2 which yield valid interpretations of trailer capacity are tested. The third setting refers to the handling time at terminals. By adding a positive value to the through time at all terminals, the likelihood both of over–handling freight and of service violations in the real world decreases. The extra handling time used to provide a buffer is tested with the values 0, 60, and 120 minutes.

In the following tables, the results are reported in relative terms, so as to not reveal confidential information about the SEFL network. The value in the "Prune" column refers to the proportion of arcs not in the HLP that are pruned. Thus, a 1.0 prune value indicates that only the arcs of the HLP were used to obtain the UND–based solution. The column marked "Frt. D." refers to the freight density setting and the column marked "Handtime" refers to the extra handling time added to all terminals. Handling errors are reported as the number of violations in the new solution minus the number of violations in the HLP divided by the number of violations in the HLP. Since the HLP contains zero service violations, a plus sign is used to indicate a positive

	Parameter Settings			UND–based Results			
Case	Prune	Frt.D.	Handtime	Handling	Service	Objective	Time (s)
S.1	0.76	1.0	0	2.00	0	-0.006	355.34
S.2	0.76	1.0	60	2.00	0	-0.001	370.95
S.3	0.76	1.0	120	2.00	0	0.008	404.70
S.4	0.76	1.1	0	1.00	0	-0.009	363.66
S.5	0.76	1.1	60	1.00	0	-0.001	376.42
S.6	0.76	1.1	120	1.00	0	0.010	412.75
S.7	0.76	1.2	0	1.00	0	-0.013	466.33
S.8	0.76	1.2	60	1.00	0	-0.007	356.60
S.9	0.76	1.2	120	1.00	0	0.005	542.97
S.10	0.90	1.0	0	1.00	0	-0.008	244.92
S.11	0.90	1.0	60	2.00	0	-0.010	236.02
S.12	0.90	1.0	120	2.00	0	-0.010	262.55
S.13	0.91	1.1	0	2.00	0	-0.008	250.98
S.14	0.91	1.1	60	2.00	0	-0.007	260.64
S.15	0.91	1.1	120	2.00	0	-0.008	269.43
S.16	0.91	1.2	0	0.00	0	-0.016	270.97
S.17	0.91	1.2	60	0.00	0	-0.011	268.42
S.18	0.91	1.2	120	0.00	0	-0.014	310.08
S.19	0.93	1.0	0	2.00	0	-0.015	246.86
S.20	0.93	1.0	60	2.00	0	-0.016	254.35
S.21	0.93	1.0	120	2.00	0	-0.014	244.35
S.22	0.93	1.1	0	5.00	0	-0.015	247.11
S.23	0.93	1.1	60	2.00	0	-0.014	258.48
S.24	0.93	1.1	120	2.00	0	-0.014	262.17
S.25	0.93	1.2	0	3.00	0	-0.015	232.65
S.26	0.93	1.2	60	2.00	0	-0.014	261.02
S.27	0.93	1.2	120	2.00	0	-0.015	268.29

Table 6.1 This table reports results with varying settings for the prune, freight density and extra handling time parameters.

	Parameter Settings			UND–based Results			
Case	Prune	Frt.D.	Handtime	Handling	Service	Objective	Time (s)
S.28	0.96	1.0	0	2.00	0	-0.014	225.92
S.29	0.96	1.0	60	2.00	0	-0.016	234.23
S.30	0.96	1.0	120	2.00	0	-0.014	239.75
S.31	0.96	1.1	0	5.00	0	-0.013	209.55
S.32	0.96	1.1	60	2.00	0	-0.014	236.31
S.33	0.96	1.1	120	2.00	0	-0.014	243.78
S.34	0.96	1.2	0	0.00	0	-0.012	215.10
S.35	0.96	1.2	60	3.00	0	-0.017	238.46
S.36	0.96	1.2	120	2.00	0	-0.012	266.39
S.37	0.99	1.0	0	1.00	0	-0.017	192.13
S.38	0.99	1.0	60	2.00	0	-0.018	185.95
S.39	0.99	1.0	120	1.00	0	-0.018	179.15
S.40	0.99	1.1	0	2.00	0	-0.015	196.14
S.41	0.99	1.1	60	2.00	0	-0.017	190.68
S.42	0.99	1.1	120	2.00	0	-0.017	286.39
S.43	0.99	1.2	0	5.00	0	-0.017	194.70
S.44	0.99	1.2	60	3.00	0	-0.019	190.12
S.45	0.99	1.2	120	2.00	0	-0.017	220.99
S.46	1.00	1.0	0	6.00	0	-0.018	178.25
S.47	1.00	1.0	60	5.00	0	-0.018	189.99
S.48	1.00	1.0	120	4.00	0	-0.017	170.07
S.49	1.00	1.1	0	4.00	+	-0.018	150.81
S.50	1.00	1.1	60	4.00	0	-0.019	191.81
S.51	1.00	1.1	120	3.00	0	-0.017	230.11
S.52	1.00	1.2	0	1.00	0	-0.020	181.97
S.53	1.00	1.2	60	1.00	0	-0.021	192.57
S.54	1.00	1.2	120	1.00	0	-0.017	251.99

Table 6.2 This table reports results with varying settings for the prune, freight density and extra handling time parameters.

number of violations and a zero is used to indicate no violations. The objective value refers to the new solution's obective value minus the HLP's objective value divided by the HLP objective value. Thus, a negative value in the column for handling errors or for objective values indicates an improvement over the HLP.

In addition to the above tests, an additional series of runs involving a new data file for the terminal information is performed. Rather than assume that all of the terminals have the same handling time, the new data contains the actual handling time for each terminal. This data is then modified by forcing the through times to have a certain minimum value. This resulted in a set of terminals with greater through times which were more reflective of real–world conditions. With this new file, similar tests to those in Tables 6.1 and 6.2 are run, but without adding any extra handling time at the terminals. These solutions generally had fewer handling violations than their counterparts and can be found in Table 6.3.

	Parameter Settings		UND–based Results			
Sedfile	Prune	Frt.D.	Hand. Errors	Serv. Errors	Objective	Time (s)
S.55	0.93	1.0	-1.00	0	-0.014	262.52
S.56	0.93	1.1	-1.00	0	-0.014	278.78
S.57	0.93	1.2	-1.00	0	-0.012	268.01
S.58	0.96	1.0	1.00	0	-0.015	226.20
S.59	0.96	1.1	1.00	0	-0.015	228.28
S.60	0.96	1.2	1.00	0	-0.010	281.33
S.61	1.00	1.0	5.00	0	-0.016	170.32
S.62	1.00	1.1	2.00	0	-0.016	201.90
S.63	1.00	1.2	0.00	0	-0.016	238.55

Table 6.3 This table reports results obtained using a new terminal data file.

Clearly the UND–based algorithm, prior to performing the add/drop heuristic, is capable of finding good solutions in the model. In every instance, the objective value is lower than the original HLP objective value. In fact, a 1% decrease (that is,

-0.01 change) in objective value can save the company several thousands of dollars a day. As for the feasibility issues, the service requirements are, for the most part, easily met. Considering that the initial number of handling violations is marginal, the number of handling violations in the solutions stays well within reason, and is even improved in certain instances. Some of the best results were obtained using the new file on terminal handling times. This seems to indicate that the better the data reflects the real problem, the better the overall performance of the UND–based algorithm. Pruning also has a large influence on the solution quality. Leaving only "good" arcs in the network for the UND not only decreases solution time, but also increases the improvement in solution quality. We suspect that the better the pruning technique, the better the resulting solution.

Finally, an add/drop procedure is applied to all of the UND–based solutions, as well as to the HLP. The results can be seen in Tables 6.4–6.6. All of these solutions are clearly superior to the original HLP, though, unfortunately, the add/drop on the HLP yields the overall best solution. The results do indicate that the UND–based algorithm, in combination with the add/drop heuristic or standing alone, does do a credible job of building a strategic load plan. It is possible that there is something inherently better about SEFL's HLP that our model simply does not capture. This is difficult to evaluate, though, without access to further SEFL data. It is also possible that our simplistic add/drop heuristic actually lessens the quality of our solutions by increasing the likelihood of service violations, given that we do not take into consideration any buffers on handling time at the terminals in the add/drop heuristic. In other words, though we maintain the same number of service violations while performing the add drop, the way in which we calculate a service violation is to always use the original terminal data with the minimal, fixed through times. As discussed previously, it is not clear that this is the best way in which to model the

Case	UND–based SLP solution				Add/Drop SLP solution			
	Handling	Service	Objective	Time (s)	Handling	Service	Objective	Time (s)
HLP	-	-	-	-	-1.00	0	-0.054	780.63
S.1	2.00	0	-0.006	355.34	-1.00	0	-0.027	483.13
S.2	2.00	0	-0.001	370.95	-1.00	0	-0.029	505.29
S.3	2.00	0	0.008	404.70	-1.00	0	-0.039	748.34
S.4	1.00	0	-0.009	363.66	-1.00	0	-0.019	343.24
S.5	1.00	0	-0.001	376.42	-1.00	0	-0.029	502.62
S.6	1.00	0	0.010	412.75	-1.00	0	-0.037	733.84
S.7	1.00	0	-0.013	466.33	-1.00	0	-0.020	327.37
S.8	1.00	0	-0.007	356.60	-1.00	0	-0.028	497.71
S.9	1.00	0	0.005	542.97	-1.00	0	-0.037	636.01
S.10	1.00	0	-0.008	244.92	-1.00	0	-0.029	338.55
S.11	2.00	0	-0.010	236.02	-1.00	0	-0.034	360.63
S.12	2.00	0	-0.010	262.55	-1.00	0	-0.040	402.06
S.13	2.00	0	-0.008	250.98	-1.00	0	-0.027	335.80
S.14	2.00	0	-0.007	260.64	-1.00	0	-0.033	362.50
S.15	2.00	0	-0.008	269.43	-1.00	0	-0.038	527.28
S.16	0.00	0	-0.016	270.97	-1.00	0	-0.033	320.52
S.17	0.00	0	-0.011	268.42	-1.00	0	-0.034	400.37
S.18	0.00	0	-0.014	310.08	-1.00	0	-0.041	856.02
S.19	2.00	0	-0.015	246.86	-1.00	0	-0.030	303.18
S.20	2.00	0	-0.016	254.35	-1.00	0	-0.034	448.24
S.21	2.00	0	-0.014	244.35	-1.00	0	-0.040	524.33
S.22	5.00	0	-0.015	247.11	-1.00	0	-0.029	313.55
S.23	2.00	0	-0.014	258.48	-1.00	0	-0.033	432.47
S.24	2.00	0	-0.014	262.17	-1.00	0	-0.039	517.95
S.25	3.00	0	-0.015	232.65	-1.00	0	-0.029	308.27
S.26	2.00	0	-0.014	261.02	-1.00	0	-0.033	429.30
S.27	2.00	0	-0.015	268.29	-1.00	0	-0.039	506.21

Table 6.4 Compares the UND–based solution to the solution obtained after performing the add/drop heuristic.

Case	UND–based SLP solution				Add/Drop SLP solution			
	Handling	Service	Objective	Time (s)	Handling	Service	Objective	Time (s)
HLP	-	-	-	-	-1.00	0	-0.054	780.63
S.28	2.00	0	-0.014	225.92	-1.00	0	-0.031	318.00
S.29	2.00	0	-0.016	234.23	-1.00	0	-0.034	333.80
S.30	2.00	0	-0.014	239.75	-1.00	0	-0.038	496.61
S.31	5.00	0	-0.013	209.55	-1.00	0	-0.028	322.79
S.32	2.00	0	-0.014	236.31	-1.00	0	-0.032	322.67
S.33	2.00	0	-0.014	243.78	-1.00	0	-0.037	483.41
S.34	0.00	0	-0.012	215.10	-1.00	0	-0.027	331.57
S.35	3.00	0	-0.017	238.46	-1.00	0	-0.033	439.50
S.36	2.00	0	-0.012	266.39	-1.00	0	-0.037	492.88
S.37	1.00	0	-0.017	192.13	-1.00	0	-0.032	301.50
S.38	2.00	0	-0.018	185.95	-1.00	0	-0.036	311.93
S.39	1.00	0	-0.018	179.15	-1.00	0	-0.038	376.25
S.40	2.00	0	-0.015	196.14	-1.00	0	-0.029	287.14
S.41	2.00	0	-0.017	190.68	-1.00	0	-0.033	306.35
S.42	2.00	0	-0.017	286.39	-1.00	0	-0.037	480.32
S.43	5.00	0	-0.017	194.70	-1.00	0	-0.029	290.54
S.44	3.00	0	-0.019	190.12	-1.00	0	-0.035	422.16
S.45	2.00	0	-0.017	220.99	-1.00	0	-0.037	484.26
S.46	6.00	0	-0.018	178.25	-1.00	0	-0.018	295.55
S.47	5.00	0	-0.018	189.99	-1.00	0	-0.026	328.02
S.48	4.00	0	-0.017	170.07	-1.00	0	-0.028	383.45
S.49	4.00	+	-0.018	150.81	-1.00	+	-0.022	294.49
S.50	4.00	0	-0.019	191.81	-1.00	0	-0.025	331.74
S.51	3.00	0	-0.017	230.11	-1.00	0	-0.030	344.90
S.52	1.00	0	-0.020	181.97	-1.00	0	-0.022	279.94
S.53	1.00	0	-0.021	192.57	-1.00	0	-0.027	326.44
S.54	1.00	0	-0.017	251.99	-1.00	0	-0.034	453.39

Table 6.5 Compares the UND–based solution to the solution obtained after performing the add/drop heuristic.

Case	UND–based SLP solution				Add/Drop SLP solution			
	Handling	Service	Objective	Time (s)	Handling	Service	Objective	Time (s)
HLP	-	-	-	-	-1.00	0	-0.054	780.63
S.55	-1.00	0	-0.014	262.52	-1.00	0	-0.044	427.77
S.56	-1.00	0	-0.014	278.78	-1.00	0	-0.044	563.49
S.57	-1.00	0	-0.012	268.01	-1.00	0	-0.044	546.68
S.58	1.00	0	-0.015	226.20	-1.00	0	-0.039	538.33
S.59	1.00	0	-0.015	228.28	-1.00	0	-0.038	489.90
S.60	1.00	0	-0.010	281.33	-1.00	0	-0.038	505.36
S.61	5.00	0	-0.016	170.32	-1.00	0	-0.026	387.80
S.62	2.00	0	-0.016	201.90	-1.00	0	-0.031	353.20
S.63	0.00	0	-0.016	238.55	-1.00	0	-0.035	505.01

Table 6.6 Compares the UND–based solution to the solution obtained after performing the add/drop heuristic.

SLP problem. Since it is, however, the setting in which we were given the HLP, it is the one to which we adhere. Also note that the entire algorithm is easily completed in under 20 minutes.

6.3.2 Averitt Data Set

Although the code was initially tailored to the SEFL data set, a second set of data was obtained to test the effectiveness of the algorithm. This data comes from the an alternate carrier called Averitt Express. Their network contains 78 terminals, rather than 48. Essentially the same type of data was supplied as in the case of SEFL. The one exception is the lack of an historical load summary. Though very useful as a pruning tool, this data is not essential to the method. Averitt also supplied an historical load plan, which we will continue to refer to as the HLP. In this case, the HLP contained several handling and service violations. This solution, while considered very good by Averitt, was not as well–tested as the HLP provided by SEFL. Solutions are evaluated relative to the HLP solution, as in the SEFL results section. Since the number of service violations in the HLP for the Averitt data set is nonzero, the relative calculation of service errors is done in the same manner as the handling errors. For the same reasons as previously discussed, we continue to disregard possible capacity violations in the model.

Since the Averitt data does not include a load summary, the pruning techniques available to us are more limited. The technique in which low freight EOL–to–EOL arcs not in the HLP are eliminated is, in this case, the only flexible pruning method we may use. In the first set of tests performed in Table 6.7, this low freight parameter is set to a reasonably low value. In the second set of tests, the low freight parameter is increased so that all EOL–to–EOL arcs not in the HLP are eliminated. Finally, in the third set of runs, all arcs not in the HLP are pruned. The different settings for freight density and extra handling time remain the same.

Table 6.7 displays the results obtained by running the UND–based algorithm on the Averitt data set. The larger network resulted in a considerable increase in time. Creating a strategic load plan is not a task that is likely to be performed frequently. Thus, the increase in time is not unreasonable, especially since the solutions obtained are, overall, much better than the HLP solution. It is particularly interesting to note that the solutions obtained using only the HLP arcs in the network have the greatest improvement in objective function value, and the worst performance in terms of handling and service violations. If a better pruning technique were available — one in which approximately 90% of the arcs not in the HLP could be pruned — it is very likely that even better solutions than those in Table 6.7 could be found. As the results stand, the solutions are quite good. In this problem instance, a decrease in the objective of 1% also translates into a couple of thousands of dollars a day.

Due to the increased time involved in solving problems in this data set, the add/drop portion of our algorithm was run on only three UND–based solutions and the HLP. As can be seen in Table 6.8, two out of three of these solutions are clearly

	Parameter Settings			UND–based Results			
Case	Prune	Frt.D.	Handling	Hand. Errors	Serv. Errors	Objective	Time (s)
A.1	0.58	1.0	0	-1.00	-0.23	-0.093	4678.31
A.2	0.58	1.0	60	-1.00	-0.23	-0.067	4248.93
A.3	0.58	1.0	120	-1.00	-0.15	-0.001	4471.12
A.4	0.59	1.1	0	-1.00	-0.19	-0.085	5221.64
A.5	0.59	1.1	60	-1.00	-0.27	-0.060	4967.60
A.6	0.59	1.1	120	-1.00	-0.10	-0.006	4584.90
A.7	0.61	1.2	0	-1.00	-0.27	-0.088	4718.58
A.8	0.61	1.2	60	-1.00	-0.19	-0.070	5013.90
A.9	0.61	1.2	120	-0.73	-0.10	-0.009	4662.14
A.10	0.72	1.0	0	-1.00	-0.23	-0.093	3631.35
A.11	0.72	1.0	60	-1.00	-0.23	-0.067	3290.05
A.12	0.72	1.0	120	-1.00	-0.15	-0.001	3458.89
A.13	0.72	1.1	0	-1.00	-0.19	-0.085	4113.02
A.14	0.72	1.1	60	-1.00	-0.27	-0.060	3920.00
A.15	0.72	1.1	120	-1.00	-0.10	-0.006	3583.44
A.16	0.72	1.2	0	-1.00	-0.27	-0.088	3786.37
A.17	0.72	1.2	60	-1.00	-0.19	-0.070	4020.06
A.18	0.72	1.2	120	-0.73	-0.10	-0.009	3737.60
A.19	1.00	1.0	0	1.64	0.17	-0.142	747.09
A.20	1.00	1.0	60	1.64	0.19	-0.134	1034.41
A.21	1.00	1.0	120	1.64	0.19	-0.127	1001.86
A.22	1.00	1.1	0	1.55	0.17	-0.141	760.95
A.23	1.00	1.1	60	1.55	0.19	-0.134	1059.33
A.24	1.00	1.1	120	1.55	0.19	-0.127	1012.96
A.25	1.00	1.2	0	1.36	0.17	-0.139	778.31
A.26	1.00	1.2	60	1.36	0.19	-0.131	1072.61
A.27	1.00	1.2	120	1.36	0.21	-0.125	1033.71

Table 6.7 UND–based results for the Averitt data set.

	UND–based SLP solution				Add/Drop SLP solution			
Case	Handling	Service	Objective	Time (s)	Handling	Service	Objective	Time (s)
HLP	-	-	-	-	0.18	0.08	-0.084	11680.50
A.7	-1.00	-0.27	-0.088	4718.58	-1.00	-0.31	-0.146	7916.99
A.14	-1.00	-0.27	-0.060	3920.00	-1.00	-0.35	-0.147	8827.30
A.22	1.55	0.17	-0.141	760.95	1.55	0.17	-0.142	7909.60

Table 6.8 Add/drop results for the Averitt data set.

superior to the HLP and the add/drop on the HLP. Though it may look odd, it is quite true that the number of handling and service errors increases in the solution of the add/drop on the HLP. This is due to the fact that the add/drop heuristic is based on finding shortest paths in terms of cost, not time. So, the very first step of the add/drop heuristic is to find a solution using the shortest paths in terms of cost along only the arcs of the inputted SLP solution. Generally this solution is quite similar to the inputted load plan, though it is not always as close as we would desire. The add/drop heuristic cannot consider both cost and time tradeoffs as well as the UND–based algorithm; and in this case, the handling and service errors increase as a result. It is interesting to note, as well, that the time it takes to run the complete algorithm is comparable to the time it takes to run just the add/drop on the HLP. Thus, in approximately the same amount of time, the UND–based method found a better solution.

6.3.3 Daiichi Data Set

The final set of data was provided by the Japanese carrier Daiichi Freight System. This network containing 92 terminals is the largest of our three data sets. Due to its size, computation on this data set had to be performed on a larger machine. All of the computation on the Daiichi data set was performed on a Sun MicroSystems i86pc 200 MHz machine with 1024 megabytes of memory.

As in the Averitt case, the Daiichi data did not include a historical load summary. The HLP for the Daiichi data contains many more handling and service violations than the previous HLP solutions, leaving significant room for improvement. In this case we either prune all of the arcs not in the HLP, or all of the EOL–to–EOL arcs not in the HLP. The significant difference in the number of arcs in the network is apparent from Table 6.9. Using the low freight pruning technique to its full potential

	Parameter Settings			Results			
Case	Prune	Frt.D.	Handtime	Handling	Service	Objective	Time (s)
D.1	1.00	1.0	0	-0.87	0.10	-0.180	3554.78
D.2	1.00	1.0	60	-0.88	0.10	-0.169	3520.67
D.3	1.00	1.0	120	-0.88	0.10	-0.161	3477.18
D.4	1.00	1.1	0	-0.89	0.10	-0.180	3621.57
D.5	1.00	1.1	60	-0.90	0.10	-0.166	3577.00
D.6	1.00	1.1	120	-0.90	0.10	-0.157	3530.64
D.7	1.00	1.2	0	-0.91	0.09	-0.179	3708.29
D.8	1.00	1.2	60	-0.92	0.10	-0.166	3650.26
D.9	1.00	1.2	120	-0.92	0.10	-0.156	3599.61
D.10	0.02	1.2	0	-0.99	-0.74	0.580	26891.51
D.11	0.02	1.2	60	-1.00	-0.74	0.665	26334.59
D.12	0.02	1.2	120	-1.00	-0.76	0.759	25780.46

Table 6.9 UND–based results for the Daiichi data set.

only eliminates 2% of the prunable arcs. Clearly, the very small percentage of EOL terminals in this set of data contributes to its large network size. Fewer tests were done in conjunction with the EOL pruning technique since the considerable number of arcs prevented finding good overall solutions. The results from the Daiichi data clearly indicate the importance of an intelligent pruning technique. While the solutions obtained by considering only the HLP arcs are certainly superior to the HLP solution, it is likely that a better pruning technique based on more data from the company would provide even better quality solutions.

	UND–based SLP solution				Add/Drop SLP solution			
Case	Handling	Service	Objective	Time (s)	Handling	Service	Objective	Time (s)
HLP	-	-	-	-	-0.99	-0.04	-0.288	31647.09
D.7	-0.91	0.09	-0.179	3708.29	-0.99	-0.05	-0.281	36504.32
D.8	-0.92	0.10	-0.166	3650.26	-0.99	-0.05	-0.281	37357.59
D.9	-0.92	0.10	-0.156	3599.61	-0.99	-0.05	-0.283	37089.78

Table 6.10 Add/drop results for the Daiichi data set.

Table 6.10 indicates the comparable solution quality of the add/drop solutions for both UND–based solutions and the HLP. Unfortunately, though, the add/drop heuristic on a network of this size takes 8 to 10 hours. Since, the UND–based algorithm only requires approximately one hour to return an improved solution, it is likely that this method, combined with an intelligent pruning technique, is a better alternative to a purely add/drop approach for a company whose network is as large as this one.

6.4 Remarks

The complete algorithm, consisting of a modified uncapacitated network design method and an add/drop procedure, found good solutions to the strategic load planning problem for the data sets tested. These solutions are found in a reasonable amount of time and are qualitatively on par with the historical load plans implemented by the carriers. The UND-based portion of the algorithm consistently improved upon the initial solution given, as did the add/drop procedure. Surprisingly, however, when the add/drop procedure is applied to the historical load plans from the SEFL carrier, the solution generated is better than the solutions generated from add/drop when applied to the UND-based solutions. This result holds regardless of whether or not the UND-based solutions input into the add/drop procedure were better than the historical load plan. We do not believe that this result reflects negatively on our algorithm, but that it strongly confirms the quality of the time-tested solution used by SEFL.

In terms of future work, a better pruning technique could improve the algorithm's effectiveness considerably. Currently, we rely on historical data provided by the carriers for our most effective pruning heuristic. However, we would like to able to ascertain which directs are reasonable or unreasonable based upon real-world considerations that apply generally to all carriers or specifically for an individual carrier. As is commonly the case when considering real–world applications, the amount of data available is key to the solution quality.

The terminal capacities were ignored in our algorithm because the company from which our primary data came was interested in deciding whether or not to add to an existing terminal or build a new one. Ignoring terminal capacities becomes a potential problem if a carrier decides that this should be a hard constraint. If necessary, in such cases, the current, though unused, capacity heuristic can be improved by basing the cost multiplier on the degree of the capacity violation as opposed to using a constant multiplier.

Lastly, the issue of actively improving minimum frequency was not fully addressed. Currently, if the total trailer-loads of freight plus the number of empty trailers on a direct meets the minimum frequency requirement, then we assume that we can safely reduce the fixed cost associated with this direct to zero. On the other hand, if the amount of freight and the number of empty trailers falls far below the minimum frequency on an arc, then we do not change the fixed costs for the arc. For arcs that fall between these two extremes the fixed costs could be modified as a function of the difference between the total number of trailers assigned (loaded and empty) and the minimum frequency.

Chapter 7

Conclusions

In this thesis, three separate routing applications are considered. The first instance is a problem from the American home health care industry in which health care providers must be scheduled to visit a set of patients. The second is a standard vehicle routing problem with time windows that represents distribution needs throughout a variety of industries. And the final problem is that of solving the strategic load–planning problem for less–than–truckload carriers. The common goals in all of these applications are to reduce cost and increase efficiency within the distribution network.

The work presented in this thesis on the home health care model indicates that a heuristic approach is better suited to this problem. The implemented heuristic behaves well and generates good solutions. The heuristic approach, though, is difficult to evaluate without more complete data. Although an exact approach using a MIP formulation is easier to evaluate, it is unable to handle problems of the size required by real–world problems. Further work on branching rules, as well as possible improvements to the formulation, is required if larger problems are to be considered. This is a relatively new area in which optimization techniques are being considered, and the savings potential is high for those companies willing to implement the appropriate methods. However, there are several obstacles which must be overcome. In particular, there is a lack of data, and there is the potential that nurses may be unwilling to release control over their schedules to a third party.

Although this thesis does not present a specific application of the VRPTW, many advancements in solving real–world problems are derived from solution methods for solving instances from the set of Solomon benchmark problems. By improving the solution method for finding 2–path cuts and by extending this approach to a separation routine for k–path cuts, we are able to solve three previously unsolved problems. In conjunction with these new methods, we illustrate the advantages to be gained from using a parallel implementation of the branch–and–cut procedure. Seven additional unsolved problems are solved using the k–path cutting plane approach and a network of computers to perform the branching. In total, our solution procedure solves eighty out of eighty–seven of the Solomon problems, ten of which were previously unsolved in the literature.

Finally, the solution strategy developed in this thesis for the LTL carriers provides an effective means of creating a cost-effective network. Data from three different trucking companies supports the premise that the UND–based algorithm is a worthwhile tool in developing a strategic load plan. The add/drop heuristic is also clearly a beneficial procedure, and one which in the future can be better tailored to specific company goals. Overall, though, the issue of data availability can greatly effect the quality of the final solution. There is strong evidence indicating that additional information used in an intelligent pruning technique will yield better solutions in less time. Even without additional data, though, the solution method succeeds in finding improved strategic load plans for each of the three LTL carriers.

Appendix A

A new transformation of the symmetric m–TSP

Given a set of cities represented by vertices in a symmetric graph $G = (V, E)$, the m-TSP is the problem of finding the least-cost set of routes originating and terminating at a single depot which visits every other city exactly once. Let $V = \{0, 1, \ldots, n-1\}$ be the set of vertices (such that the depot is vertex 0) and E be the set of edges. Also let c_{ij} be the cost of an edge $(i, j) \in E$. We will consider only the case when the number of routes, m, is free (that is, $1 \leq m \leq n-1$). There must exist an edge between the depot and every other vertex in the graph; in other words, it is a feasible solution to have $n-1$ routes all of which start and end at the depot and visit only one vertex. Given such an m-TSP, we will prove that there exists a transformation of this problem into a standard TSP on a graph with the same number of vertices.

First, we will give two integer linear programming formulations for the m-TSP. Let x_{ij} defined on $\{(i, j) \in E : i, j \in \{1, 2, \ldots, n-1\}, i < j\}$ be a binary variable equal to 1 if edge (i, j) appears in the solution, and equal to 0 otherwise. Since we assume that the edge costs are symmetric, we can define x_{ij} only for $i < j$, although we may use x_{ij} and x_{ji} to refer to the same variable. It is assumed throughout the discussion, that $i \neq j$. Also let x_{0i} be an integer variable equal to 2 if i is alone on a single route, equal to 1 if i is the first or last vertex on a route with 2 or more non-depot vertices, and equal to 0 otherwise. Thus, the following is a valid formulation of the m-TSP:

$$\text{Minimize} \quad d = \sum_{(i,j)\in E} c_{ij} x_{ij} \tag{A.1}$$

$$\text{subject to} \quad \sum_{j=1}^{n-1} x_{ij} + x_{0i} = 2, \qquad (i = 1, \ldots, n-1) \tag{A.2}$$

$$x(\gamma(S)) \leq |S| - 1, \qquad (S \subseteq \{1, \ldots, n-1\}) \tag{A.3}$$

$$x_{ij} = 0 \text{ or } 1, \qquad ((i,j) \in E, 1 \leq i < j \leq n-1) \tag{A.4}$$

$$x_{0i} = 0, 1, \text{ or } 2, \qquad (i = 1, \ldots, n-1) \tag{A.5}$$

Constraints (A.2) are the degree constraints which ensure that every city is visited by exactly one route. Constraints (A.3) are the well-known subtour elimination constraints. These are used to eliminate routes that do not visit the depot. Now, note that the x_{0i} variables only appear in the degree constraints. These variables can be treated as slack variables and eliminated from the formulation. This yields the following:

$$\text{Minimize} \qquad d = 2\sum_{i=1}^{n-1} c_{0i} + \sum_{(i,j)\in E, 1\leq i<j\leq n-1} (c_{ij} - c_{0i} - c_{0j})x_{ij} \tag{A.6}$$

$$\text{subject to} \qquad \sum_{j=1}^{n-1} x_{ij} \leq 2, \qquad (i = 1, \ldots, n-1) \tag{A.7}$$

$$x(\gamma(S)) \leq |S| - 1, \qquad (S \subseteq \{1, \ldots, n-1\}) \tag{A.8}$$

$$x_{ij} = 0 \text{ or } 1, \qquad ((i,j) \in E, 1 \leq i < j \leq n-1) \tag{A.9}$$

Clearly the first term in the objective function is a constant which can be removed from the formulation and the substitution $c'_{ij} = c_{ij} - c_{0i} - c_{0j}$ can be used. Thus, the new objective can be written as:

$$\text{Minimize} \qquad s = \sum_{(i,j)\in E, 1\leq i<j\leq n-1} c'_{ij}x_{ij} \tag{A.10}$$

Now the objective functions are related by the formula: $d = 2\sum_{i=1}^{n-1} c_{0i} + s$. The values of the c' terms correspond exactly to the negative of the savings term s_{ij} introduced by Clarke and Wright [20] to represent the savings in cost that would result from linking i and j in a route rather than visiting each separately from the depot. Due to this connection, we will refer to this formulation (A.6, A.7 – A.9) as the *savings* form of the problem. This problem can be solved looking only at the reduced graph in which $G' = (V', E')$ where $V' = V \backslash \{0\}$ and the edge set E' is the set of edges from E with both endpoionts in V'.

Now, consider a solution vector x to the savings IP. The x-graph induced by a vector x is the graph $G^* = (V', E^*)$ such that $E^* = \{(i,j) \in E' : x_{ij} = 1\}$. Clearly, any feasible solution must be a collection of paths and single vertices in the x-graph. A single vertex can actually be thought of as a path with no edges, so the solution is a collection of paths. This is true since constraints (A.7) restrict the set of solutions to those in which the corresponding x-graph contains vertices with only 0, 1, or 2 edges adjacent to a vertex. Constraints (A.8) guarantee that there are no cycles in an x-graph corresponding to a feasible solution.

Since the zero vector is a solution to the savings formulation, the optimal value Z is bounded above by zero. Clearly, any solution using a positive cost edge can be improved by simply not using that edge. Thus, the problem may be reduced by deleting all edges (i,j) such that $c'_{ij} > 0$. Let this reduced graph be denoted by $\overline{G}' = (V', \overline{E}')$. Although the set of feasible solutions has been reduced by this transformation, the set of optimal solutions remains the same. Thus, the following theorem holds.

Theorem A.1 A solution to the savings formulation of the m-TSP is optimal on graph G' if and only if it has a corresponding optimal solution on the graph $\overline{G}'$.

Proof Suppose x is an optimal solution to the savings formulation of the m-TSP on the graph G' with objective value Z. *Claim:* every edge (i,j) such that $x_{ij} = 1$ must have cost $c_{ij} \leq 0$. If some edge (i,j) has a positive cost and $x_{ij} = 1$, then setting $x_{ij} = 0$ yields a new feasible solution with a smaller objective value; this is a contradiction. So, the same solution in $\overline{G}'$ corresponds to the x solution extended so that $x_{ij} = 0$ for all edges (i, j) not in G'. The objective value, Z, remains the same. Suppose this solution is not optimal for $\overline{G}'$. This implies that there exists a feasible solution to the savings formulation on the graph $\overline{G}'$ with an objective value $W < Z$. Deleting all zero-cost edges in the new solution yields a feasible solution to the problem on G' with objective value $W < Z$. This contradicts the original assumption that x is optimal on G'; thus, the solution must be optimal on $\overline{G}'$.

Now, suppose that x is an optimal solution to the savings formulation of the m-TSP on the graph $\overline{G}'$ with objective value Z. The corresponding solution on the graph G' is the x vector defined only over the edges in G'. Since the only edges not in both graphs have cost zero in $\overline{G}'$, the objective value remains the same. Suppose the solution is not optimal on G'. Then, there must exists a feasible solution with a smaller objective value. But, this solution must also be feasible on the graph $\overline{G}'$ with the same (lower) objective value. This is a contradiction. So, the solution on G' must be optimal. $\square$

Thus, solving the m-TSP can be done by finding the optimal solution to the savings formulation on the graph $\overline{G}'$. Now, let G'' be the complete graph obtained from adding a zero–cost edge (i, j) to the graph $\overline{G}'$ for every pair of vertices i, j such that $i \neq j$ and $(i, j) \notin \overline{E}'$. Thus, given a Hamiltonian path in G'', a corresponding solution to the savings formulation of the m-TSP with the same objective value can be obtained by simply deleting the zero-cost edges in the solution. So, finding the optimal Hamiltonian path in G'' will find the optimal solution to the savings formulation on the graph $\overline{G}'$. Finally, it is well known that the optimal Hamiltonian path problem can be transformed into an equivalent TSP problem by adding a single node. Thus, the symmetric m-TSP problem on n vertices with m defined as a free variable can be solved by finding the optimal solution to a traveling salesman problem on a complete graph with n vertices.

Appendix B

A two–index formulation of the nursing problem

As was previously mentioned, this problem has a formulation that uses only O(n^2) integer variables. For example, Kulkarni and Bhave [48], with corrections by Laporte [50], and Laporte et al. [57] gave such formulations for other vehicle routing problems. We now provide an alternative formulation applicable to our problem.

$$\text{Minimize } \alpha \sum_{n \in F} (z_n - y_n + s_n - d_{L(n)} - SL) + \beta \sum_{n \in H} (z_n - y_n) \tag{B.1}$$

subject to

$$x(\delta(L(n))) = 1 \qquad n \in F \tag{B.2}$$

$$x(\delta(L(n))) = (1 - x_{DP(n),n}) \qquad n \in H \tag{B.3}$$

$$x(\delta(i)) - \sum_{k \in L} x_{ik} = x(\rho(i)) = 1 \qquad i \in N \cup P \tag{B.4}$$

$$y_i + t_{ij} - y_j \leq (b_i + t_{ij} - a_j)(1 - x_{ij}), \qquad i, j \in P, \text{ or } i \in N, j \in P \cap B_i, i \neq j \tag{B.5}$$

$$d_{L(n)} x_{nL(n)} + y_i + t_{in} - z_n \leq (d_{L(n)} + b_i + t_{in} - a_n)(1 - x_{in}), \; n \in N, i \in P \cap B_n \tag{B.6}$$

$$y_j + d_j - y_i \leq (b_j + d_j - a_i)(1 - x_{ij}), \qquad i \in P, j \in L \tag{B.7}$$

$$y_{L(n)} + d_{L(n)} - z_n \leq (b_{L(n)} + d_{L(n)} - a_n)(1 - x_{nL(n)}), \qquad n \in N \tag{B.8}$$

$$y_i + t_{ij} - y_k \leq (b_i + t_{ij} - \frac{1}{2} a_k)(2 - x_{ij} - x_{jk}) \qquad i \in N \cup P, j \in P, k \in L, i \neq j \tag{B.9}$$

$$y_i + t_{in} - y_{L(n)} \leq (b_i + t_{in} - \frac{1}{2} a_{L(n)})(2 - x_{in} - x_{nL(n)}), \qquad i \in P, n \in N \tag{B.10}$$

$$y_n + d_{L(n)} \leq z_n, \qquad n \in F \tag{B.11}$$

$$y_n \leq z_n, \qquad n \in H \tag{B.12}$$

$$z_n - y_n + s_n - d_{L(n)} - SL \geq 0, s_n \geq 0, \qquad n \in F \tag{B.13}$$

$$z_n - y_n \leq MSL, \qquad n \in N \tag{B.14}$$

$$\pi_j^n - \pi_i^n \leq 1 - x_{ij}, \qquad (i,j) \in A, n \in N \tag{B.15}$$

$$\pi_j^n - \pi_n^n \geq 1, \qquad n \in N, j \in (P \cup N \cup L) \backslash B_n \tag{B.16}$$

$$x_{nDP(n)} = 1, \qquad n \in N \tag{B.17}$$

$$a_n \leq y_n, z_n \leq b_n \qquad n \in N \tag{B.18}$$

$$a_i \leq y_i \leq b_i, \qquad i \in P \cup L \tag{B.19}$$

$$x_e \in \{0, 1\}, \qquad e \in A \tag{B.20}$$

Note that a solution from the triple–indexed formulation could be made feasible to the above formulation with the following assignment: $x_{ij} = \sum_{n \in N} x_{nij}$. Without a separate arc for each nurse between two points in the graph in the double–indexed formulation, we must now restrict the solution set so that each main circuit has only one (feasible) nurse node and only one "tail." This is guaranteed by the shortest paths constraints (B.15) and (B.16). To see this, let $w_e = 1 - x_e$. Let v a particular nurse-node and u be any other node. Then they are in the same component if and only if there exists a path of weight 0 from v to u. Since the weight function is integral, if a path is not of weight 0, it is of weight at least 1. So they are in different components if and only if the shortest path from v to u is at least 1. Let $\{\pi_t : t \in V\}$ be a set of values, known as *feasible potentials*, satisfying $\pi_b - \pi_a \leq w_{a,b}$ for every edge (a, b). Then it is well-known (and easy to show) that the value of a shortest path from v to u is at least $\pi_u - \pi_v$. Hence if we have a set of feasible potentials with the extra condition that $\pi_u - \pi_v \geq 1$, then u and v are in different components.

Bibliography

[1] Amza, C., A.L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu and W. Zwaenepoel. TreadMarks: Shared Memory Computing on Networks of Workstations. *IEEE Computer* vol 29 no 2 (1996) 18–28.

[2] Anderson, R.J., and J.C. Setubal. "Goldberg's algorithm for maximum flow in perspective: a computational study", in D.S. Johnson and C.C. McGeoch (eds.), *Network Flows and Matching*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol 12, American Mathematical Society (1993) 1–18.

[3] Assad, A.A. Modeling and implementation issues in vehicle routing, in B. Golden and A.A. Assad (eds.), *Vehicle Routing: Methods and Studies*, North–Holland, Amsterdam (1988) 7–45.

[4] Balakrishnan, A., T. L. Magnanti, and P. Mirchandani. Network Design. *Annotated Bibliographies in Combinatorial Optimization* (1997) 311–334.

[5] Balakrishnan, A., T. L. Magnanti, and R.T. Wong. A Dual–Ascent Procedure for Large–Scale Uncapacitated Network Design. *Operations Research* 37:5 (1989)716–740.

[6] Bellmore, M., and S.Hong. Transformation of multisalesmen problem to the standard travelling salesman problem, *Journal of the Association for the Computing Machinery* 21 (1974) 500–504.

[7] Bellmore, M., and J.C. Malone. Pathology of traveling salesmen subtour–elimination algorithms, *Operations Research* 19 (1971) 278–307.

[8] Bodin, L., and B. Golden. Classification in vehicle routing and scheduling, *Networks* 11 (1981) 97–108.

[9] Braklow, John W., W. W. Graham, S. M. Hassler, K. E. Peck, and W. B. Powell. Interactive Optimization Improves Service and Performance for Yellow Freight System. *Interfaces* 22 (1992) 147–172.

[10] Bramel, J. and D. Simchi–Levi. On the Effectiveness of Set Covering Formulations for the Vehicle Routing Problem with Time Windows. *Operations Research* 45, 2 (1997) 295–301.

[11] Breedam, A.V. Improvement heuristics for the vehicle routing problem based on simulated annealing, *European Journal of Operational Research* 86 (1995) 480–490.

[12] Carpaneto, G., M. Dell'Amico, M. Fischetti, and P. Toth. A branch and bound algorithm for the multiple depot vehicle scheduling problem. *Networks* 19 (1989) 531–548.

[13] Caseau, Y., and P. Koppstein. A cooperative–architecture expert system for solving large time/travel assignment problems, *International Conference on Databases and Expert Systems Applications*, Valencia, Spain, September 1992.

[14] Chao, I.–M., B.L. Golden, and E. Wasil. A new heuristic for the multi–depot vehicle routing problem that improves upon best–known solutions, *American Journal of Mathematical and Management Sciences* 13 (1993) 371–406.

[15] Christofides, N. Vehicle scheduling and routing, *presented at the 12th International Symposium of Mathematical Programming*, Cambridge, Massachusetts (1985).

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

[17] Christofides, N., A. Mingozzi, and P. Toth. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations, *Mathematical Programming* 20 (1981) 255–282.

[18] Christofides, N., A. Mingozzi, and P. Toth. State–space relaxation procedures for the computation of bounds to routing problems, *Networks* 11 (1981) 145–164.

[19] Chvátal, V. Edmonds polytopes and weakly hamiltonian graphs, *Mathematical Programming* 5 (1973) 29–40.

[20] Clarke, F. and J. Wright. Scheduling of vehicles from a central depot to a number of delivery points, *Operations Research* 12 (1964) 568–581.

[21] Cook, W., Cunningham, W.H., Pulleyblank, W.R., and Schrijver, A., *Combinatorial Optimization*, Wiley, New York, 1998.

[22] *Using the CPLEX Callable Library,* Version 5.0. CPLEX Optimization, Incline Village, NV, 1997.

[23] Dantzig, G.B., D.R. Fulkerson, and S.M. Johnson, Solution of a large scale traveling salesman problem, *Operations Research* 2 (1954) 393–410.

[24] Desrochers, M. An Algorithm for the Shortest Path Problem with Resource Constraints, *Les Cahiers du GERAD* G–88–27, Écoles des Hautes Études Commerciales, Montreál, Canada.

[25] Desrochers, M., J. Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows, *Operations Research* 40 (1992) 342–354.

[26] Desrochers, M., G. Laporte. Improvements and extensions to the Miller–Tucker–Zemlin subtour elimination constraints, *Operations Research Letters* 10 (1991) 27–36.

[27] Desrosiers, J., F. Soumis, and M. Desrochers. Routing with time windows by column generation. *Networks* 14 (1984) 545–565.

[28] Desrochers, M., F. Soumis. A Generalized Permanent Labelling Algorithm for the Shortest Path Problem with Time Windows. *INFOR*, vol 26 (1988) 191–211

[29] Dumas, Y., J. Desrosiers, E. Gélinas, and M. Solomon. An Optimal Algorithm for the Traveling Salesman Problem with Time Windows, *Operations Research* 43 (1995) 367–371.

[30] Eilon, S., C.D.T. Watson–Gandy, and N. Christofides. *Distribution Management: Mathematical Modeling and Practical Analysis*, Griffin (1971).

[31] Fischetti, M., P. Toth, and D. Vigo. A branch–and–bound algorithm for the capacitated vehicle routing problem on directed graphs, *Operations Research* 42:5 (1994) 846–859.

[32] Fisher, M.L. Vehicle Routing, *Network Routing, Handbooks in Operations Research and Management Science, Vol. 8* (Ball, Magnanti, Monma, Nemhauser, eds.) Elsevier Science, Amsterdam (1995) 1–33.

[33] Fisher, M.L., K.O. Jörnsten, and O.B.G. Madsen, Vehicle Routing with Time Windows: Two Optimizations Algorithms, *Operations Research* 45 (1997) 488–492.

[34] Fisher, M. L. Optimal solution of vehicle routing problems using minimum K–trees, *Operations Research* 42:4 (1994) 626–642.

[35] Fisher, M.L., R. Jaikumar. A generalized assignment heuristic for vehicle routing, *Networks* 11 (1981) 109–124.

[36] Florkiewicz, B., and M. Kulej. Feasibility and optimality of time schedules in vehicle routing and scheduling problem with time window constraints, *Central European Journal of Operations Research and Economics* 2:1 (1993) 65–78.

[37] Garey, M.R., and D.S. Johnson. *Computers and Intractability: A guide to the theory of NP–Completeness.* W.H. Freeman and Company, New York (1979).

[38] Glover, F., and M. Laguna. *Tabu Search.* Kluwer Academic Publishers (1997).

[39] Grötschel, M. and Holland, O. Solution of large–scale symmetric travelling salesman problems. *Mathematical Programming* 51 (1991) 141–202.

[40] Grötschel, M. and M.W. Padberg. On the symmetrical travelling salesman problem II: Lifting theorems and facets, *Mathematical Programming* 16 (1979) 282–302.

[41] Houck Jr, D.J., J–C. Picard, M. Queyranne, and R.R. Vemuganti. The Travelling Salesman Problem as a Constrained Shortest Path Problem: Theory and Computational Experience. *Opsearch* 17 (1980) 93–109.

[42] Karger, D. R. Global min–cuts in $\mathcal{RNC}$ and other ramifications of a simple mincut algorithm. *Proceedings of the 4th Annual ACM–SIAM Symposium on Discrete Algorithms* ACM–SIAM, (1993) 84–93.

[43] Karger, D. R. and C. Stein. An $\tilde{O}(n^2)$ Algorithm for Minimum Cuts. *Proceedings of the25th ACM Symposium on the Theory of Computing*, ACM Press (1993) 757–765.

[44] Kearney, A. Improving productivity in physical distribution, *Report undertaken for CPDM, London* (1980).

[45] Kohl, N. Exact Methods for Time Constrained Routing and Related Scheduling Problems, Ph.D. Dissertation no. 16, Institute of Mathematical Modelling, Technical University of Denmark, DK-2800 Lyngby, Denmark.

[46] Kohl, N., J. Desrosiers, O.B.G. Madsen, M.M. Solomon, and F. Soumis. k–Path cuts for the vehicle routing problem with time windows, *Technical Report*, IMM–REP–1997–12, (1997) The Technical University of Denmark, Lyngby.

[47] Kolen, A.W.J., A.H.G. Rinnooy Kan, and H.W.J.M. Trienekens. Vehicle routing with time windows. Operations Research Vol. 35, No. 2, March–April 1987.

[48] Kulkarni, R.V., and P.R. Bhave. Integer programming formulations of vehicle routing problems. *European Journal of Operations Research* 20 (1985) 58–67.

[49] Laporte, G. The vehicle routing problem: An overview of exact and approximate algorithms, *European Journal of Operational Research* 59 (1992) 345–358.

[50] Laporte, G. Integer programming formulations for the multi–depot vehicle routing problem: comments on a paper by Kulkarni and Bhave: "Integer programming formulations of vehicle routing problems," *European Journal of Operational Research* 38:2 (1989) 227.

[51] Laporte, G. and Y. Nobert. Exact algorithms for the vehicle routing problem, *Annals of Discrete Mathematics* 31 (1987) 147–184.

[52] Laporte, G., and Y. Nobert. A cutting planes algorithm for the m–salesmen problem, *The Journal of the Operational Research Society* 31:11 (1980) 1017–1023.

[53] Laporte, G., Y. Nobert, and D. Arpin. Optimal solutions to capacitated multi-depot vehicle routing problems, *Congressus Numerantium* 44 (1984) 283–292.

[54] Laporte, G., Y. Nobert, and M. Desrochers. Optimal routing under capacity and distance restrictions, *Operations Research* 33 (1985) 1050–1073.

[55] Laporte, G., Y. Nobert, and T.K. Nga. An optimal flow circulation algorithm for the asymmetrical multiple travelling salesman problem, *Congressus Numerantium* 57 (1987) 235–248.

[56] Laporte, G., Y. Nobert, and P. Pelletier. Transformations of the multiple travelling salesman problem into a single travelling salesman problem, *Proceedings of the Administrative Sciences Association of Canada, vol 2, Management Science*, G. Pederzoli, ed. (1981) 37–44.

[57] Laporte, G., Y. Nobert, and S. Taillefer. Solving a family of multi–depot vehicle routing and location–routing problems, *Transportation Science* 22:3 (1988) 161–172.

[58] Lin, S. Computer solutions of the traveling salesman problem, *Bell Systems Computer Journal* 44 (1965) 2245–2269.

[59] Lin, S., and B.W. Kernighan. An effective heuristic algorithm for the traveling salesman problem, *Operations Research* 21 (1973) 498–516.

[60] Miller, C.E., A.W. Tucker, and R.A. Zemlin. Integer programming formulations and traveling salesman problems, *Journal of the Association for Computing Machinery* 7 (1960) 326–329.

[61] Padberg, M.W. and M. Grötschel. Polyhedral Computations, in Lawler, E.L, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys (eds.), *The Traveling Salesman Problem* Wiley, Chichester (1985) 307–360.

[62] Sansó, B., M. Desrochers, J. Desrosiers, Y. Dumas, and F. Soumis. Modeling and solving routing and scheduling problems: GENCOL user guide, GERAD, 5255 Decelles, Montréal (1990) 63–64.

[63] Solomon, M. Algorithms for the Vehicle Routing and Scheduling Problem with Time Window Constraints, *Operations Research* 35 (1987) 254–265.

[64] Svestka, J.A. and V.E. Huckfeldt. Computational experience with an m–salesman traveling salesman algorithm, *Management Science* 19:7 (1973) 790–799.