SOLVING REAL-LIFE TRANSPORTATION SCHEDULING PROBLEMS

By

JIAN LIU

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2003

This document is dedicated to my wife ...

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor, Dr. Ravindra K. Ahuja, for his patient guidance, constant encouragement, endless support and constructive criticism during my study. His profound knowledge and deep understanding of transportation networks set a high bar for me to reach in the future.

I would like to acknowledge the help of Dr. Dharma Acharya, the General Manager of Operations Research at CSX Transportation, and Dr. Pooja Dewan, the Manager of Operations Research in the Technology Services Department at BNSF Railway. I gratefully appreciate their help for giving me the access to their data and assessing the implementability of the algorithmic approaches developed in the dissertation.

I also want to express my gratitude to Dr. Sartaj K. Sahni, Dr. Joseph P. Geunes and Dr. Zuo-Jun Shen for serving as my supervisory committee members and for giving me wonderful ideas and advice.

Special thanks are owed to Krishna C. Jha for his generous help with editing the original edition.

Finally, I am grateful to my parents and my wife for their love, support and encouragement.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

SOLVING REAL-LIFE TRANSPORTATION SCHEDULING PROBLEMS

By

Jian Liu

August, 2003

Chair: Dr. Ravindra K. Ahuja
Major Department: Industrial and Systems Engineering

Transportation by railroads and airlines contains a rich set of optimization problems with substantial potential savings in transportation costs. In the past few decades, unfortunately, optimization models were not widely used in transportation industries, because of (i) the large size and tremendous complexity of these problems, (ii) the lack of suitable algorithmic approaches for solving them, and (iii) insufficient computing power available. However, as major advances have taken place in algorithm design, analysis and implementation, complemented by enhanced computer systems, transportation scheduling problems now appear to be tractable.

The goal of this dissertation is to study several real-life transportation scheduling problems that are of great importance for railroads and airlines. The related literatures of these problems have only dealt with simplified models or small instances failing to incorporate the characteristic of real-life applications. In Chapter 2, we present an integrated model for the *locomotive scheduling problem.* In Chapter 3, we propose two approaches to solve the *railroad blocking problem.* In Chapters 4 and 5, we study

extensions and generalizations of *combined through and fleet assignment models*. The focus of this dissertation is to model these problems with realistic constraints and solve the real-life instances of those models with modern optimization techniques. The major solution approaches developed in this dissertation are based on *Very Large Scale Neighborhood* (VLSN) search, which is a heuristic approach but works very well for real-life instances. The computational tests for those problems are performed on real-life data from major U.S. transportation carriers. The results reveal that the models and solution approaches developed in this dissertation are practically implementable and capable of generating significant economic impact on transportation industries.

CHAPTER 1
INTRODUCTION

Transportation is one of the most vital services in modern society. In the U.S. and Europe, transportation service industries account for between 4% and 7% of overall GNP. The U.S. transportation system is an extensive, inter-related network, and consists of around 46.5 thousand miles of interstate highways, 5 thousand airports, 171 thousand miles of freight railroads and 26 thousand navigable waterways. In 1999, the transportation network supported 4.8 trillion passenger-miles and 3.9 trillion ton-miles; and around 8228 air crafts and 1.38 million freight cars are involved. In 2000, transportation related goods and services contributed $1050 billion (10.8 %) to a $9.87 trillion U.S. gross domestic product.

With so many resources involved, the transportation industry encompasses numerous decision problems which can be formulated mathematically. One important kind of decision problem is called the *scheduling problem*, which aims to plan for the movement of passengers and freight efficiently in constrained environments with the given resources. The transportation scheduling problem has attracted many researchers in the past due to its interesting nature and economic scale. Some of those researchers are involved in studying the mathematical nature of the problem and in theoretical approaches for solving the problem, whereas others have tried to solve real-life problems and to produce implementable schedules. In this dissertation, we present our effort on modeling and solving several real-life railroad and airline scheduling problems.

This chapter is organized as follows. In Section 1.1 and 1.2, we describe some general scheduling problems in railroad and airline transportation in detail. In Section 1.3, we give the summary and outline of this dissertation.

1.1 Railroad Scheduling Problems

In the U.S. transportation industry, railroads play a major role in freight movement. Railroads transport over 40% of the ton-miles of intercity freight, including 70% of new automobiles, 40% of farm products and 65% of coal (Association of American Railroads, 1999). Many railroad transportation problems can be modeled and solved using mathematical optimization techniques. Unfortunately, the related research was not very successful in the past, failing to incorporate the characteristics of real-life applications (Assad [1981] and Haghani [1987]). The development of optimization models for routing of trains and freight was for a long time hindered by the large size of the problems and lack of optimization software to solve problems of that magnitude. In addition, the computing capabilities of computers were limited. As a result, practical implementations of optimization models often had a limited success, which deterred both researchers and practitioners from pursuing the effort. However, in recent years, with increased profit incentives, strong competition from other freight carriers (especially trucking companies), and better availability of computer systems, railroads have begun to adopt optimization-based decision support for their scheduling problems with some success (Cordeau et al. [1998]). Following are some typical railroad scheduling problems that have appeared in the operations research literature with increased level of realism.

1.1.1 Railroad Blocking Problem

Railroads carry millions of shipments annually from their origins to their respective destinations. A typical shipment is composed of a set of individual cars that all share a

common origin and destination. To reduce the intermediate handling of shipments as they travel over the railroad network, a set of shipments is classified (or grouped) together to create a *block*. The blocking problem is to identify this classification plan for all shipments in this network, called a blocking plan, so that the total shipment cost is minimum. A blocking plan significantly affects the shipment cost. However, the blocking problem is a very large-scale optimization problem, and its complexity and sheer size have not allowed this problem to be solved to optimality or near-optimality satisfying all the practical considerations required for an implementable plan. The two most recent references on blocking problems are by Newton et al. [1998] and Barnhart et al. [2000]. We present our study on the blocking problem in Chapter 3.

1.1.2 Freight Car Management Problem

Every loaded movement on a rail network leads to a supply of empty cars at the destinations. Therefore, if transportation demand is unbalanced, steps must be taken to reposition empty cars and avoid their accumulation in some parts of the network where more traffic is directed. Repositioning empty freight cars can thus help the railroad offer better service to its customers and decrease the capital investment associated with equipment ownership. The freight car management problem consists of dynamically distributing empty cars in the network to improve the railroad's ability to promptly answer requests for empty cars while minimizing the costs associated with their movement. Some recent papers on this problem are by Crainic et al. [1990], Powell et al. [1995], and Nozick and Morlok [1997].

1.1.3 Train Routing, Timetabling and Dispatching Problem

Given a blocking plan and a distribution plan for empty cars, the train routing and time-tabling problem is to identify train routes and their time-tables so as to minimize the

cost of carrying cars from their origins to destinations. Real-time operations of trains also require synchronization of train movements on the lines of the physical railway network. In the U.S., lines are often made of a single track. To allow trains traveling in different directions on a single track, sidings are located at regular intervals along the line. These short track sections allow one train to pull over and free the way for the other one. Sidings are also used to permit a fast train to pass a slower one. Given a train timetable, the train dispatching problem determines a feasible plan of meets and overtakes that satisfies a system of constraints on the operation of trains. Some recent papers on this problem are by Farvolden and Powell [1994], Campbell [1996], Kraft [1998], and Brannlund et al. [1998].

1.1.4 Crew Scheduling Problem

Once the train timetable is planned, the next step is to assign required personnel to perform train services (the actual journey with freight and the transfer of empty trains or equipment between stations). The crew management is concerned with building the work schedules of crews needed to perform train services for a planned train timetable. Each train service has first been split into a sequence of trips, defined as segments of train journeys which must be serviced by the same crew without rest. Each trip is characterized by a departure time, a departure station, an arrival time, an arrival station, and possibly by additional attributes. Each daily occurrence of a trip has to be performed by one crew. In fact each crew performs a roster, defined as a sequence of trips whose operational cost and feasibility depend on several rules laid down by union contracts and company regulations. The problem consists of finding a set of rosters covering every trip of the given time period, so as to satisfy all the operational constraints with minimum cost.

Models for crew management and relevant references can be found in Caprara et al. [1997].

1.1.5 Block-to-Train Assignment Problem

Once the blocking plan is developed and train routings have been identified, the next step is to determine which trains should carry blocks to move shipment from their origins to destinations. At this point, blocks are assigned to the trains so that all blocks are shipped from their origins to their destinations and the total cost of transportation is minimum. A block may be carried by a number of trains as it travels from its origin to its destination. This problem is known as the block-to-train assignment problem and can also be formulated as a mixed integer programming problem. However, we are not aware of any effective exact or heuristic algorithm to solve the block-to-train assignment problem. At most railroad companies, this problem is solved either manually or using fairly elementary heuristic algorithms (Shan [1997]). As a result, there is substantial inefficiency in the system, blocks travel much longer distances than they should, and there is unnecessary block swapping between trains. There is a need for effective algorithms for solving the block-to train assignment problem.

1.1.6 Locomotive Scheduling Problem

Major U.S. railroads maintain large fleets of locomotives of various types with different pulling capacities. The locomotive scheduling problem is to assign locomotives to the trains to fulfill the motive power requirement of the trains in the train schedule. Usually the motive power requirement by a train is determined by the block-train assignment, and is expressed in terms of horsepower and tonnage. A set of operational constraints must also be satisfied for a practical locomotive assignment. In Chapter 2 we

give more details on this problem. Recent literature can be found in Ziarati et al. [1997] and Ziarati et al. [1999].

1.2 Airline Scheduling Problems

In the U.S., airlines are the main mode of passenger movement. Growing demand from customers, steep competition among airline carriers, high sensitivity/uncertainty with respect to sociological and political circumstances, and development in the IT industry have motivated the airline companies to adopt Operational Research (OR) tools to improve cost efficiency and customer satisfaction. Airline industries now regularly use optimization models in their planning and scheduling tasks and the use of these models has resulted in hundreds of millions of dollars in annual saving for major airlines (Yu [1998], Barnhart and Talluri [1997] and Ahuja et al. [2001c]).

In scheduled passenger air transportation, airline profitability is critically influenced by the airlines' ability to construct flight schedules containing flights at desirable times in markets (defined by origins and destinations pairs) with high demand. To construct such schedules, airlines engage in a complex decision making process referred to as airline schedule planning. Airline scheduling is one of the most important OR problems in the airline industry. This problem has drawn considerable attention from OR researchers and airline companies have adopted many of the OR techniques in practice.

We now briefly present here the concepts of demand for air travel, supply of air passenger services and their interaction to better understand the airline scheduling problems. The demand for air-travel in a market is derived from the need for individuals to travel by plane in that market. A market is defined by an origin and destination pair. Two markets are called opposite markets if demands in these two markets are not

interacting (for example, Atlanta – Miami and Miami –Atlanta) and are called parallel markets if the demands in the two markets interact (for example, Atlanta – SFO and Atlanta – Oakland). Several methods for demand forecasting are in use. Some of them are as follows: (i) stepwise forecasting (first forecast the overall travel demand in a region or zone, then derive the demand for travel for an origin-destination pair, then determine the air-travel demand for that origin-destination), (ii) demand forecasting based on the population of the cities and distance between cities, and (iii) regression analysis. An airline determines its demand for an origin-destination pair based on overall air travel demand for that origin-destination, market share of the airline, and capacity of the airline.

Once an airline determines the demand to be fulfilled, it either builds or modifies the flight network to compete for the market share. A flight network consists of nodes representing airports and arcs representing direct flights between two nodes. With respect to flight network, the basic airline scheduling problem reduces to deciding the network structure to be followed, flight arcs to be made in the network, and frequency of flights on each arc in network.

Generally, U.S. airlines adopt a *hub-and-spoke network* (other alternatives may be *linear network* or *complete network*), in which the hub airport is directly connected to each of the non-hub airports in the network. If there are n nodes in the flight network, the hub-and-spoke network requires $2(n-1)$ links. The reason for adopting the hub-and-spoke network lies in its ability to reduce the variability in the number of passengers in a flight leg (a non-stop flight from an origin to a destination with a specified departure time and an arrival time) by mixing and consolidating demands from different markets on each flight leg.

Once the flight network structure is determined, airlines prepare the schedules. Generally they adopt a sequential approach in which schedules are prepared in steps. We now briefly describe the steps involved in airline scheduling.

1.2.1 Flight Schedule Design Problem

The flight schedule design problem consists of deciding when and where flights should be flown. A schedule is fixed for a period of time, usually up to three months, with some minor alternations from month to month. The objective of the schedule design is to develop a schedule defining an origin, a destination, a departure time, and an arrival time for each service to accommodate passenger demands. The schedule should meet resource availability constraints. Constrained resources in airline scheduling include vehicles, crews, maintenance facilities, staff, etc. The schedule design problem can be further divided into two problems: (i) route generation – deciding which candidate flights should be considered; and (ii) route selection – selecting the most profitable flight legs out of all the candidate flight legs.

1.2.2 Fleet Assignment Problem

In this step, every flight in the schedule is assigned to an aircraft type, called fleet, subject to the availability of aircraft types and number of aircrafts available in each type. An airline's fleet is typically made up of a variety of equipment types such as F100 (Fokker), DC-9 (McDonnell Douglas), 757 (Boeing), etc. There are considerable differences in the capacities and operational characteristics of these aircraft types. An F100, for example, has a capacity of 98 in coach class, while a 757 can seat up to 185 in coach. Operational characteristics differ in speed, fuel burn rates, landing weight (airport fees are based on this), range, maintenance costs and minimum turnaround times. These operational differences mean that the cost of flying a flight leg depends on the equipment

type assigned to the leg. Therefore, the fleet assignment problem is to assign aircraft type for each flight leg so that airline can fly the schedule, already designed, while matching capacity to the demand as much as possible and minimizing the total operating cost of flying the schedule. Some of the very rigid constraints to be observed in fleet assignments are as follows: (i) the right aircraft should be present at the right place at the right time, (ii) no more aircraft of each type than present in a fleet, and (iii) every flight leg in the schedule has to be assigned exactly one aircraft type. Additional constraints considering maintenance requirements and noise and gate restrictions can also be included.

1.2.3 Through Assignment Problem

Once the aircraft types are assigned to each flight leg, decisions are made to connect flight legs with same the aircraft types at an airport. This ensures that the number of incoming aircrafts of a type at an airport is equal to the number of outgoing aircrafts of that type from the airport. A *through* connection is a connection between an inbound flight leg and an outbound flight leg at a station which ensures that the same plane flies both legs. Since a through connection allows passengers to remain onboard instead of changing gates at busy airports, passengers are willing to pay a premium for such connections; this premium is termed the *through benefit*. The through assignment problem takes as an input a list of candidate pairs of flight legs that can make through connections with corresponding through benefits, and identifies a set of most profitable through connections.

1.2.4 Aircraft Routing Problem

Once the schedule has been fleeted, the next step is to determine the aircraft rotations (a sequence of flights beginning and ending at the same location) for every plane subject to maintenance requirements. The aircraft routing problem (also known as

the aircraft maintenance routing problem) takes a fleeted schedule and the available number of aircraft for each fleet as input. In a traditional fleet assignment process, maintenance requirements are modeled only approximately by ensuring a sufficient number of maintenance opportunities for each fleet type. A maintenance opportunity exists when an aircraft overnights at one of its maintenance locations. While this ensures that, on average, enough aircraft of each type are in maintenance nightly, it does not guarantee that individual aircrafts are treated equally: one aircraft might have one maintenance opportunity per day while another might not have any in a week. The aircraft maintenance routing problem addresses this. It determines the actual rotation, or sequences of connected flights beginning and ending at the same location, of individual aircraft subject to maintenance rules imposed by both the regulatory agency and the airline itself. Oftentimes the airline's rules are more stringent than those of the regulatory agency, to avoid the expensive penalty associated with violating maintenance rules.

1.2.5 Crew Scheduling Problem

In this step, the planners allocate flight crews – pilots and flight attendants – to flight legs such that all work rules are satisfied, each flight leg has the necessary crew and crew costs are minimized. There are many restrictions to be met by the crew scheduling problem. Some of these are listed below.

- Pilots are qualified to fly only certain aircraft types;
- Work schedules must satisfy maximum time-away-from-base (the period that flight crews are away from their domicile stations) restrictions;
- Crews are not allowed to stay on duty longer than a maximum flying time;
- Work schedules must satisfy minimum rest time.

Typically the crew scheduling problem is broken into two steps: a crew pairing problem and a crew assignment problem. The objective of the crew pairing problem is to find a set of work schedules that covers each flight the appropriate number of times and minimize total crew costs. In crew assignment, these pairings are combined with rest periods, vacations, training times, etc. to create extended work schedules that can be performed by an individual. The objective of the crew assignment problem is to find a minimum cost assignment of employees to these work schedules.

The drawback of the stepwise scheduling process is that the overall schedule obtained by this procedure may not be best with respect to system perspective. However, integrated models encompassing all the steps are too complex. Recently, attempts have been made to solve integrated models consisting of two or more steps. Our models in Chapters 4 and 5 solve integrated scheduling problems.

1.3 Dissertation Summary and Outline

The focus of this dissertation is to study several real-life scheduling problems in railroad and airline transportation. Airlines have adopted many optimization models in their everyday scheduling, and our research tries to bring more integration in the optimization models that are currently solved sequentially in practice. On the other hand, there is not much use of optimization tools in railroad transportation. Our research tries to model and solve two scheduling problems that are of great importance to railroads with optimization techniques. The outline of this dissertation is as follows.

In Chapter 2, we develop an optimization model to solve the locomotive scheduling problem. Because the Mixed Integer Programming (MIP) formulation of this problem is too large for real life instances, we simplify it using a novel approach of solving first the daily locomotive schedule, then extending this solution to the weekly locomotive

schedule. One important feature of our model is that it integrates a set of operational constraint, which are critical for real-life locomotive scheduling. The computational testing of our algorithms on data provided by CSX Transportation gives savings of over 400 locomotives, which can be translated into hundreds of millions of dollars per year.

Chapter 3 describes the two approaches we have developed to solve the railroad blocking problem: (i) a Lagrangian relaxation based approach, and (ii) a Very Large Scale Neighborhood (VLSN) search based approach. In the first approach we formulate the blocking problem as an MIP and apply multi-level Lagrangian relaxation on the constraints. However, the computational testing of this approach has produced solutions not satisfactory in terms of quality of the solutions and computational time taken. Railroads are more interested in obtaining implementable near-optimal solutions in reasonable computational time, and hence we develop a VLSN search based approach for this problem, which produces very good solutions in reasonable time. The added advantage of this approach is that many additional constraints can be incorporated. Those constraints are necessary to obtain implementable blocking policies. The computational testing of our VLSN search algorithms on real-life data from CSX Transportation and BNSF Railway has been validated by these two railroads and proved capable of cutting their freight shipping cost substantially.

The integrated model for combined through and fleet assignment is called the combined through-fleet assignment model (ctFAM) (Ahuja et al. [2001c]). In Chapter 4, we incorporate ctFAM with two additional criteria: (i) ground manpower planning and (ii) crew planning in the objective function of ctFAM. The solution of the multi-criteria ctFAM is more conducive to crew planning and ground manpower planning, which are

subsequent scheduling problems of airline operations. The computational testing on the real-life data from United Airlines indicates the quality of the solutions obtained by our algorithms is very good with respect to ground manpower planning and crew planning.

In Chapter 5, we study a generalized approach of ctFAM with the additional feature of time windows (ctFAM-TW). In ctFAM-TW each flight leg has an associated time window with its departure time, so that it can depart flexibly at any time within the time window. It can be noted that ctFAM is a special case of ctFAM-TW with zero time window. This extended version of ctFAM allows more profitable through connection opportunities for flight legs. The computational testing of ctFAM-TW on real-life data from United Airlines gives revenue improvement of millions of dollars per year over the solution of ctFAM without time windows.

CHAPTER 2
SOLVING THE REAL-LIFE LOCOMOTIVE SCHEDULING PROBLEM

2.1 Introduction

Transportation is one of the most vital services in modern society. Transportation of goods by railroads is an integral part of the U.S. economy. Railroads play a leading role in multi-modal and container transportation. The rail transportation industry is very rich in terms of problems that can be modeled and solved using mathematical optimization techniques. However, research in railroad scheduling has experienced a slow growth and, until recently, most contributions used simplified models or had small instances failing to incorporate the characteristics of real-life applications. The strong competition facing rail carriers (most notably from trucking companies) and the ever increasing speed of computers have motivated the use of optimization models at various levels in railroad organizations. In addition, recently proposed models tend to exhibit an increased level of realism. As a result, there is growing interest for optimization techniques in railroad problems. In the last few years, a growing body of advances concerning several aspects of rail freight and passenger transportation has appeared in the operations research literature (see, for example, Jovanovic and Harker [1991], Brannlund et al. [1998], Newton et al. [1998], Cordeau et al. [1998], and Sherali and Suharko [1998]). This chapter concerns the development of new models and algorithms for solving real-life locomotive scheduling problems faced by U.S. railroad companies.

The locomotive scheduling problem (or the locomotive assignment problem) is to assign a consist (a set of locomotives) to each train in a pre-planned train schedule so as

to provide them sufficient power to pull them from their origins to their destinations. Locomotive scheduling problems are among the most important problems in railroad scheduling (Florian et al. [1976], Mao and Martland [1981], Smith and Sheffi [1988], Chih et al. [1990], Forbes et al. [1991], Fischetti and Toth [1997], Nou et al. [1997], and Ziarati et al. [1997, 1999]). Often, locomotive availability is the constraining factor in whether a train departs on time. With new locomotives costing in excess of $1.8 million, it is paramount to the railroad business that they be managed efficiently. The variety of types of locomotives, different types of trains, and the diverse geographic networks create very difficult combinatorial optimization problems. There are not satisfactory algorithms to solve these problems. As a result, there are inefficiencies in locomotive management, and substantial savings may be achieved through the development of better models and algorithms. CSX Transportation has over 3,000 locomotives, which translates into a capital investment of over $5 billion, and over $1 billion in yearly maintenance and operational costs. Our study reports an improvement of 5% in average locomotive utilization for CSX, which translates into a saving of over $100 million per year.

Locomotive scheduling problems can be studied at two levels - planning level or operational level. At the planning stage of the locomotive scheduling problem, we assign locomotive types to various trains. Typically, a railroad company has different type of locomotives with different pulling and cost characteristics. For example, we may assign two CW44AC and one CW40-8 locomotive to a train. But the railroad company may have several hundred locomotives of type CW44AC and CW40-8. Of these, which specific units get assigned to the train is handled by the operational locomotive scheduling problem. The operational locomotive scheduling model also takes into

account the fueling and maintenance needs of the locomotives, which are ignored in the planning model.

Railroad companies differ on their views of the relative importance of the planning problem versus the operational problem. One view holds that the day-to-day variability in traffic patterns, locomotive unreliability, changing service priorities and the wide range of train schedule operations create an environment that is so unpredictable that it is a futile exercise to develop a static locomotive scheduling plan. This philosophy, which we refer to as the tactical planning philosophy, would attempt to take into account current conditions on an on-going basis and dynamically solve the locomotive scheduling problem. This creates an ever changing set of solutions where particular trains may run with very different consists each day, and each train may source those locomotives from a variety of inbound trains, depending on that days' situation.

These philosophy differences carry over to many aspects of the business. Ultimately, they can be summarized in two camps: those who believe it is best to operate a scheduled railroad and those who believe it is best to operate a "tonnage" or tactical railroad. The first camp would hold that while running a specific schedule the same way every time may in fact result in local inefficiencies (such as trains with very few cars or terminals that may be over staffed on some shifts), the macro level total cost of the operation is minimized. The second camp would hold that the total cost can be reduced by tactically adjusting the operation to reduce the number of local inefficiencies. These strategic approaches are linked to the view of the role of line managers as well. The operational model based railroad requires managers to be flexible and adjust their local operations to an ever changing pattern of trains, cars, locomotives and crews. The local

managers also are expected to make quick, tactical decisions to adjust the operations to reduce inefficiencies both at the local and the network level. The planning model based railroad assumes that managers will anchor their local operation to the foundation of regular, routine, repeatable network operations. Over time, they will make finite adjustments to their many sub-processes that optimize their responsibility area. This fine tuning of the operation is not possible in a tactical operation due to the lack of stability.

These strategic approaches are linked to the view of the role of line managers as well. The operational model based railroad requires managers to be flexible and adjust their local operations to an ever changing pattern of trains, cars, locomotives and crews. The local managers also are expected to make quick, tactical decisions to adjust the operations to reduce inefficiencies both at the local and the network level. The planning model based railroad assumes that managers will anchor their local operation to the foundation of regular, routine, repeatable network operations. Over time, they will make small adjustments to their many sub-processes that optimize their responsibility area. This fine tuning of the operation is not possible in a tactical operation due to the lack of stability.

The CSX managers who sponsored this research chose to emphasize the planning part of the locomotive assignment problem for several reasons. First of all, they believe that the planning problem is of value in running the railroad. The CSX management philosophy is to operate a scheduled railroad and believes the inherent value of a repeatable, routine, scheduled set of decisions will ultimately not only minimize total locomotive costs but also total operating costs while improving the service product. Second, the planning system could be used as part of a network planning tool to evaluate

which engines to buy in the future, and to study the impact of modifying their schedule. Third, any technology developed for the planning problem could possibly be extended to deal with tactical decisions in an operational setting.

In this chapter, we consider the planning version of the locomotive scheduling model (LSM). We will now summarize the features of the LSM to give the reader a better understanding of the problem. A large railroad company has a train schedule which consists of several hundred trains with different weekly frequencies. Some trains run each day in a week, whereas others run less frequently. At CSX, there are several thousand train departures per week (assuming that we may count the same train running on different days multiple times). Many trains have long hauls and take several days to go from their origins to their destinations. To power these trains, CSX has several thousand locomotives of different types. Some locomotives are AC powered, some DC powered; they have different manufacturers, and some are more powerful than others. In LSM, we assign a set of locomotives to each train in the weekly train schedule so that each train gets sufficient tractive effort (that is, gets sufficient pulling power) and sufficient horsepower (that is, gets sufficient speed), and the assignment can be repeated indefinitely week to week. At the same time, assigning a single locomotive to a train is undesirable because if that locomotive breaks down, the train gets stranded on the track and blocks the movement of other trains.

An additional feature of the LSM is that some locomotives may be *deadheaded* on trains. Deadheaded locomotives do not pull the train; they are just pulled by active locomotives from one place to another place. Deadheading plays an important role in locomotive scheduling models since it allows extra locomotives to be moved from the

places where they are in surplus to the places where are in short supply. Locomotives also *light travel*; that is, they travel on their own between different stations to reposition themselves between two successive assignments to trains. A set of locomotives in light travel forms a group, and one locomotive in the group pulls the others from an origin station to a destination station. Light travel is different from deadheading of locomotives since it is not limited by the train schedule. In general, light travel is faster than deadheading. However, light travel is more costly as a crew is required to operate the pulling locomotive, and the transportation does not generate any revenue as there are no cars attached.

Since we assign a set of locomotives (or a *consist*) to trains, we need to account for *consist-busting*. Whenever a train arrives at its destination, its consist is either assigned to an outbound train in its entirety, or its consist goes to the pool of locomotives where new consists are formed. In the former case, we say that there is a *train-to-train connection* between the inbound and outbound trains and no *consist-busting* takes place. In the latter case we say that consist-busting takes place. Consist-busting leads to mixing of locomotives from inbound trains and regrouping them to make new consists. This is undesirable from several angles. First, consist-busting requires additional locomotive time and crew time to execute the moves. Second, consist-busting often results in outbound trains getting their locomotives from several inbound trains. If any of these inbound trains is delayed, the outbound train is also delayed, which potentially propagates to further delays down the line. In an ideal schedule, we try to maximize the train-to-train connections of locomotives and thus minimize consist-bustings. A major contribution of our research is to explicitly model the economic impacts of consist-

busting, and to reduce its impacts on the system. Moreover, a schedule that performs a lot of consist-busting may be too complex to be implementable in practice.

Another important feature of the locomotive scheduling model is that we want a solution that is *consistent* throughout the week in terms of the locomotive assignment and train-to-train connections. If a train runs five days a week, we want it to be assigned the same consist each day it runs. If we make a train-to-train connection between two trains and if on three days in a week both trains run, then we want them to have the same train-to-train connection on all three days. Consistency of the locomotive assignment and train-to-train connections is highly desirable from an operational point of view. Another contribution of our research is that we model the effects of inconsistency, and try to reduce the inconsistency in a schedule.

This chapter reports the development of a locomotive scheduling model that models the assignment of active and deadheaded locomotives to trains, light traveling of locomotives, consistency and consist-busting decisions in an integrated model. The objective in the model is to minimize the total cost, which is the sum of the active locomotive costs, deadheading costs, light travel costs, consist-busting costs, locomotive usage costs, and the penalty for using single locomotive consists. The solution is required to provide sufficient power to every train in a timely fashion to meet their prescribed schedules. We first describe a mixed integer programming formulation. Unfortunately, this MIP formulation is too large to be solved to optimality or near-optimality. We solve this model heuristically using a combination of techniques taken from linear programming, integer programming, and neighborhood search algorithms.

Locomotive scheduling problems are similar to the airline scheduling problems where one assigns planes of different types to flight legs. The planning version of the locomotive scheduling problem studied in this paper is similar to the well known fleet assignment model (see, for example, Subramanium et al. [1994] and Hane et al. [1995]), and the operational version of the locomotive scheduling problem is similar to the aircraft routing problem (see, for example, Barnhart et al. [1998]). However, the locomotive scheduling problem considered in this paper is substantially more difficult than the fleet assignment model (FAM). In the FAM, we assign a single plane to a flight leg, but in LSM we assign a set of locomotives to a train. In FAM, there is no deadheading, no light travel, and no consist-busting. Further, the popular and well-solved FAM models assume that the flight leg schedule is the same each day of the week; that is, they solve the daily scheduling problem and repeat it each day of the week. But in the locomotive scheduling problem, we need to consider the weekly schedule of trains, and train schedules do not repeat on a daily basis. Hence, the LSM is combinatorially much harder to solve than the FAM.

The locomotive division at CSX Transportation has developed in-house software for the LSM which was developed, refined, and improved over a period of 10 years. This software has two main parts: *consist-builder* and *locomotive scheduler*. The consist-builder assigns active consists to trains; that is, which set of locomotives will actively pull the trains. This assignment is done based on train types, geography and additional business rules. The locomotive scheduler then routes locomotives in the weekly train network so that each train gets the desired active consist. In order to provide the desired active consists to trains, deadheading of the locomotives is often necessary. The

locomotive scheduler considers each locomotive type one by one and solves a minimum cost flow problem to determine locomotive flow in the network. Breaking the LSM into two distinct parts leads to inefficiencies. When consist-builder assigns active consists, it does not take into account how locomotives will flow in the network to provide those consists to trains. When locomotive flow is determined, then consists cannot be changed. Another source of inefficiency in their approach is that the locomotive scheduler solved the problem one locomotive type (or, one commodity) at a time. In addition, this approach did not handle light travel of locomotives (which was done manually) and was unable to model consist-bustings. The planning solution provided by their software had very high consist-bustings. As many as 85% of the trains had their consists busted. Our model integrates both parts of their system into a single model, treats all locomotive types simultaneously, and models the impacts of consist-bustings, light travel, and deadheading.

Our LSM model is substantially different than locomotive scheduling models studied previously by researchers. Single locomotive models have been studied by Forbes et al. [1991], and Fischetti and Toth [1997]. Multicommodity flow based models for planning decisions have been studied by Florian et al. [1976], Smith and Sheffi [1988], and Nou et al. [1997]. Multicommodity flow based models for operational decisions have been developed by Chih et al. [1990], and Ziarati et al. [1997, 1999]. Our multicommodity flow based model for planning decision has more features than any of the existing planning models.

The locomotive scheduling problem is a very large-scale combinatorial optimization problem. We formulate it as a mixed integer programming (MIP) problem,

which is essentially an integer multicommodity flow problem with side constraints. The underlying flow network is the weekly space-time network where arcs denote trains, nodes denote events (that is, arrival and departure of trains), and different locomotive types define different commodities. Since we assign only integer number of locomotives to trains, we get integer multicommodity flow problems. The constraints that the locomotives assigned to a train must provide sufficient tonnage and horsepower and that the number of locomotives of each type is in limited quantity gives rise to the side constraints. In addition, our formulation has fixed charge variables which result from modeling the light travel and consist-bustings. Even when we ignore the consistency constraints, our formulation contains about 197 thousand integer variables and 67 thousand constraints, and is too large to be solved to optimality or near-optimality using existing commercial-level MIP software.

We developed a methodology to solve the locomotive scheduling problem heuristically. By using linear programming, mixed integer programming, and very large-scale neighborhood search techniques, we have attempted to obtain very good solutions of the locomotive scheduling problem. We solve the LSM in two stages. In the first stage, we modify the original problem so that all trains run seven days a week. This approximation of the original problem allows us to handle consistency constraints satisfactorily. In the second stage, we modify the solution of the first stage to solve the original problem where trains do not run all seven days a week. The first stage problem, though substantially smaller than the original problem, is still too large to be solved by the existing MIP software. The source of difficulty is the fixed charge variables introduced by modeling the light travel and consist-bustings. We developed linear

programming based greedy algorithms to determine the values of these variables. Once these variables were determined, the resulting MIP gave very good solutions quickly, usually within 10 minutes.

We developed prototype software for our algorithms and compared our software with the in-house software used by CSX. On one benchmark instance, for which CSX software required 1,614 locomotives to satisfy the train schedule, our software required only 1,210 locomotives, thereby achieving a savings of over 400 locomotives. We obtained similar improvements on other benchmark instances. In the solutions obtained by the CSX software, locomotives actively pull a train about 31.3% of the time, deadhead about 19.6% of the time, and idle at stations about 49.1% of the time. In the solutions obtained by our software, locomotives actively pull the train about 44.4% of the time, deadhead about 8.1% of the time, light travel about 0.8% of the time, and idle at stations about 46.7% of the time.

This chapter is organized as follows. In Section 2.2, we describe the problem in greater detail and define our notation. In Section 2.3, we describe the space-time network which will be the basis of all of our formulations. Section 2.4 describes the MIP formulation of the problem. In Section 2.5, we show the motivation for solving the problem in two stages, first as a daily scheduling problem followed by the weekly scheduling problem. Section 2.6 describes how we solve the daily scheduling problem, and Section 2.7 the weekly scheduling problem. In Section 2.8, we present a summary of our algorithmic approach. Section 2.9 presents the computational results of our approach and compares with the approach used at the CSX Transportation. Finally, Section 2.10 summarizes our contributions and outlines the future research issues.

2.2 Problem Details

In this section, we give the details and notation of the locomotive scheduling problem used for planning at CSX.

Train Data:

Locomotives pull a set *L* of trains from their origins to destinations. The train schedule is assumed to repeat from week to week. Trains have different weekly frequencies; some trains run every day, while others run less frequently. We will consider the same train running on different days as different trains; that is, if a train runs five days a week, we will consider it as five different trains for which all data is the same except that they will have different departure and arrival times.

We use the index *l* to denote a specific train. For the planning model, the train schedule is deterministic and pre-specified. There are three classes of trains: *Auto*, *Merchandize*, and *Intermodal.* Each train belongs to exactly one class. The required tonnage and horsepower is specified. The tonnage of a train represents the minimum pulling power needed to pull the train. The tonnage depends upon the number of cars pulled by the train, weight of the cars, and the slope or ruling grade of that train's route. The horsepower required by the train is its tonnage multiplied by the factor that we call the *horsepower per tonnage*. The greater the horsepower per tonnage, the faster the train can move. Different classes of trains have different horsepower per tonnage. For greater model flexibility, we allow each train to have its own horsepower per tonnage. We associate the following data with each train *l.*

- *dep-time*(*l*) : The departure time for the train *l*. We express this time in terms of the weekly time as the number of minutes past Sunday midnight. For example, if the train *l* leaves on Monday 6 AM, then *dep-time*(*l*) = 360; and if it leaves on Tuesday 6 AM, then *dep-time*(l) = 1,800.

- *arr-time*(l) : The arrival time for train l (in the same format as the *dep-time*(l)).
- *dep-station*(l) : The departure station for train l.
- *arr-station*(l) : The arrival station for train l.
- T_l : Tonnage requirement of train l.
- β_l : Horsepower per tonnage for train l.
- H_l : Horsepower requirement of train l, which is defined as $H_l = \beta_l T_l$.
- E_l : The penalty for using a single locomotive consist for train l.

Locomotive Data:

A railroad company typically has several different types of locomotives with different pulling and cost characteristics and different number of axles (often varing from 4 to 9). Locomotives with different characteristics allow railroads greater flexibility in locomotive assignments, but also make the locomotive scheduling problem substantially more difficult. We denote by K the set of all locomotive types, and use the index k to represent a particular locomotive type. We associate the following data with each train $k \in K$:

- h^k : Horsepower provided by a locomotive of type k.
- α^k : Number of axles in a locomotive of type k.
- G^k : Weekly ownership cost for a locomotive of type k.
- B^k : Fleet size of locomotives of type k, that is, the number of locomotives available for assignment.

Active and Deadheaded Locomotives:

Locomotives assigned to a train either actively pull the train or deadhead. Deadheading allows extra locomotives to be moved from places where they are in surplus to the places where they are in short supply. For example, more tonnage leaves a coal

mine than arrives at the coal mine; so more pulling power is needed on trains departing from the mine. This creates a demand for locomotives at the mine. Similarly, more tonnage arrives at a thermal power plant than leaves it; so more pulling power is needed on trains arriving at the power plant. This creates a surplus of locomotives at the power plant. Effective deadheading of locomotives reduces the total number of locomotives used and improves the average locomotive utilization. We need the following data for train-locomotive type combinations:

- c_l^k : The cost incurred in assigning an active locomotive of type *k* to train *l*.
- d_l^k : The cost incurred in assigning a deadheaded locomotive of type *k* to train *l*.
- t_l^k : The tonnage pulling capability provided by an active locomotive of type *k* to train *l*.

The active cost c_l^k captures the economic asset cost of the locomotive for the duration of the train and the fuel and maintenance costs. The deadhead cost d_l^k captures the same asset cost, a reduced maintenance cost, and zero fuel cost. Observe that the tonnage provided by a locomotive depends upon the train. Different train routes have different ruling grades (that is, slopes) and the pulling power provided by a locomotive type is affected by the ruling grade.

Also specified for each train *l* are three disjoint sets of locomotive types: (i) *MostPreferredl*, the preferred classes of locomotives; (ii) *LessPreferredl*: the acceptable (but not preferred) classes of locomotives; and (iii) *Prohibitedl*, the prohibited classes of locomotives. CSX uses business rules based on train types and geographical considerations to determine these classes for each train. When assigning locomotives to a

train, we can only assign locomotives from the classes listed as *MostPreferredl* and *LessPreferredl (*a penalty is associated for using *LessPreferredl).*

Light Travel**:**

Our model allows light travel of locomotives, that is, locomotives traveling in a group on their own between different stations to reposition themselves. Similar to deadheading, light travel can be an effective way to reposition locomotives. The light travel cost has a fixed component that depends upon the distance of travel in the light move since we need a crew, and a variable component that depends upon the number of locomotives light traveling.

Consist-Busting**:**

Consist-busting is a normal phenomenon in railroads because the needs for outgoing locomotives at a station do not precisely match the incoming needs. However, consist-busting incurs a cost in complexity of managing the system, and in delays in repositioning locomotives. Consist-busting can be reduced by a better scheduling of locomotives. We model the cost of consist-busting with a fixed component, B, per consist-busting and a variable component that depends upon the number of locomotives involved in the consist-busting.

We will now describe the constraints in the LSM. The constraints can be classified into two parts: *hard constraints* (which each locomotive assignment must satisfy) and *soft constraints* (which are desirable but not always required to be satisfied). We incorporate soft constraints by attaching a penalty for each violation of these constraints.

Hard Constraints**:**

- Power requirement of trains: Each train must be assigned locomotives with at least the required tonnage and horsepower.

- Locomotive type constraints: Each train *l* is assigned locomotive types belonging to the set *MostPreferredl* and *LessPreferredl* only.

- Locomotive balance constraints: The number of incoming locomotives of each type into a station at a given time must equal the number of outgoing locomotives of that type at that station at that time.

- Active axles constraints: Each train must be assigned locomotives with at most 24 active axles. This business rule is designed to protect the standard couplers used in North America. Exceeding 24 powered axles may result in overstressing the couplers and causing a train separation.

- Consist size constraints: Each train can be assigned at most 12 locomotives including both the active and deadheaded locomotives. This rule is a business policy of CSX that reduces its risk exposure if the train were to suffer a catastrophic derailment.

- Fleet size constraints: The number of assigned locomotives of each type is at most the number of available locomotives of that type.

- Repeatability of the schedule: The number and type of locomotives positioned at each station at the beginning of the week must equal the number and type of locomotives positioned at that station at the end of the week. This ensures that the assignment of locomotives can be repeated from week to week.

Soft Constraints:

- Consistency in locomotive assignment: If a train runs five days a week, then it should be assigned the same consist each day it runs. CSX believes that crews will perform more efficiently and more safely if they operate the same equipment on a particular route and train. As the crews learn the operating nuances associated with each combination, they will adjust their throttle and braking control accordingly.

- Consistency in train-to-train connections: If locomotives carrying a train to its destination station connect to another train originating at that station, then it should preferably make the same connection on each day both the trains run. This is useful to help terminal managers optimize their sub-processes associated with arriving and departing trains.

- Same class connections: Trains should connect to other trains in the same class, e.g., auto trains should connect to auto trains; merchandise trains should connect to merchandise trains, etc. This is useful in that different trains have different preferred types of locomotives and may originate and terminate at different locations within a larger terminal area (an unloading ramp, for example).

- Avoid consist-busting: Consist-busting should be avoided as much as possible.

Objective Function:

The objective function for the locomotive scheduling model contains the following terms:

- Cost of ownership, maintenance, and fueling of locomotives
- Cost of active and deadheaded locomotives
- Cost of light traveling locomotives
- Penalty for consist-busting
- Penalty for inconsistency in locomotive assignment and train-to-train connections
- Penalty for using single locomotive consists

2.3 Space-Time Network

We will formulate the locomotive scheduling problem as a multicommodity flow problem with side constraints on a network, which we call the *weekly space-time network*. Each locomotive type defines a commodity in the network. We denote the space-time network as $G^7 = (N^7, A^7)$, where N^7 denotes the node set and A^7 denotes the arc set. We construct the weekly space-time network as follows. We create a *train arc* (l', l'') for each train l; the tail node l' of the arc denotes the event for the departure of train l at *dep-station*(l) and is called a *departure node*. The head node l'' denotes the arrival event of train l at *arr-station*(l) and is called an *arrival node*. Each arrival or departure node has two attributes: place and time. For example, *place*(l') = *dep-station*(l) and *time*(l') = *dep-time*(l). Similarly, *place*(l'') = *arr-station*(l) and *time*(l'') = *arr-time*(l). Some trains are called *forward train*s and some trains are called *backward trains*. *Forward trains* are those trains for which *dep-time*(l) > *arr-time*(l) and *backward trains* are those trains for which *dep-time*(l) < *arr-time*(l). For example, a train that leaves on Monday and arrives at its destination on Tuesday is a forward train; whereas a train that

leaves on Saturday and arrives on Monday is a backward train. (Recall that our timeline begins on Sunday at midnight.)

To allow the flow of locomotives from an inbound train to an outbound train, we introduce *ground nodes* and *connection arcs*. For each arrival node, we create a corresponding *arrival-ground node* with the same place and time attribute as that of the arrival event. Similarly, for each departure event, we create a *departure-ground node* with the same place and time attribute as that of the departure node. We connect each arrival node to the associated arrival-ground node by a directed arc called the *arrival-ground connection arc*. We connect each departure-ground node to the associated departure node through a directed arc called the *ground-departure connection arc*. We next sort all the ground nodes at each station in the chronological order of their time attributes, and connect each ground node to the next ground node in this order through directed arcs called *ground arcs*. (We assume without any loss of generality that ground nodes at each station have distinct time attributes.) The ground nodes at a station represent the pool (or storage) of locomotives at the station at different instants of times, when events take place. As trains arrive, they bring in locomotives to the pool through arrival-ground connection arcs. As train departs, they take out locomotives from the pool through ground-departure connection arcs. The ground arcs allow inbound locomotives to stay in the pool as they wait to be connected to the outbound trains. We also connect the last ground node in the week at a station to the first ground node of the week at that station through the ground arc; this ground arc models the ending inventory of locomotives for a week becoming the starting inventory for the following week.

We also model the possibility of an inbound train sending its entire consist to an outbound train. We capture this possibility by creating *train-train connection arcs* from an arrival node to those departure nodes whenever such a connection can be feasibly made. Railroads have some business rules about which train-train connections can be feasibly made. The arrival nodes i'' for a train i can be connected to a departure node j' for train j provided $\textit{min-connection-time} \leq \textit{dep-time(j)} - \textit{arr-time(i)} \leq \textit{max-connection-time}$, where *min-connection-time* and *max-connection-time* are two specified parameters. For example, *min-connection-time* = 120 (in minutes) and *max-connection-time* = 480.

We also allow the possibility of *light travel*, that is, several locomotives forming a group and traveling on their own as a group from one station to another station. Using a method described later in Section 2.6.4, we create possibilities for light travel among different stations. We create a *light arc* in the weekly space-time network corresponding to each light travel possibility. Each light arc originates at a ground node (with a specific time and at a specific station) and also terminates at a ground node. Each light arc has a fixed charge which denotes the fixed cost of sending a single locomotive with crew from the origin of the light arc to its destination. We denote this fixed charge for a light travel arc l by F_l. The light arc also has a variable cost which depends upon the number of locomotives light traveling as a group.

To summarize, the weekly space-time network $G = (N, A)$ has three types of nodes - arrival nodes (*ArrNodes*), departure nodes (*DepNodes*), and ground nodes (*GrNodes*); and four kinds of arcs - train arcs (*TrArcs*), connection arcs (*CoArcs*), ground arcs (*GrArcs*) and light travel arcs (*LiArcs*). Let $\textit{AllNodes} = \textit{ArrNodes} \cup \textit{DepNodes} \cup \textit{GrNodes}$, and $\textit{AllArcs} = \textit{TrArcs} \cup \textit{CoArcs} \cup \textit{GrArcs} \cup \textit{LiArcs}$. We show in Figure 2-1, a

part of the weekly space-time network at a particular station, which illustrates various kinds of arcs.

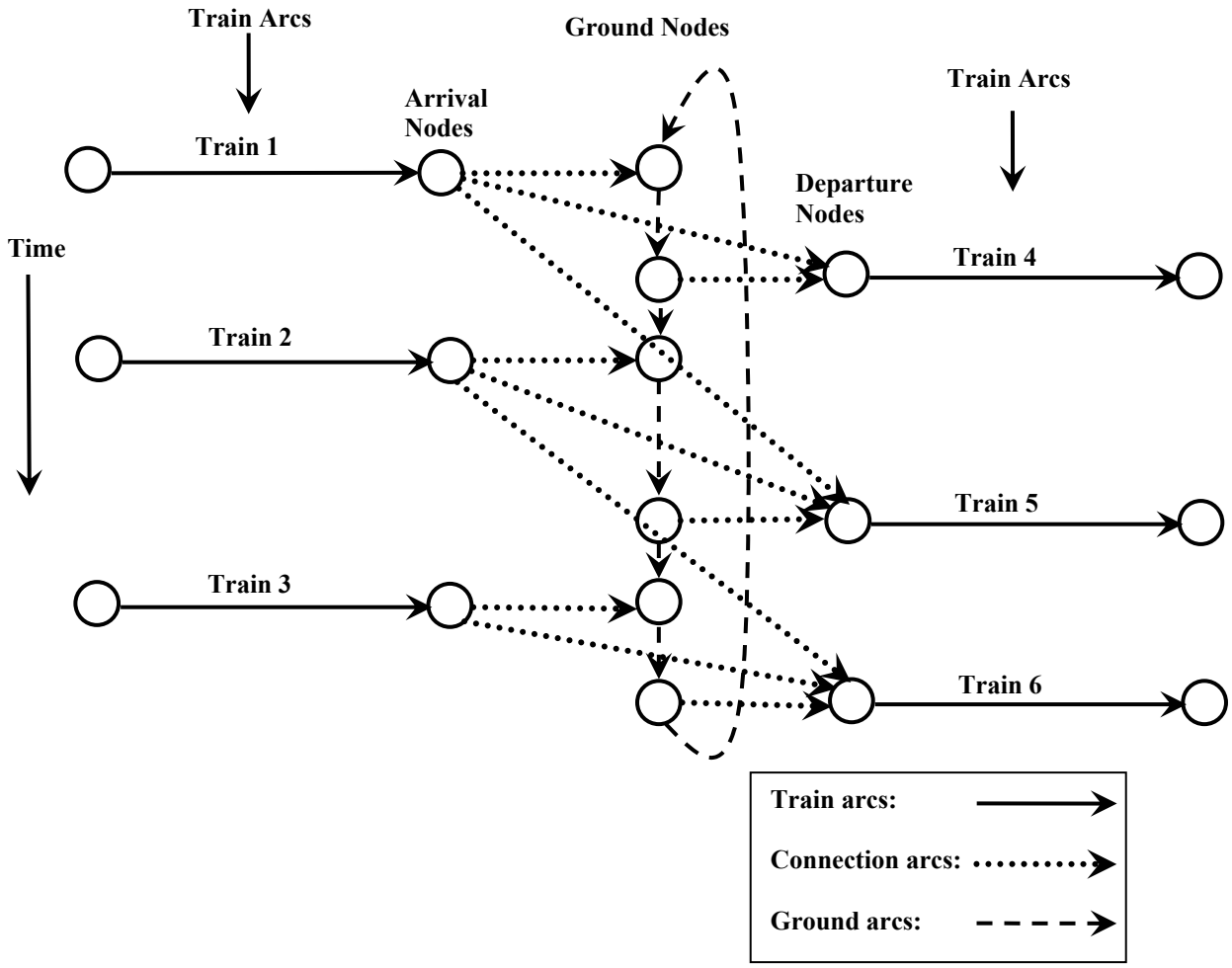


Figure 2-1. A part of the weekly space time network

We will formulate the locomotive scheduling problem as a flow of different types of locomotives in the weekly space-time network. Locomotives flowing on train arcs either are *active* or *deadheading*, those flowing on light arcs are *light traveling*, and those flowing on connection arcs are *idling* (that is, waiting between two consecutive assignments). We shall use the following additional notation for the weekly space-time networks in our MIP formulations:

- $I[i]$: Set of incoming arcs into node $i \in AllNodes$.

- $O[i]$: Set of arcs emanating from node $i \in AllNodes$.

- $tail(l)$: The tail node of arc $l \in AllArcs$.

- $head(l)$: The head node of arc $l \in AllArcs$.

- F_l: This cost term denotes the cost of an (active) light traveling locomotive with crew on a light arc $l \in LiArcs$. This cost includes the crew cost as well. If several locomotives light travel on an arc, then only one locomotive is active while others deadhead. For simplicity, we assume that the cost term is independent of the locomotive type pulling the consist.

- d_l^k: Recall that in Section 2.2, we defined d_l^k as the cost of deadheading of locomotive type k on train arc *l*. We now define it for every arc $l \in AllArcs$. For a light arc $l \in LiArcs$ d_l^k captures the cost of traveling for a non-active locomotive of locomotive type *k* on arc *l*. For a connection arc $l \in CoArcs \cup GrArcs$, d_l^k captures the cost of idling for locomotive type *k* on arc *l*.

- *CB*: The set of all connection arcs from arrival nodes to ground nodes. These are the arcs on which positive flow represents consist-busting. Alternatively, $CB = \{(i, j) \in AllArcs: i \in ArrNodes \text{ and } j \in GrNodes\}$.

- $CheckTime$: It is a time instant of the week when no event takes place; that is, no train arrives or departs at any station. We will henceforth assume that *CheckTime* is Sunday midnight.

- *S*: The set of arcs that cross the *CheckTime*; that is, $S = \{(i, j) \in AllArcs: time(i) < CheckTime < time(j)\}$.

2.4 The Mixed Integer Programming Formulation

In this section, we present the mixed integer programming (MIP) formulation of the locomotive scheduling model.

Decision Variables:

- x_l^k: Integer variable representing the number of active locomotives of type $k \in K$ on the arc $l \in TrArcs$;

- y_l^k: Integer variable indicating the number of non-active locomotives (deadheading, light-traveling or idling) of type $k \in K$ on the arc $l \in AllArcs$;

- z_l : Binary variable which takes value 1 if at least one locomotive flows on the arc $l \in LiArcs \cup CoArcs$, and 0 otherwise;
- w_l : Binary variable which takes value 1 if there is a flow of a single locomotive on arc $l \in TrArcs,$ and 0 otherwise;
- s^k : Integer variable indicating the number of unused locomotives of type $k \in K$.

Objective Function:

$$\min z = \sum_{l \in TrArcs} \sum_{k \in K} c_l^k x_l^k + \sum_{l \in AllArcs} \sum_{k \in K} d_l^k y_l^k + \sum_{l \in LiArcs} F_l z_l + \sum_{l \in CB} B z_l + \sum_{l \in TrArcs} E_l w_l - \sum_{k \in K} G^k s^k \tag{2.1a}$$

Constraints:

$$\sum_{k \in K} t_l^k x_l^k \geq T_l, \quad \text{for all } l \in TrArcs, \tag{2.1b}$$

$$\sum_{k \in K} h^k x_l^k \geq \beta_l T_l, \quad \text{for all } l \in TrArcs, \tag{2.1c}$$

$$\sum_{k \in K} a^k x_l^k \leq 24, \quad \text{for all } l \in TrArcs, \tag{2.1d}$$

$$\sum_{k \in K} (x_l^k + y_l^k) \leq 12, \quad \text{for all } l \in TrArcs, \tag{2.1e}$$

$$\sum_{l \in I[i]} (x_l^k + y_l^k) = \sum_{l \in O[i]} (x_l^k + y_l^k), \quad \text{for all } i \in AllNodes, \text{ for all } k \in K, \tag{2.1f}$$

$$\sum_{k \in K} y_l^k \leq 12 z_l, \quad \text{for all } l \in CoArcs \cup LiArcs, \tag{2.1g}$$

$$\sum_{l \in O[i]} z_l = 1, \quad \text{for all } i \in ArrNodes, \tag{2.1h}$$

$$\sum_{l \in I[i]} z_l = 1, \quad \text{for all } i \in DepNodes, \tag{2.1i}$$

$$\sum_{k \in K} (x_l^k + y_l^k) + w_l \geq 2, \quad \text{for all } l \in TrArcs, \tag{2.1j}$$

$$\sum_{l \in S} (x_l^k + y_l^k) + s^k = B^k, \quad \text{for all } k \in K, \tag{2.1k}$$

$$x_l^k,\ y_l^k \geq 0 \text{ and integer}, \quad \text{for all } l \in TrArcs, \text{ for all } k \in K, \tag{2.1l}$$

$$z_l \in \{0, 1\}, \quad \text{for all } l \in CoArcs \cup LiArcs, \tag{2.1m}$$

$$w_l \in \{0, 1\}, \quad \text{for all } l \in TrArcs. \tag{2.1n}$$

We now give some explanation for the above formulation to show that it correctly represents the LSM. We first discuss the constraints. The constraint (2.1b) ensures that the locomotives assigned to a train provide the required tonnage, and the constraint (2.1c) ensures that the locomotives assigned provide the required horsepower. The constraint (2.1d) models the constraint that the number of active axles assigned to a train does not exceed 24. The constraint (2.1e) models the constraint that every train is assigned at most 12 locomotives. The flow balance constraints (2.1f) ensures that the number of incoming locomotives equal the number of outgoing locomotives at every node of the weekly space-time network. The constraint (2.1g) makes the fixed charge variable z_l equal to 1 whenever a positive flow takes place on a connection arc or a light arc; this constraint also ensures that no more than 12 locomotives flow on any light arc. The constraint (2.1h) states that, for each inbound train, all the inbound locomotives use only one connection arc; either all the locomotives go to the associated ground node (in which case consist-busting takes place) or all the locomotives go to another outbound train (in which case consist-busting does not take place and there is a train-to-train connection). The constraint (2.1i) states that for each outbound train all the outbound locomotives either come from a ground node or all the locomotives come from an incoming train. The constraint (2.1j) makes the variable w_l equal to 1 whenever a single locomotive consist is assigned to train *l*. Finally, the constraint (2.1k) counts the total number of locomotives used in the week; which is the sum of the flow of locomotives on all the arcs crossing the

CheckTime. The difference between the number of locomotives available minus the number of locomotives used gives the number of locomotives saved (s^k).

We now discuss the objective function (2.1a) which contains six terms. The first term denotes the cost of actively pulling locomotives on train arcs. The second term captures the cost of deadheading locomotives on train and light travel arcs, and the cost of idling locomotives. We also include the variable cost of consist-busting in the definition of the term d_l^k for each arc $l \in CB$. The third term, $\sum_{l \in LiArcs} F_l z_l$ denotes the fixed cost of light traveling locomotives. The fourth term, $\sum_{l \in CB} Bz_l$, denotes the fixed cost of consist-busting. The fifth term, $\Sigma_{l \in TrArcs} E_l w_l$, denotes the penalty associated with the single locomotive consists; and the sixth term, $\Sigma_{k \in K} G^k s^k$, represents the savings accrued from not using all the locomotives.

Observe that the formulation (2.1) assumes that any locomotive type can flow on any train arc. But recall from our discussion in Section 2 that each train arc *l* has *MostPreferredl*, *LessPreferredl*, and *Prohibitedl* sets of locomotives. We handle these constraints in the following manner. To the formulation (2.1) we add the constraints $x_l^k = 0$ for each $k \in$ *Prohibitedl* and each $l \in$ *TrArcs*. This constraint ensures that prohibited locomotives are never used on train arcs. To discourage the flow of locomotive types belonging to the *LessPreferred* sets, we multiply c_l^k by a suitable parameter larger than (for example, 1.2) for each $k \in$ *LessPreferredl* and each $l \in$ *TrArcs.*

Our formulation (2.1) incorporates all the hard constraints described in Section 2, but not all the soft constraints. The formulation does not incorporate the consistency constraints. Including those constraints would make the formulation unmanageably large.

We, however, do handle the same class constraints implicitly in the definition of the term d_l^k. If a train-train connection arc l is not the same class connection arc, we multiply its d_l^k value by a constant parameter greater than 1, which penalizes the use of these arcs. By suitably selecting the value of this parameter, we can satisfy the same class constraints to varying degrees. We can also obtain different levels of consist-busting by selecting different values of the consist-busting cost B. The greater the value of B we choose, the less will be the amount of consist-busting in an optimal solution.

In the data provided to us by CSX, there were 538 trains, each of which operate several days in a week, and 5 locomotive types. The weekly space-time network consisted of 8,798 nodes and 30,134 arcs. The MIP formulation (2.1) consisted of 197,424 variables and 67,414 constraints. This formulation could not be solved to optimality or near-optimality using the state of the art commercial software. As a matter of fact, we were unable to solve even the linear programming (LP) relaxation of the problem. When we solved the LP relaxation of a much smaller problem, we found that the LP solution was highly fractional. Most of the variables were non-integer and the integer programming algorithm did not find a good integer solution in several hours. In addition, our MIP formulation (2.1) omits the consistency constraints that the same train running on different days should have the same locomotive assignment. In our formulation, a train running on different days is treated as separate trains and their locomotive assignments can be quite different. In principle, we can add new constraints to enforce the consistency constraints, but that would be impractical as it would increase the size of the MIP substantially. Hence, satisfying the consistency constraints by introducing new constraints did not seem to be a feasible option. Rather we developed an

alternative approach that simultaneously helped enforce consistency while dramatically reducing the problem size. We describe our approach in the next section.

2.5 Simplifying the Model

We next analyzed the number of trains running with different frequencies in a week for CSX. The Table 2-1 shown gives these numbers. The first column in the table gives the train frequency in a week; that is, how often the trains run in a week. The second column in the table gives the number of trains in the train schedule with the frequency given in the first column. For example, in the train schedule, there are 372 trains that run all seven days a week, there are 62 trains that run six days a week, and so on. The third column is a product of the first and second columns, the fourth column gives a cumulative sum of the third column, and the fifth column expresses the values in the fourth column in the percentage notation. Observe that the third column gives the number of train arcs in the weekly space-time network for trains with different frequencies. The table shows that 94% of the train arcs in the space-time network correspond to the trains that run 5, 6, or 7 days.

Table 2-1. Analysis of trains and their frequencies

Train Frequency (A)	Number of Trains (B)	A*B	Cum. Sum of A*B	Cumulative Percentage A*B
7	372	2,604	2,604	78%
6	62	372	2,976	90%
5	29	145	3,121	94%
4	24	96	3,217	97%
3	20	60	3,277	99%
2	16	32	3,309	100%
1	15	15	3,324	100%

We now state an approximation that reduces the size of the MIP substantially and also helps us satisfy the consistency constraints. We create a daily locomotive scheduling

model that is a simplification of the weekly locomotive scheduling model in the following manner. Each train with a frequency of p days per week or larger is a train in the daily model. Each train with frequency strictly less than p days is not included in the daily model. After some analysis and testing, we chose $p = 5$. So, if a train has a frequency of 4 or less in the weekly model, we eliminate it from our daily model. Our daily model is roughly seven times smaller than the weekly model, and accordingly much easier to solve. In addition, since we repeat the train's daily schedule for each day of the week, the solution automatically satisfies the consistency constraints. Hence this simplification achieves both of our goals, which is to reduce the problem size and achieve consistency of the solution.

The solution to the daily model is not feasible for the weekly model. It assigns locomotives to some train arcs that do not exist (those trains that run 5 or 6 days per week) and does not assign locomotives to train arcs that exist (those trains that run 4 days per week or fewer). But for the CSX data, the approximation we make is relatively small compared to the size of the problem. With our choice of $p = 5$, we assign locomotives to 120 train arcs that do not exist in the weekly model and do not assign locomotives to 203 train arcs that do exist in the weekly model. These numbers are small compared to the total number of train arcs, which is 3,324.

We take the solution for the daily model and transform it to a feasible solution to the weekly model in a separate algorithm. We thus solve the locomotive scheduling problem in two stages. The first stage solves the daily locomotive scheduling problem, and the second stage takes in the daily locomotive schedule and modifies it to obtain the weekly locomotive schedule. We will in the next few sections focus on the daily

locomotive scheduling problem, and return to the weekly locomotive scheduling problem in Section 2.7. In our investigations, we chose $p = 5$ primarily because the solution of the model with $p = 5$ was more easily converted to a solution of the weekly locomotive scheduling problem than other values of p.

2.6 Solving the Daily Locomotive Scheduling Problem

In this section, we describe how to solve the daily scheduling problem. We describe (i) how to construct the daily space-time network and formulate the MIP for the daily scheduling problem; and (ii) how to solve the daily locomotive scheduling problem using a decomposition-based approach.

2.6.1 Constructing the Daily Space-Time Network and Formulating the MIP

The daily space-time network is constructed for the daily locomotive scheduling problem where each train is assumed to run each day of the week. It is constructed in a similar manner to the way in which we construct the weekly space-time network. We represent the daily space-time network as $G^l = (N^l, A^l)$. The daily space-time network is about seven times smaller than the weekly space-time network.

In the daily model, the departure time is the number of minutes past midnight that the train departs, and the arrival time is the number of minutes past midnight that the train arrives. The day of the week does not play a role in this daily model. For a train l, let day(l) denote the number of times the train will cross the midnight time line as it goes from its origin to its destination. For example, consider a train that leaves station A on 7 AM on one day and arrives at 4 PM two days later at Station B. For example, the train starts on Monday at 7 AM and arrives at 4 PM on Wednesday. In this case, day(l) = 2. If this train were to repeat each day in our weekly model, then two copies of the train would cross the time line at midnight on Sunday - the train that starts at 7 AM on Saturday and

ends at 4PM on Monday, and the one that starts at 7 AM on Sunday and ends at 4 PM on Tuesday. In general, if $day(l)$ = k in our daily model, then there are k copies of the train l that cross the midnight at Sunday time line in our weekly model. Therefore, assigning a locomotive in our daily model to a train l with $day(l)$ = k corresponds to assigning k locomotives of the same type in the weekly model.

To account for the impact of $day(l)$, we modify the fleet size constraint of the locomotive scheduling problem as follows:

$$\sum_{l \in S} (x_l^k + y_l^k)\, day(l) + s^k = B^k, \quad \text{for all } k \in K.$$

The rest of the formulation of the locomotive scheduling problem (2.1) on the daily space-time network is identical to the one given earlier.

2.6.2 Solving the MIP for the Daily Scheduling Problem

Though the space-time network for the daily scheduling problem was substantially smaller than the space-time network for the weekly scheduling problem, we found it to be too large to be solved to optimality or near-optimality. The daily locomotive scheduling problem consisted of 463 train arcs, and the daily space-time network contained 1,323 nodes and 30,034 arcs. The MIP formulation consisted of 22,314 variables and 9,974 constraints. The LP relaxation took a few seconds to solve but the MIP did not give any integer feasible solution in 72 hours of running time. We conjecture that the biggest source of difficulty was the presence of fixed charge variables z_l (for connection and light arcs) which take value 1 whenever there is a positive flow on arc l. It is well known that MIPs with many fixed charge variables often produce weak lower bounds.

In order to obtain high quality feasible solutions and to keep the total running time of the algorithm small, we decided to eliminate the fixed charge variables from the MIP

formulation using heuristics. The heuristics also allowed us greater flexibility in determining what kind of solution we want. In our formulation, we have two kind of fixed charge variables – one corresponding to connection arcs, and the other corresponding to light arcs. We will first consider the fixed charge variables corresponding to the connections arcs, followed by the fixed charge variables corresponding to light arcs. We also consider some cases in which fixed charge variables can be eliminated without any loss of generality.

2.6.3 Determining Train-Train Connections

Train companies often specify some "hardwired" train-train connections, that is, they specify some inbound trains whose consist must go to the specified outbound trains. These hard-wired train-train connections are easy to enforce. If the inbound train i at a station is hardwired to the outbound train j at that station, then in the space-time network we create the train-train connection arc (i'', j') and do not allow any other connection arc to emanate node i'' or enter node j'. This arc is the only connection arc leaving node i'' and the only connection arc entering node j'. This will ensure that all locomotives brought in by the inbound train i directly go to the outbound train j without consist-busting. In this case, there will be no fixed charge variables corresponding to the trains i and j.

Sometimes, a station has a unique inbound train and a unique outbound train. If this station has no light traveling locomotives coming into it, then all the locomotives brought in by the inbound train will be automatically assigned to the outbound train. Hence, there will be no consist-busting and there will be no need for fixed charge variables. Sometimes, a station has one inbound train and two outbound trains, or it has two inbound trains and one outbound trains. In these cases too, consist-busting cannot be

avoided and there is no need to introduce any train-train connection arcs. Again, there will not be any fixed charge variables; all inbound train(s) send their locomotives to the ground node, and all outbound train(s) will get their locomotives from the ground node.

Railroad companies also have rules about which train-train connections are permissible or desirable. The difference between the arrival time of the inbound train and the departure time of the outbound train must be between some specified lower and upper bounds in order for a valid train-train connection to be made. They also prefer connections between trains of the same class; that is, auto trains sending locomotives to auto trains and merchandize trains to the merchandize trains, etc. They also want that the tonnage and HP requirements of the two trains be similar in order for it to be a desirable train-train connection. Using these business rules, we determine the set of all "candidate" train-train connections. In the next step, we will "fix" some of these candidate train-train connections into hardwired train-train connections and all "unfixed" trains will send (or receive) locomotives to (from) ground nodes. Using these heuristics we eliminate fixed charge variables corresponding to consist-busting in our model.

We fix some candidate train-train connections by solving an iterative procedure that solves a sequence of linear programming relaxations of the daily locomotive scheduling problem given in Section 2.6.1 with some changes. We start with a daily space-time network containing all the candidate train-train connection arcs and candidate light arcs. We next solve the linear programming relaxation of the locomotive scheduling model where there are no fixed charge variables and constraints (that is, constraints (2.1g) through (2.1i) and all the train-to-ground and ground-to-train connections have a large cost, to discourage the flow on such arcs. This is an alternative way of discouraging

consist-busting. Let $\alpha(l)$ denote the total flow of locomotives (of all types) on any arc l in the daily space-time network. We next select a candidate train-train connection arc h with the largest value of $\alpha(h)$. This arc indicates a good potential train-train connection. We make this connection arc the unique connection arc for the two corresponding trains, that is, we make it mandatory for the inbound train to send its locomotive to the corresponding outbound train, and resolve the linear programming relaxation. If this linear programming relaxation is infeasible or increases the cost of the new solution by an amount greater than β, we do not make this train-train connection; otherwise we keep this connection. In any case, we remove arc h from the list of candidate train-train connections. This completes one iteration of our procedure to fix train-train connections. We select another candidate train-train connection arc h with the largest value of $\alpha(h)$ in the current flow solution, and repeat this process until either we have reached the desired number of train-train connections (as specified by some parameter γ), or the set of candidate train-train connections becomes empty.

Our method of identifying the train-train connections is a greedy method. It identifies a promising choice, makes it, and resolves the LP problem to assess the impact of this choice. If it turns out to be a bad choice, it is ignored; otherwise, it is made. By suitably identifying the values of the two parameters β and γ, we can govern the behavior of the algorithm. By choosing the higher value of β, we can increase the number of train-train connections. We can also increase the number of train-train connections by increasing the value of the parameter γ. We show in Section 2.9 that using this heuristic technique we were able to increase the number of train-train connections to 72%, which was substantially higher than the goals originally set by our industry sponsor. In fact,

72% was originally not considered an achievable goal. The parameters β and γ need to be assigned right values so that we get the desired consist-busting. We used β = \$1,000, and varied γ to get different levels of consist-bustings.

2.6.4 Determining Light Travel Arcs

Light travel of locomotives plays an important role in locomotive scheduling and can substantially impact the quality of the solution. Incorporating light travel in the locomotive scheduling model requires two decisions: (i) *candidate light arcs*: what should be the light travel arcs in the space-time network; and (ii) *flow on light travel arcs*: which locomotives will flow on the candidate light arcs. In principle, we could allow light travel of locomotives from any station to any station at any time of the day. To capture all these possibilities of light travel in our network, we would need to add a large number of arcs in the daily space-time network. This would increase the size of the network substantially and also the number of flow variables. In addition, since we associate a fixed charge variable with each candidate light arc, the number of fixed charge variables would be very high. Rather than introducing a large number of light travel arcs, we developed a heuristic procedure to create a small but potentially useful collection of light travel arcs, and another heuristic procedure for selecting a subset of these arcs for light travel. We now describe these procedures in greater detail.

Our first procedure determines the candidate light travel possibilities. For each such arc, we need to decide its origin station, its start time, its destination station, and its end time. The end time can be computed from the origin station, its start time, and its destination station if we know the average speed of the light traveling locomotives. In the railroad network, light traveling locomotives typically travel from "power sources"

(stations where inbound trains require substantially more tonnage/horsepower than the outbound trains) to "power sinks" (stations where outbound trains require substantially more tonnage/horsepower than the inbound trains). We first construct the *space network*, which is the same as the daily space-time network described in Section 2.6.1 except that we ignore the time element. Equivalently, we shrink all the station nodes with different times into a single station node. We next determine the additional availability of pulling power at the power sources and the additional demand for pulling power at the power sinks in the space network of our weekly model. To incorporate both horsepower and tonnage constraints simultaneously, we translate this information into the number of locomotives of some standard type, such as, SD40. This gives us node imbalances of locomotives in the space network. We then solve a minimum cost flow problem (see, for example, Ahuja, Magnanti and Orlin [1993]) in the space network to determine the optimal locomotive flow in the network. If there is a positive flow of locomotives on arc (i, j) in the space network greater than some threshold value (say, two locomotives), then we create candidate light travel arcs from city i to city j in the space-time network departing from city i every 8 hours. For the data provided to us by CSX, we found 58 arcs in the space network with flow greater than 2 locomotives and we created 174 candidate light arcs in the space-time network.

In our formulation described before, we need to create a fixed charge variable for each candidate light arc and we cannot solve the formulation for these many fixed charge variables in the required time. Our second procedure eliminates the fixed charge variables. From the list generated by the previous heuristic, it selects a small subset of light arcs and assumes that there will be light travel on these arcs. The procedure works

as follows. It first solves a linear programming formulation of the locomotive scheduling problem where all the candidate light arcs are added. It then removes all those light arcs which have zero flow. Let $\alpha(l)$ denote the total number of locomotives flowing on an arc l. We then perform the following iterative step. We select the light arc l with the smallest value of flow $\alpha(l)$. Suppose this is arc q. We delete arc q from the network and solve the LP relaxation of the locomotive scheduling problem. If the deletion of this light arc causes the total cost of flow (of the LP relaxation) to go up by at least η units, we do not delete this arc; otherwise we delete this arc and replace α by the new flow. We repeat this process until there are no unexamined light arcs. The number of light arcs generated by this module crucially depends upon the value of the parameter η. If we increase the value of η, then we will increase the number of arcs deleted from the network which will result in lesser number of light arcs being generated. Hence, by assigning suitable values to this parameter, one can generate appropriate number of light arcs. We used η = \$1,000 in our implementation.

2.6.5 Determining Active and Deadheaded Locomotive Flow Variables

As described before, the use of heuristics to determine train-train connections and light travel arcs eliminates the fixed charge variables. We next determine the remaining variables that are the active and deadheaded locomotives on arcs in the network. To determine these variables, we solve the following integer programming problem:

Objective Function**:**

$$\min z = \sum_{l \in TrArcs} \sum_{k \in K} c_l^k x_l^k + \sum_{l \in AllArcs} \sum_{k \in K} d_l^k y_l^k + \sum_{l \in TrArcs} E_l w_l - \sum_{k \in K} G^k s^k \qquad (2.2a)$$

Constraints:

$$\sum_{k \in K} t_l^k x_l^k \geq T_l, \quad \text{for all } l \in TrArcs, \tag{2.2b}$$

$$\sum_{k \in K} h^k x_l^k \geq \beta_l T_l, \quad \text{for all } l \in TrArcs, \tag{2.2c}$$

$$\sum_{k \in K} a^k x_l^k \leq 24, \quad \text{for all } l \in TrArcs, \tag{2.2d}$$

$$\sum_{k \in K} (x_l^k + y_l^k) \leq 12, \quad \text{for all } l \in TrArcs \cup LiArcs, \tag{2.2e}$$

$$\sum_{l \in I[i]} (x_l^k + y_l^k) = \sum_{l \in O[i]} (x_l^k + y_l^k), \quad \text{for all } i \in AllNodes, \text{ for all } k \in K, \tag{2.2f}$$

$$\sum_{k \in K} (x_l^k + y_l^k) + w_l \geq 2, \quad \text{for all } l \in TrArcs, \tag{2.2g}$$

$$\sum_{l \in S} (x_l^k + y_l^k)\, day(l) + s^k = B^k, \quad \text{for all } k \in K, \tag{2.2h}$$

$$x_l^k,\ y_l^k \geq 0 \text{ and integer}, \quad \text{for all } l \in TrArcs, \text{ for all } k \in K, \tag{2.2i}$$

$$w_l \in \{0,1\}, \quad \text{for all } l \in TrArcs. \tag{2.2j}$$

We refer to a solution of (2.2) by (*x, y*), since *w* can be deduced using the solution (*x, y*). The integer programming model here is much easier to solve than the model described in Section 2.6.1. It does not contain fixed charge variables and has fewer locomotive flow variables. For the locomotive scheduling problem we solved, it had 2,315 x_l^k variables, 6,120 y_l^k variables, 463 w_l variables, and 7,782 constraints. We solved (2.2) using CPLEX 7.0, and set the CPLEX parameters so that a very good integer solution is obtained in the early part of the branch and bound algorithm. We found that CPLEX 7.0 gave a very good solution within 15 minutes of execution time, but it did not terminate even when it was allowed to run for over 48 hours. We also found that the algorithm, when run for 24 hours, did not improve much over the best integer solution

obtained within 15 minutes; hence prematurely terminating the algorithm after 15 minutes did not affect much the quality of the solution obtained.

2.6.6 Neighborhood Search Algorithm

The integer solution obtained by the integer programming software CPLEX 7.0 is not, in general, an optimal solution of the locomotive scheduling problem. We found that this solution can be easily improved by a modest perturbation of the solution. We used a neighborhood search algorithm to look for possible improvements. A neighborhood search algorithm typically starts with an initial feasible solution and repeatedly replaces it by an improved neighbor until we obtain a solution which is at least as good as its neighbors. At this point the current solution is called a local optimal solution (Aarts and Lenstra [1997]). We call a neighborhood search algorithm a *Very Large-Scale Neighborhood (VLSN) Search Algorithm* if the size of the neighborhood is very large, possibly exponential in terms of the input size parameters. We use the solution obtained by the integer programming software as the starting solution and improve it using a VLSN search algorithm.

The locomotive scheduling problem can be conceived of as a multicommodity flow problem with side constraints in the space-time network where the flow of each locomotive type defines a commodity. When viewed in this manner, the constraints (2.2f) define the flow balance constraints of each commodity and the remaining constraints define the side constraints. The multicommodity flow problem is polynomially solvable when the flow variables x_l^k and y_l^k take real values, but is NP-complete when flow variables are required to take integer values. (For the time being, we are ignoring the fixed penalty for single locomotive trains captured by the constraints (2.2g).)

We define the neighborhood for a feasible solution $(\bar{x}, \bar{y})$ of (2.2) as follows: For a specified value of $k \in K$, send $\delta > 0$ locomotives of type k along a cycle in the daily space-time network so that the flow balance and all side constraints remain satisfied. The new solution is a lower cost solution (or a better neighbor) if the cycle along which the flow is sent is a negative cost cycle. In this neighborhood, the neighborhood search algorithm would proceed by identifying negative cost cycles with respect to a given solution $(\bar{x}, \bar{y})$ and a locomotive type k and augmenting flows along these cycles until there is no negative cost cycle for any locomotive type k.

We now describe how we identify negative cost cycles for a given locomotive type k. Let α_l^k denote the total number of locomotives of type k on arc l (that is, $\alpha_l^k = x_l^k + y_l^k$). We first create a *residual network* $G(\alpha, k)$, similar to those used in solving minimum cost flow problems (see, for example, Ahuja, Magnanti and Orlin [1993]). We consider each arc $l \in AllArcs$ and add arcs to the residual network in the manner described below:

- Case 1: If we can send additional locomotives of type k on arc l, then we add arc l to the residual network. The capacity u_l^k of the arc l is equal to the maximum number of locomotives of type k that can be sent on arc l without violating any of constraints in (2.2b)-(2.2e) and (2.2h)-(2.2j). The cost f_l^k of the arc l *is* equal to the increase in the cost of sending one additional locomotive on the arc l.
- Case 2: If we can reduce locomotives of type k on arc l, then we add the reversal of arc l, say, arc $\bar{l}$, to the residual network. The capacity $u_{\bar{l}}^k$ of the arc $\bar{l}$ is equal to the maximum number of locomotives of type k that can be reduced on arc l without violating any of constraints in (2.2b)-(2.2e) and (2.2h)-(2.2j). The cost $f_{\bar{l}}^k$ of the arc $\bar{l}$ equals the decrease in the cost of sending one fewer locomotive of type k on arc l.

For any negative cost directed cycle W in the residual network, there is a way to improve in the current solution. If the cost of cycle W is c_W and if its capacity is δ units (the capacity of W is the minimum capacity of an arc in W), then δ locomotives of type k can be sent around the cycle W resulting in a total savings of δc_W. There exist several efficient negative cycle detection algorithms (see, for example, Ahuja, Magnanti and Orlin [1993], and Goldberg et al. [1995]), and any of these algorithms can be used to identify negative cycles.

The neighborhood for a given solution $(\overline{x}, \overline{y})$ is defined so that each neighbor is any solution that can be obtained by sending one or more units of flow around one of the directed cycles in the residual network. The number of directed cycles in each residual network can be very large and may not be polynomially bounded in terms of the number of nodes and arcs in the network. Hence our neighborhood search algorithm is a very large-scale neighborhood (VLSN) search algorithm. In the VLSN search algorithms, the size of the neighborhood is too large to be searched explicitly and we use implicit enumeration methods to identify improved neighbors. Algorithms based on VLSNs have been successfully used in the context of optimization problems (Ahuja et al. [2001a, 2001b, 2001c, 2002]). To apply this algorithm, we start with a feasible solution of the locomotive scheduling problem, select a locomotive type *k*, construct the residual network, identify negative cycles in it, and send maximum possible flow along this cycle. We then update the residual network and again send flow along a negative cycle. We repeat this process until there is no negative cycle in the residual network. At this time, we select another locomotive type *k′* and reapply the method. We terminate when no negative cycle is found for any locomotive type.

This algorithm, when implemented, ran very efficiently. We were able to obtain a local optimal solution in a few seconds of computer time. The algorithm obtained some improvements over the solution obtained by CPLEX. Although our neighborhood was quite large, ultimately we decided that it was not large enough. The side constraints (2.2h) were often tight and did not permit many locomotives to be rerouted without violating the constraints. In order to find more substantial improvements, we need to permit flows to change in many arcs simultaneously. This observation motivated us to consider another neighborhood that we describe next. This neighborhood can be considered as a specific implementation of the concept of *referent domain optimization* briefly described in Glover and Laguna [1997].

Let $(\bar{x}, \bar{y})$ be a feasible solution of (2.2). We call a solution (x, y) a *neighbor* of $(\bar{x}, \bar{y})$ if (x, y) is feasible for (2.2b)- (2.2j) and it differs from $(\bar{x}, \bar{y})$ for one locomotive type only, that is $\bar{x}^q = x^q$ and $\bar{y}^q = y^q$ for all $q \in K \setminus \{k\}$ for some locomotive type *k*. In other words, all the feasible locomotive flows that can be obtained by changing the locomotive flow for one locomotive type only define the neighborhood of the solution (x, y). Mathematically, all the solutions of the following integer program define the neighborhood of the solution $(\bar{x}, \bar{y})$.

Objective Function:

$$\min\ z^k = \sum_{l \in TrArcs} c_l^k x_l^k + \sum_{l \in AllArcs} d_l^k y_l^k + \sum_{l \in TrArcs} E_l w_l - G^k s^k \qquad (2.3a)$$

Constraints:

$$t_l^k x_l^k \geq T_l - \sum_{q \in K \setminus \{k\}} \bar{x}_l^q, \quad \text{for all } l \in TrArcs, \qquad (2.3b)$$

$$h^k x_l^k \geq \beta_l T_l \; - \sum_{q \in K \setminus \{k\}} h^q \overline{x}_l^q, \quad \text{for all } l \in TrArcs \tag{2.3c}$$

$$a^k x_l^k \leq 24 \; - \sum_{q \in K \setminus \{k\}} a^q \overline{x}_l^q, \quad \text{for all } l \in TrArcs, \tag{2.3d}$$

$$x_l^k + y_l^k \leq 12 \; - \sum_{q \in K \setminus \{k\}} (\overline{x}_l^k + \overline{y}_l^k), \quad \text{for all } l \in TrArcs \cup LiArcs, \tag{2.3e}$$

$$\sum_{l \in I(i)} (x_l^k + y_l^k) = \sum_{l \in O(i)} (x_l^k + y_l^k), \quad \text{for all } i \in AllNodes \tag{2.3f}$$

$$x_l^k + y_l^k + w_l \geq 2 \; - \sum_{q \in K \setminus \{k\}} (\overline{x}_l^k + \overline{y}_l^k), \quad \text{for all } l \in TrArcs, \tag{2.3g}$$

$$\sum_{l \in AllArcs} (x_l^k + y_l^k) day(l) + s^k = B^k, \tag{2.3h}$$

$$x_l^k, y_l^k \geq 0 \text{ and integer}, \quad \text{for all } l \in TrArcs, \tag{2.3i}$$

$$w_l \in \{0,1\}, \quad \text{for all } l \in TrArcs. \tag{2.3j}$$

This new neighborhood subsumes our previous cycle-based neighborhood and is also a very large-scale neighborhood. We call the solution $(\overline{x}, \overline{y})$ a local optimal solution if $(\overline{x}, \overline{y})$ is an optimal solution of (2.3) for each $k \in K$. Thus, a solution $(\overline{x}, \overline{y})$ is a local optimal solution of the locomotive scheduling problem if each single locomotive type is optimally scheduled when the schedule of other locomotive types is not allowed to change. Though (2.3) is an integer programming problem with 2,168 variables and 3,437 constraints, CPLEX solved to optimality within a few seconds.

To summarize, we take the solution provided by the integer programming software for the daily scheduling problem (2.3) as the starting solution of our neighborhood search algorithm and solve a sequence of problems (2.3) for all locomotive type $q \in K$. We stop when the solution cannot be improved for any locomotive type. The solution at this stage is a local optimal solution.

2.7 Solving the Weekly Locomotive Scheduling Problem

We now describe how we solve the weekly locomotive scheduling problem. Recall from our discussion in Section 2.5 that to handle the consistency constraints and to keep the problem size manageable, we solve the problem in two stages. The first stage assumes that all trains run every day of the week. To satisfy this assumption, we eliminate trains which run fewer than p days (for example, $p = 5$), and all other trains are assumed to run every day of the week. When we apply the solution of the daily scheduling to the weekly scheduling problem by repeating it every day, then we provide locomotives to some trains that don't exist and do not provide locomotives to those trains that do exist. To transform this solution into a feasible and effective solution for the weekly scheduling problem, we take locomotives from the trains that exist in the daily problem but do not exist in the weekly problem and assign them to the trains that do not exist in the daily problem but exist in the weekly problem, and possibly use additional locomotives to meet the constraints. In this section, we describe this method in greater detail.

We define some notation to simplify the presentation. There are two kinds of trains in the locomotive scheduling problem: (i) Φ : trains that operate p or more days a week; and Ψ : trains that operate fewer than p days a week. Let Φ^1 denote the set of train arcs corresponding to the trains in Φ in the daily space-time network G^1, and Φ^7 denote the set of train arcs corresponding to the trains in Φ in the weekly space-time network G^7. Observe that each arc in Φ^1 induces p or more corresponding arcs in Φ^7. Let Ψ^7 denote the set of train arcs corresponding to the trains in Ψ in the weekly space-time network G^7. Observe that trains in Ψ do not have any corresponding train arc in G^1. Also observe that in the weekly space-time network, TrArcs = $\Phi^7 \cup \Psi^7$.

Suppose that the optimal solution for the daily locomotive scheduling problem produces the following solution:

- $\overline{x}_l^k$: The number of active locomotives of type k assigned to train arc $l \in \Phi^1$.
- $\overline{y}_l^k$: The number of deadheaded locomotives of type k assigned to train arc $l \in \Phi^1$.

We have the following objectives when we solve the weekly locomotive scheduling problem:

(i) Each arc $l \in \Phi^7$ has a corresponding arc in Φ^1, say, $\overline{l}$. The locomotive assignment of arc $\overline{l} \in \Phi^1$ in the daily locomotive scheduling problem becomes the "target flow" for the corresponding arc $l \in \Phi^7$ in the weekly locomotive scheduling problem. We would like the locomotive flow on each arc in Φ^7 be as close to its target flow as possible as this would ensure that the weekly locomotive schedule is consistent. Changes to the target flow can be made but they are penalized as it may affect the consistency of the solution. Henceforth, we will assume that $\overline{x}_l^k$ and $\overline{y}_l^k$ denote the target flow for each arc $l \in \Phi^7$.

(ii) Each arc $l \in \Psi^7$ should be assigned sufficient number of locomotives so that its horsepower and tonnage requirements are satisfied. Observe that we do not consider consistency for train arcs in Ψ^7. Only 10% of the train arcs are in Ψ^7 and ignoring consistency of these arcs will not have a major impact on the overall consistency of the solution.

(iii) If there is a train-train connection from train A to train B in the daily space-time network, then the same train-train connection should be carried over to the weekly space-time network on all the days when both the trains A and B run.

We now describe the weekly space-time network. The objective (iii) requires that we slightly modify the weekly space-time network defined earlier in Section 2.3. Recall that in the daily space-time network, we have train-train connection arcs which force the locomotives to go from an inbound train to an outbound train without consist-busting. When we create the weekly space-time network, we create these train-train connections on all days wherever it is possible. For example, suppose that the inbound train l_1 runs Monday through Friday, the outbound train l_2 runs Wednesday through Sunday, and we

had a train-train connection from l_1 to l_2 in the daily space-time network. Then in the weekly space-time network, we will have train-train connection arcs only on Wednesday through Friday when both the trains run. On all other days, the train l_1 will send its locomotives to the ground node and the train l_2 will get its locomotive from the ground node. Those trains that do not have a train-train connection in the daily space-time network, send their locomotives to the corresponding ground node and get their locomotives from the corresponding ground node. Thus, each arrival node has exactly one outgoing arc; either it is a train-train connection arc or it is a train-ground connection arc. Similarly, each departure node has exactly one incoming arc. Either it is a train-train connection arc or it is a train-ground connection arc. The weekly space-time network also has light arcs. Each light travel arc in the daily space-time network is repeated seven times in the weekly space-time network to allow the possibility of light travel each day of the week.

We now discuss how to determine a locomotive schedule in the weekly space-time network. We first formulated this problem as an integer programming problem, which is similar to the IP formulation (2.1) presented in Section 4. But we were unable to solve it using CPLEX 7.0. The integer programming problem for the weekly space-time network is about seven-times larger than the corresponding problem for the daily space-time network, and CPLEX ran for 72 hours without obtaining any integer feasible solution. Since our goal was to solve the locomotive scheduling problem within 30 minutes of computational time, we needed to follow another approach.

We next attempted to determine the locomotive schedule one locomotive type at a time. This converted a multicommodity flow problem (with each locomotive type

defining a commodity) with side constraints into a sequence of single commodity flow problems with side constraints, one for each locomotive type. We found that these single commodity flow problems with side constraints were solved very efficiently by CPLEX 7.0. We now describe this approach in greater detail. We first arranged different locomotive types in some order, and then considered them one by one in this order.

Suppose we are considering locomotive type k. Our goal is to determine that schedule of locomotive type k so as to:

Objective Function:

$$\min \sum_{l\in\Psi}(c_l^k - M)x_l^k + \sum_{l\in\Phi} c_l^k x_l^k + \sum_{l\in AllArcs} d_l^k x_l^k + \sum_{l\in TrArcs} E_l w_l - G^k s^k + P\sum_{l\in\Phi}(\alpha_l^{+k} + \alpha_l^{-k}) \quad (2.4a)$$

subject to (2.3b)-(2.3g), (2.3i)-(2.3j), and the following constraints:

$$x_l^k \le U_l^k \quad \text{for all } l \in \Psi, \quad (2.4b)$$

$$\alpha_l^{+k} - \alpha_l^{-k} = (x_l^k + y_l^k) - (\overline{x}_l^k + \overline{y}_l^k) \quad \text{for all } l \in \Phi, \quad (2.4c)$$

$$\alpha_l^{+k}, \alpha_l^{-k} \ge 0 \text{ and integer}, \quad \text{for all } l \in TrArcs, \quad (2.4d)$$

$$\sum_{l\in S}(x_l^k + y_l^k) + s^k = B^k, \quad \text{for all } k \in K, \quad (2.4e)$$

$$U_l^k = \max\left\{\frac{T_l - \sum_{q\in K\setminus\{k\}} t_l^q \overline{x}_l^q}{t_l^k}, \frac{\beta_l T_l - \sum_{l\in K\setminus\{k\}} h_l^q \overline{x}_l^q}{h^k}\right\} \quad (2.4f)$$

where M is a large positive number, and U_l^k denotes the number of active locomotives of type k needed to meet the tonnage and horsepower requirements of the train arc l. This formulation is similar to the formulation (2.3), which is to find the optimal flow of locomotives of type k while keeping the flow of other locomotive types intact. In addition, the formulation attempts to send the sufficient number of locomotives

on arcs in Ψ ; this is achieved by subtracting a large positive number *M* from the active locomotive costs of arcs in Ψ in the objective function (2.4a). The formulation also attempts to find a flow which is "close" to the target flow on arcs in Φ ; this is accomplished by measuring the positive or negative deviation (α_l^{+k} or α_l^{-k}) from the target flow through the constraints (2.4c) and (2.4d), and penalizing it in the objective function (2.4a). By assigning a suitable value to the coefficient *P*, we make the locomotive flow (x^k, y^k) sufficiently close to the target flow on arcs in the set Φ. In our computational investigations, we used *P* = $1,000.

We solved the formulation (2.4) multiple times and with different objective functions to obtain improved solutions. For example, our primary objective is to provide sufficient number of locomotives to trains in Ψ . We accomplish it by choosing a sufficiently large value of M (say, 10^6), and solving (2.4) for all locomotive types. Once all trains have sufficient number of active locomotives, we set $M = 0$ and solve it again for all locomotive types. Depending upon the value of *P*, the optimal solution will balance the total flow cost and the penalty for deviation from the target flow. We may need to experiment with different values of *P* before we obtain the right balance between the two terms.

For our data, the formulation (2.4) is defined over a network with 8,798 nodes and 9,139 arcs. The formulation (2.4) comprises of 15,266 variables and 13,142 constraints. CPLEX 7.0 was able to solve it for one locomotive type in a few seconds. We took a total of 2-3 minutes to obtain a satisfactory solution in the weekly space-time network. Whereas the integer programming formulation of the weekly space-time network was

intractable, solving it heuristically using the constrained network flow algorithm was quite efficient.

After we obtained a feasible solution of the weekly locomotive scheduling problem, we applied neighborhood search algorithm to it. We considered each locomotive type one by one and tried to determine the optimal flow of the selected locomotive type while keeping the flow of other locomotive types intact. This subproblem is a (single commodity) constrained minimum cost flow problem and can be very efficiently solved. We terminate when the solution value does not improve for any locomotive type. Observe that this algorithm can also be regarded as a very large-scale neighborhood search algorithm and is a modification of the neighborhood search algorithm described in Section 2.6 for the daily space-time network.

Recall our discussion in Section 2.3 that we create train-train connections between several inbound and outbound trains to reduce consist-bustings. After we have obtained the solution of the weekly locomotive scheduling problem, we can create additional train-train connections. After we have solved the daily locomotive scheduling problem, we know the consists of all trains. Suppose that an outbound train l_2 at a station has the same consist as that of an inbound train l_1 at the same station and locomotives from train l_1 can feasibly connect to train l_2, Then we can create a train-train connection between l_1 and l_2, provided locomotives coming from train l_1 are not needed by another train departing before train l_2. We developed an efficient algorithm (details are omitted here) to determine a maximal number of train-train connections that can be made between inbound and outbound trains. We applied this algorithm at each station to identify

additional train-train connections. For the data provided to us by CSX, we were able to create about 10-15% additional train-train connections by the use of this algorithm.

2.8 Summary of the Algorithmic Approach

We now summarize our algorithm to solve the weekly locomotive scheduling problem. Figure 2-2 gives the steps of our algorithm.

algorithm *locomotive scheduling*;

begin

construct the daily space-time network including light arcs (see Section 2.5);

solve a sequence of LPs in the daily space-time network to determine train-train connections (see Section 2.6.3);

solve a sequence of LPs in the daily space-time network to determine light travel arcs (see Section 2.6.4);

solve the MIP problem (2.2) to determine the active and deadheaded locomotives for the daily locomotive scheduling problem (see Section 2.6.5);

solve a sequence of (single commodity) integer programs (2.3) to improve the solution obtained by the MIP (see Section 2.6.6);

use the solution of the daily locomotive scheduling problem to construct the target solution f the weekly locomotive scheduling problem (see Section 2.7);

solve a sequence of single commodity integer programs (2.4) to obtain a solution of the weekly locomotive scheduling problem (see Section 2.7);

identify additional train-train connections between inbound and outbound trains at each station (see Section 2.8);

end;

Figure 2-2. Summary of the algorithmic steps.

2.9 Computational Results

In this section, we present our computational results. We were supplied data files for several scenarios by CSX Transportation. However, for the sake of consistency, we performed most of our computational results on one scenario only. The scenario we tested comprised of 3,324 trains originating from and terminating at 119 stations. It contained 3,316 locomotives belonging to five locomotive types. All of our algorithms were implemented in C++ and we made extensive use of callable libraries in CPLEX 7.0.

The algorithms were run and tested on a Pentium III PC with a speed of 750 Megahertz.

We call our software the *Advanced Locomotive Scheduling* (*ALS*) system.

Table 2-2. Summary of problem sizes and solution times at different steps of the algorithm.

Problem	Problem Size	Solution Time
MIP model (2.1) for the weekly locomotive scheduling problem	The weekly network contains 8,798 nodes and 30,134 arcs, of which 3,324 are train arcs, 117 light arcs, 24,577 connection arcs, and 2,116 ground arcs. The MIP formulation contains 197,424 integer variables and 67,414 constraints.	The MIP problem is too large to be solved by the available MIP software.
MIP model (2.1) for the daily locomotive scheduling problem	The daily network contains 1,134 nodes and 3,965 arcs, of which 463 are train arcs, 26 light arcs, 3,178 connection arcs, and 298 ground arcs. The MIP formulation contains 22,314 integer variables and 9,974 constraints.	The MIP problem is too large to be solved by the available MIP software.
Identifying train-train connections	The daily network contains 1,134 nodes and 3,965 arcs, of which 463 are train arcs, 26 light arcs, 3,178 connection arcs, and 298 ground arcs. The MIP formulation contains 22,314 integer variables and 9,974 constraints.	We solve 132 LP relaxations to determine train-train connection arcs. The total time is 11 minutes.
Identifying light travel arcs	The daily network contains 1,134 nodes and 1,276 arcs, of which 463 are train arcs, 26 light arcs, 489 connection arcs, and 298 ground arcs. The MIP formulation contains 6,385 integer variables and 7,835 constraints.	We solve 14 LP relaxations to determine light traveling arcs. The total time is 30 seconds.
Determining locomotive flow variables	The daily network contains 1,110 nodes and 1,242 arcs, of which 463 are train arcs, 12 light arcs, 475 connection arcs, and 292 ground arcs. The MIP formulation contains 8,993 integer variables and 7,782 constraints.	We are able to get the first integer solution in 5 minutes, and a very good solution in 15 minutes.
Solving the weekly locomotive scheduling problem	For each locomotive type, there are 10,768 variables, and 14,592 constraints.	We solve 15 constrained minimum cost flow problems. The total time is 1 minute.

Table 2-2 summarizes the sizes of the MIP and LP problems we solved in various stages, the number of such problems solved, and the time taken to solve these problems. The MIP problem we solved to obtain a solution of the daily locomotive assignment problem was solved heuristically, as the algorithms did not terminate even after several hours of running time. We set the CPLEX parameters so that a very good integer solution to the MIP was obtained early on by the software.

Table 2-3. Comparison of number of locomotives used by the LSM and ALS solutions

Type of Locomotives	ALS	LSM
SD40	249	498
SD50	160	171
C40	487	621
AC44	154	164
AC60	160	160
Total	1210	1614

Table 2-4. Comparison of other statistics for ALS and LSM solutions

Performance Measure	ALS	LSM
Active Time	44.4%	31.3%
Deadheading Time	8.1%	19.6%
Idling Time	46.7%	49.1%
Light Traveling Time	0.8%	0%
Total Cost	$9.2 million	$13.5 million
Consist-Busting Rate	49.4%	85.0%

Table 2-3 and Table 2-4 compare the solution obtained by ALS with the solution obtained by the CSX software, called *LSM* (*Locomotive Scheduling Model*). The ALS solution is substantially superior to the LSM solution; it reduces the total cost

substantially and dramatically decreases the number of locomotives used. The ALS solution used more than 400 fewer locomotives than the LSM solution with medium consist busting (50%) and medium light travel (0.8%). Recall from Section 2.1 that LSM handles consist assignment and locomotive scheduling separately, and in the locomotive scheduling phase considers each locomotive type one by one. We achieved the dramatic decrease in the number of locomotives, in comparison with the LSM, mainly by combining consist assignment and locomotive scheduling, and considering all locomotive types together in place of each locomotive type one by one.

Table 2-5. Sensitivity of the solution to three levels of light travel

Consist-Busting Rate: 50%	No Light Travel	Medium Light Travel	Sufficient Light Travel
Total Number Of Locomotives Used	1268	1210	1192
Total Cost	$9,431,228	$9,242,092	$9,182,314
Percentage Active Time	39.14%	44.44%	45.26%
Percentage Deadheading Time	12.37%	8.09%	7.33%
Percentage Idling Time	48.49%	46.67%	46.06%
Percentage Light Traveling Time	0.00%	0.80%	1.35%

We reduce the number of locomotives by substantially reducing the fraction of the locomotives that deadhead. Table 2-6 presents the statistics on the percentage of the times locomotives are actively pulling the trains, deadheading on trains, light traveling, or idling at stations (for maintenance or just waiting to be assigned to outbound trains). We observe that in the ALS solution the percentage of the time locomotives are actively

pulling the trains is about 13.1% more than the LSM solution. Hence our solution significantly increases the locomotive productivity. We also observe that in the ALS solution, the percentage of the time locomotives deadhead on trains is considerably less than in the LSM solution

Table 2-6. Sensitivity of the solution to light different levels of consist-busting

	Very Hight Consist-Busting	Hight Consist-Busting	Medium Consist-Busting	Low Consist-Busting	Minimum Consist-Busting
Consist Busting Rate	66.39%	59.48%	49.42%	39.09%	31.06%
Number Of Locomotive Used	1155	1165	1210	1281	1340
Total Cost	$9,052,779	$9,064,822	$9,242,092	$9,532,021	$9,772,958
Percentage Active Time	46.68%	45.74%	44.44%	42.67%	41.25%
Percentage Deadheading Time	6.28%	6.79%	8.09%	9.99%	9.57%
Percentage Idling And Dwelling Time	46.23%	46.68%	46.67%	46.35%	48.04%
Percentage Light Traveling Time	0.81%	0.79%	0.80%	0.99%	1.14%

We also conducted some tests to measure the sensitivity of the solution to the extent of light travel and the number of train-train connections. Table 2-5 shows the statistics of the final solution when we allow three levels of light travel: no light travel, medium light travel, and sufficient light travel. We observe that a reasonable level of light travel can increase the locomotive utilization rate significantly. Table 2-6 shows the statistics of the final solution when we consider several levels of train-train connections.

We observe that consist-busting can be decreased at the expense of lower locomotive utilization rate and requiring more locomotives.

2.10 Summary and Conclusions

In this chapter, we present the results of a study of a locomotive scheduling problem faced by CSX Transportation, a major U.S. railroad company. We focused on the planning version of the locomotive scheduling problem, where we assign locomotive types to various trains. Our goal was to develop excellent plans along a number of dimensions. Our primary metric for evaluating a solution was to compare it to the existing software used by the locomotive planning division at CSX. The existing software was in-house software developed over a period of ten years. While it has the substantial advantage of capturing many constraints and objectives not dealt with in the existing literature, it also has some important limitations. In particular, it was unable to produce acceptable solutions without considerable manual adjustment of the solution produced. Our approach to solve the planning version of the locomotive scheduling problem produced solutions that satisfied the constraints and business rules specified by CSX and offered considerable savings in cost, especially in terms of a significant reduction in the number of locomotives needed. Our model is the first approach to account for consist busting, light travel, and consistency of the solution in a unified framework. Our approach relies upon breaking the locomotive scheduling problem into a two-step process: first solving the daily locomotive scheduling problem, and then solving the weekly locomotive scheduling problem. This approach works reasonably well for those situations where a large number of trains run most days in a week.

By incorporating more realistic constraints and costs in our models, we needed to rely on a multistage heuristic procedure to solve the model. In particular, we were unable

to solve MIPs involving many fixed charge constraints. We avoided this with a sequential heuristic procedure for determining the three sets of decision variables (i) light travel, (ii) train-train connections, and (iii) active and deadhead locomotive flow variables sequentially instead of determining them in a single model. Nevertheless, our sequential procedure heavily exploited information from a sequence of LPs, and thus relied on a system wide view of the entire planning problem. When we determine light travel arcs, we consider locomotive flow variables are also part of the LPs solved. Similarly, when we determine train-train connection variables, we again consider locomotive flow variables in the decision process. Hence we do achieve a certain level of integration in our sequential approach. Perhaps in the future we will be able to obtain a better integration of these different sets of variables.

From the CSX perspective, this research demonstrated that advances in OR techniques enabled a simultaneous locomotive assignment and scheduling solution on a much larger network than previously thought possible. CSX used the model in support of its operations planning effort associated with its acquisition of Conrail territory, a 30% increase in network size. Without the aid of the model, CSX would not have been confident in its ability to execute the new integrated operating plan. In addition, CSX executed a major locomotive trade with a locomotive leasing company where they swapped 156 older, low and medium horsepower locomotives for 31 newer, high horsepower locomotives. The model helped quantify the network benefits of re-centering the fleet composition. In the future, CSX is interested in expanding on this work to include locomotive fueling and servicing constraints into the algorithm. CSX also intends to introduce decision support tools into its tactical management process.

CHAPTER 3
SOLVING THE REAL-LIFE RAILROAD BLOCKING PROBLEM

3.1 Introduction

Most railroad companies in the U.S. are engaged in rail freight transportation. Demand for freight transportation is usually expressed in terms of tonnage of certain commodities to be moved from an origin to a destination. Given these demands, the railroad must establish a set of operating policies that will govern the routing of trains and freight. For every origin-destination pair of traffic demand, the corresponding freight may be shipped either directly or indirectly. When demand is important enough, delivery delays are obviously minimized by using direct trains as opposed to sending the traffic through a sequence of links. However, when demand is does not warrant the dispatching of direct trains, delays are inevitable. Either the traffic is consolidated and routed through intermediate nodes, or freight cars have to wait at the origin node until sufficient tonnage has been accumulated. To benefit from economies of scale, trains are thus often formed by grouping cars with various commodities and having different origins and destinations. These trains operate between particular nodes of the network, called *classification yards*. At these yards, cars are separated, sorted according to their final destination, and combined to form new outbound trains. However, the classification process is labor and capital intensive because many workers and large quantities of equipment are needed to sort the traffic, and construction and maintenance of large yards is necessary to handle the sorting task. To prevent shipments from being reclassified at every yard they pass through, several shipments may be grouped together to form a *block*. Cars in the same

block may then pass through a series of intermediate classification yards, being separated and reclassified only after they have reached the destination of the block. The *blocking policy* specifies what blocks should be built at each yard of the network and which cars should go into each block.

A blocking policy is usually specified as follows: cars at yard *i* which are destined for yard *j* must be added to a block that will next be shipped to yard *k* (possibly transiting by other intermediate yards). As explained earlier, cars in a block will not be reclassified until the block reaches its final destination. A blocking model thus places the emphasis on the movement of cars as opposed to the movement of trains. Its solution indicates the routing of freight through the network and the distribution of classification work among yards, but does not specify the trains to be run or the assignment of blocks to trains. Instead, an additional problem must then be solved to determine the routing of trains and their makeup. We next describe the blocking problem in more detail with respect to physical rail system.

The physical rail system consists of undirected links (one or more railroad tracks) connecting nodes (rail yards). Classification usually is done on a group of classification tracks that, together, roughly resemble a hexagon (or half-hexagon) with different length tracks. Researchers on the blocking plan have traditionally assumed that an acceptable approximation of the classification capacity is a rectangle with the same area as the hexagon. We do not use these approximations to determine yard capacity. Rather we use data provided by the railroad based on their experience for each of their yards. Also, in our model, we do not limit the size of the blocks based on the physical characteristics of the yard because train frequency will determine how many cars, on average, are present

on a given classification track. Cost factors that may be considered in an instance of blocking problem include costs based on the distance traveled (car-miles), the time elapsed (car-hours) and the re-classifications (handlings). Other costs that may be considered are usage fees for links belongings to other railroads and premiums for using routings that are serviced by older, less efficient locomotives. Link-specific costs depending on the grades involved or congestion in an urban area, might also be included, as well as link or station-dependent labor and equipment costs. There may also be a fixed cost associated with choosing to build a given block. As supplied by the railroad company, we consider the costs of distance traveled, link-specific impedance factor and car handling cost as objective function.

Mathematically, the blocking problem is designing a network, called a *blocking network*, and routing shipments over this network so as to optimize the total shipment cost. Figure 3-1 gives a sample blocking network, where there are three types of nodes: *origins* (where shipments originate), *yards* (where shipments are reclassified), and *destinations* (where shipments terminate). (We show here a simplified network as in practice, yards can be origins as well as destinations, and nodes can send as well as receive shipments.) Each arc in the network represents a block with the origin at the tail of the arc and destination at the head of the arc. There is an associated cost c_{ij} with each arc (i, j), representing the shortest path length from node i to node j in physical railroad network. Shipments travel from their origins to their destinations over this blocking network. The arcs on the path of travel for a shipment in blocking network give the sequence of blocks the shipment will follow. Consider, for example, the first shipment S_1,

which originates at node 1 and destined for destination T_1 at node 9. This shipment can follow two paths shown in Table 3-1.

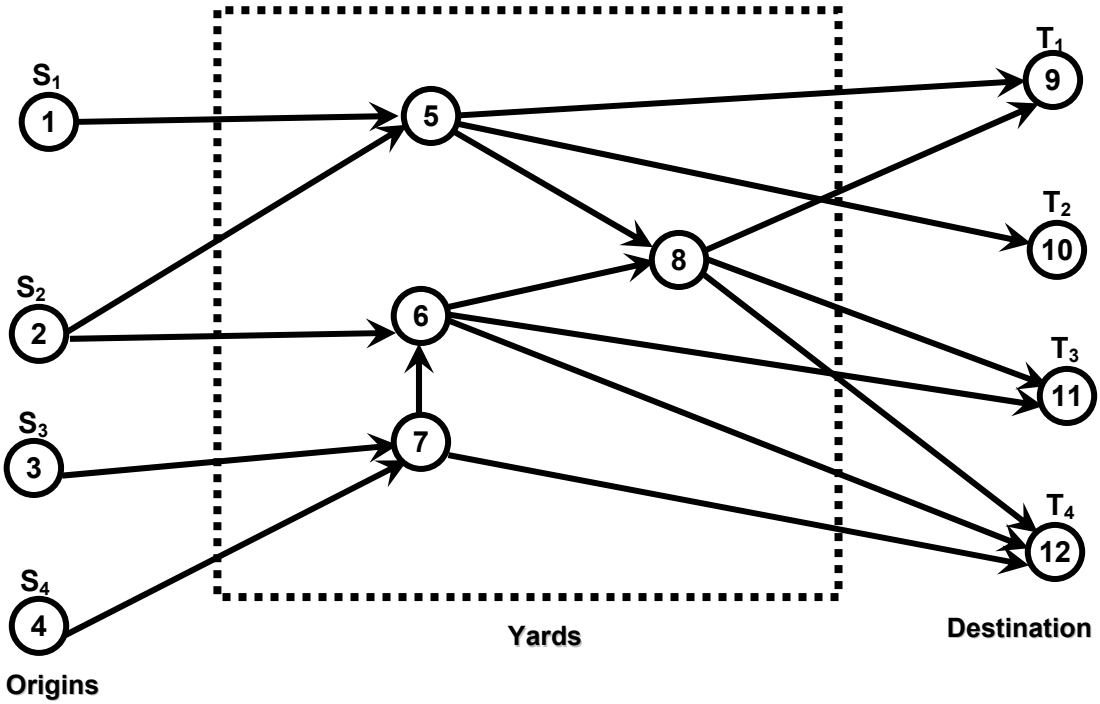


Figure 3-1. An example of a blocking network.

Table 3-1. Blocking paths illustration with respect to Figure 3-1.

Path	Length Of The Blocking Path	Number Of Intermediate Handlings
1-5-9	$c_{15} + c_{59}$	1
1-5-8-9	$c_{15} + c_{58} + c_{89}$	2

This chapter is organized in seven sections. Section 3.2 describes the past attempts made in solving blocking problem. In section 3.3, our IP formulation and new approaches to solve IP formulation along with their limitation is described. The very large scale neighborhood search technique based heuristics to solve blocking problem is presented in section 3.4 and additional constraints are described in section 3.5. Sections 3.6 and 3.7 are for computational results and conclusions respectively.

3.2 Background

One of the first models for car blocking belongs to Bodin et al. [1980]. This paper formulates the blocking problem as non-linear mixed integer programming problem. The model, which is a multicommodity flow problem with additional side constraints, simultaneously determines the optimal blocking strategies for all the classification yards in a railroad system. Besides flow equations that constitute the backbone of the model, yard capacity and block formation constraints are also considered. In particular, the model imposes upper bounds on the number of cars that may be formed in any given yard. With some manual intervention, the authors solved an instance with 33 classification yards and found a solution with 3% of a tight lower bound. Assad [1983] proposes a solution approach for a problem defined on a line network composed of n yards, with traffic flowing from yard 1 to yard n. A dynamic programming formulation with relaxed assumptions led to an efficient solution method. Van Dyke [1986, 1988] describe an interactive heuristic approach that attempts to improve an existing blocking plan by solving a series of shortest-path problems on a network whose arcs represent available blocks. Keaton [1989, 1992] have developed a non-linear MIP for blocking problem, Huntley [1995] has developed a simulated annealing approach, and Gorman [1995] presents a genetic algorithm for constructing operating plans including blocking, train connections, and block-to-train assignments.

Most recent and promising works for blocking problem are by Newton et al. [1998] and Barnhart et al. [2000]. Newton et al. [1998] models the blocking problem as a network design model and formulates the problem as MIP. Their decision variables are to select a block to be made and distribution of traffic among potential paths (consists of blocking arcs). They apply column generation and branch-and-price algorithms to solve

the problem. Barnhart et al. [2000] use same formulation as Newton et al. [1998] and propose Lagrangian relaxation technique to decompose the problem in two sub-problems. In spite of using novel techniques, their approach falls short of needs of a large railroad company (CSXT). Their approaches focus on determining a solution which is very close to the optimal solution, but the running times were excessive which prohibited its use. The software developed by the CSXT team using their approaches ran for several days without producing an implementable solution.

3.3 IP Formulation and Lagrangian Relaxation Approach

Using the approach of Barnhart et al. [2000] and Newton et al. [1998] as the starting point, we propose a new formulation to solve the Blocking problem. Our approaches in this section use Lagrangian relaxation in place of integer/linear programming problems to solve Lagrangian sub-problems which significantly reduces the running time of algorithms and develops special-purpose network flow based techniques to convert the intermediate solutions obtained by solving Lagrangian subproblems into very good or near optimal feasible solutions. As against the formulation in section 3.2., our formulation does not split a shipment among different paths.

3.3.1 An Integer Programming Formulation

Now we give an integer programming formulation of the blocking problem. We first present some notation.

N = a set of nodes denoting the yards where shipments start, end or are swapped. We will use the index i to denote a terminal.

A = a set of potential blocking arcs in $N \times N$, i.e., $(i, j) \in A$ if it is reasonable to send a block from i to j. We use index a to denote a potential blocking arc.

$G = (N, A)$, the potential blocking network.

$\delta^+(i)$ = the set of arcs in A emanating from i.

$\delta^-(i)$ = the set of arcs in A entering node i.

K = a set of OD shipments. We will refer to an OD shipment as a *commodity*. We will use index k to denote a commodity.

$o(k)$ = the origin of commodity $k \in K$.

$d(k)$ = the destination of commodity $k \in K$.

v_k = volume of cars to be shipped for commodity $k \in K$.

V_a = the maximum number of cars that can flow on block $a \in A$ (observe that $V_a \leq \sum_{k \in K} v_k$).

d_i = maximum volume of cars that can be handled at yard $i \in N$.

b_i = maximum number of blocks that can be made at yard $i \in N$.

m_a = cost of flow per car on block $a \in A$ (it can be a scalar multiple of the distance of block a).

h_i = the cost of handling a car at yard $i \in N$.

It will be useful to describe the size of the blocking problem for CSXT as a test case. CSXT has an underlying railroad network spanning 13,000 locations. OD shipments originate, terminate, and swap only at a subset of these locations. Further, many of these locations send or receive shipments only from one location. We can use this information to shrink many nodes into other nodes. In addition, when considering potential blocking possibilities, one need not consider blocking between every pair of locations. One can use some business rules to eliminate unnecessary blocking possibilities. We estimate that after suitable preprocessing has been done, the size of the blocking network will be reduced to about 3,000 nodes and 3,240,000 potential blocking arcs. We anticipate that there are about 80,000 distinct OD shipments that need to be sent in this network.

In the blocking problem we require that (i) each commodity $k \in K$ consisting of v_k cars must be sent along a single path in the blocking network; (ii) the total number of cars

entering node i in the blocking network be at most d_i; and (iii) the number of blocking arcs out of node i used by any path in the blocking network be at most b_i. The objective function in the blocking problem is to minimize the weighted sum of (i) the total number of car handlings, and (ii) the total number of miles traveled by cars as they travel from their origins to their respective destinations.

The blocking problem has two sets of decision variables: x_a^k and y_a. The decision variable x_a^k denotes the number of cars of commodity k flowing on arc $a \in A$, and y_a is 1 if arc $a \in A$ is used by the flow of some commodity k, and 0 otherwise. The blocking problem can be formulated as the following mathematical programming problem:

Objective Function:

$$\min \sum_{k \in K} \sum_{a \in A} m_a x_a^k + \sum_{i \in N} \sum_{k \in K} \sum_{a \in \delta^+(i)} h_i x_a^k \tag{3.1a}$$

Constraints:

$$\sum_{k \in K} x_a^k \le V_a y_a, \qquad \forall a \in A, \tag{3.1b}$$

$$\sum_{k} \sum_{a \in \delta^-(i)} x_a^k \le d_i, \qquad \forall i \in N, \tag{3.1c}$$

$$\sum_{a \in \delta^+(i)} y_a \le b_i, \qquad \forall i \in N, \tag{3.1d}$$

x^k induces a flow of v_k units along a single path in G from $o(k)$ to $d(k)$, (3.1e)

$$y = \{0, 1\}. \tag{3.1f}$$

We refer to the constraint (3.1b) as the *cardinality constraints*, and (3.1c) as the *volume constraints*. Let us now analyze the size of the problem (3.1). The dimensions of the problem at CSXT are approximately $|N|$ = 3,000, $|A|$ = 3,240,000, and $|K|$ = 80,000. The constraints on the total car volume passing through a node are not required to be

"hard" constraints and they can be violated to a small extent. It should be noted that this formulation does not allow splitting of a shipment of a given commodity along several paths. In contrast with this, the model of Barnhart et al. [2000] does allow such splitting.

3.3.2 A Bi-Level Lagrangian Relaxation Approach

It is imperative to develop some relaxation approaches for the above formulation. We propose a bi-level Lagrangian relaxation based approach to solve (3.1). For a comprehensive discussion of the Lagrangian relaxation technique, we refer the reader to the book by Ahuja, Magnanti, and Orlin [1993]. Observe that in the formulation (3.1) of the blocking problem, there is only one set of constraints (3.1b) that involve both the variables x and y; all other constraints involve either x or y. We propose to solve the blocking problem by the Lagrangian relaxation technique and relaxing the constraints (3.1b). For a given set of Lagrangian multipliers $u_a \geq 0$, we define the Lagrangian subproblem as follows:

Objective Function:

$$L(u) = \min \sum_{k \in K} \sum_{a \in A} m_a x_a^k + \sum_{i \in N} \sum_{k \in K} \sum_{a \in \delta^+(i)} h_i x_a^k + \sum_{a \in A} u_a \left(\sum_{k \in K} x_a^k - V_a y_a \right) \qquad (3.2a)$$

Constraints:

$$\sum_{k \in K} \sum_{a \in \delta^-(i)} x_a^k \leq d_i, \qquad \forall i \in N, \qquad (3.2b)$$

$$\sum_{a \in \delta^+(i)} y_a \leq b_i, \qquad \forall i \in N, \qquad (3.2c)$$

x^k induces a flow of v_k units on a single path in G from $o(k)$ to $d(k)$, (3.2d)

$y = \{0, 1\}$. (3.2e)

The Lagrangian multiplier problem is to determine the value of u for which $L(u)$ is maximum. We can use subgradient optimization, dual ascent, the volume algorithm, or

any other effective multiplier adjustment method to solve the Lagrangian multiplier problem. For an example of a relevant dual ascent procedure see Balakrishnan et al. [1989] and for the volume algorithm see Barahona and Anbil [2000].

The relaxation described above decouples the problem into two separate problems in the *x* and *y* variables respectively. We refer to the problem in *y* as the *blocking subproblem*, and the problem in *x* as the *flow subproblem*.

3.3.3 Solving the Blocking Subproblem

The blocking subproblem (BSP) is as follows:

Objective Function:

$$z_{BSP} = \min -\sum_{a \in A} V_a u_a y_a \tag{3.3a}$$

Constraints:

$$\sum_{a \in \delta^+(i)} y_a \le b_i, \quad \forall i \in N, \tag{3.3b}$$

$$y = \{0, 1\}. \tag{3.3c}$$

Observe that BSP further decomposes for each $i \in N$. The decomposition for each $i \in N$, denoted by (BSP_i), is as follows:

Objective Function:

$$z_{BSP(i)} = \min \sum_{a \in \delta^+(i)} \left(-V_a u_a\right) y_a \tag{3.4a}$$

Constraints:

$$\sum_{a \in \delta^+(i)} y_a \le b_i, \tag{3.4b}$$

$$y = \{0, 1\}. \tag{3.4c}$$

The BSP_i (3.4) can be solved by the obvious greedy method, i.e., sort the arcs $a \in \delta^+(i)$ in the decreasing order of $V_a u_a$ and choose $y_a = 1$ for the first b_i arcs in $\delta^+(i)$ and $y_a = 0$ for the rest of the arcs.

3.3.4 Solving the Flow Subproblem

We now discuss how to solve the flow subproblem, the problem in *x*. The flow subproblem (FSP) is given below.

Objective Function:

$$z_{FSP} = \min \sum_{k \in K} \sum_{a \in A} (m_a + u_a) x_a^k + \sum_{i \in N} \sum_{k \in K} \sum_{a \in \delta^+(i)} h_i x_a^k \tag{3.5a}$$

Constraints:

$$\sum_{k \in K} \sum_{a \in \delta^-(i)} x_a^k \le d_i, \qquad \forall i \in N, \tag{3.5b}$$

x^k induces a flow of v_k units on a single path in G from $o(k)$ to $d(k)$. (3.5c)

It can be shown that the flow subproblem is a 0-1 multicommodity flow problem, where the flows of different commodities are linked by the volume constraint, and the flows must induce single OD paths. This is a hard combinatorial optimization problem. We next use the fact that the volume constraints are not binding constraints since in practice it is difficult to determine d_i, the maximum volume of cars that can pass through the node *i*. Hence we suggest relaxing these constraints too. If we take $p_i \ge 0$ to be the (Lagrangian) multiplier for flow into node *i*, then we get the relaxed flow subproblem, FSP(*p*), given as follows, where $s(a)$ denotes the tail node of arc *a* and $t(a)$ denotes its head node:

Objective Function:

$$z_{FSP}(p) = -\sum_{i \in N} p_i d_i + \min \sum_{k \in K} \sum_{a \in A} (m_a + h_{s(a)} + u_a + p_{t(a)}) x_a^k \tag{3.6a}$$

Constraints:

x^k induces a flow of v_k units on one path from $o(k)$ to $d(k)$. (3.6b)

The problem (3.6) decomposes for each commodity k and the solution for each commodity k would be simply a flow of v_k units along the shortest path in G from $o(k)$ to $d(k)$ with arc lengths given by $m_a + h_{s(a)} + u_a + p_{t(a)}$ for arc a. Observe that these arc lengths are independent of the commodity k. Thus, using these arc lengths, there are three alternative approaches that might be considered for solving the FSP(p):

1. For each commodity k, determine a shortest path from $o(k)$ to $d(k)$, and send the flow v_k along this path.
2. Solve for each origin node $o \in N$ the problem of finding a shortest path tree rooted at node o, in which case we have simultaneously solved the shortest path problem for each commodity k with $o(k) = o$.
3. Solve an all-pairs shortest path problem in G, which simultaneously determines the shortest path distances for all commodities from their origins to their destinations.

While the approaches in (3.2) and (3.3) might look attractive since they determine shortest path distances for several commodities simultaneously, a careful analysis of the computational effort likely to be incurred in each case shows that the first is likely to be the most efficient. The computational effort required in case (3.3) is of the order of $|N||A|$. In case (3.2), we are likely to consider the whole blocking network G for each shortest path tree calculation and each of these requires work of the order of $|A|$ (if we solve the shortest path problem using the label-correcting algorithm), and we have to perform about $|N|$ of these calculations (one per node); again the total work required is of the order of $|N||A|$. Since for the blocking problem at CSXT, $|N| = 3{,}000$ and $|A| = 3{,}240{,}000$, the work required is of the order of 2×10^{13}.

By contrast, in case (3.1), for each commodity we could reasonably restrict ourselves to consider only a few alternative physical routes. If for each commodity, we

consider R physical routes and assume that all blocking subpaths are possible for each route, and also assume that the commodity will use no more than B blocking arcs to reach its destination, then we can solve, for each commodity, R separate shortest path problems, each on an acyclic network containing $B(B+1)/2$ arcs. This will require work of the order of $|K|RB(B+1)/2$, where $|K|$ is the total number of commodities. If we take $R = 10$, $B = 5$, and $|K| = 80{,}000$, this works out to be equal to 10^7, which is an order of magnitude less than that required for the other two approaches.

3.3.5 Connectivity in the Blocking Subproblem

We have described above how to solve the blocking subproblem efficiently. It has been observed by Barnhart et al. [2000]. that the lower bounds provided by (3.4) are quite weak. To strengthen the bounds for the Lagrangian subproblem (3.4), we may impose additional constraints that y variables must satisfy. For example, y variables must induce a graph in which there is a path between every OD pair. This is not quite the same thing as saying that the graph must be connected, but is close to it. Barnhart et al. [2000] observed that this constraints tightens the relaxed problems and yields substantially tighter bounds. The blocking subproblem BSP with the additional connectivity constraints is as follows:

Objective Function:

$$\min -\sum_{a \in A} V_a u_a y_a \tag{3.7a}$$

Constraints:

$$\sum_{a \in \delta^+(i)} y_a \le b_i, \quad \forall i \in N, \tag{3.7b}$$

$$\sum_{a \in \delta^+(S)} y_a \ge 1, \quad \forall S \subset N, \text{ such that there is a } k \text{ with } o(k) \in S, d(k) \notin S, \tag{3.7c}$$

$$y = \{0, 1\}. \tag{3.7d}$$

In the above formulation, $\delta^+(S)$ denotes the set of all arcs a $\in$ A with the tail node s(a) $\in$ S and the head node t(a) $\in$ N\S. Let Ω denote the set of all sets S $\subset$ N that separate the OD pair of some commodity, i.e., such that there exists a commodity k with o(k)$\in$ S and d(k)$\in$ N\S. To solve (3.7), we propose to use Lagrangian relaxation. We introduce Lagrange multipliers rS $\geq$ 0 for each S$\in$ Ω, and solve the following problem which we refer to as BSP(r):

Objective Function:

$$z_{BSP}(r) = \sum_{S \in \mathbf{\Omega}} r_S + \min \{-\sum_{a \in A} V_a u_a y_a - \sum_{S \in \mathbf{\Omega}} r_S \sum_{a \in \delta^+(S)} y_a\} \tag{3.8a}$$

Constraints:

$$\sum_{a \in \delta^+(i)} y_a \leq b_i, \quad \forall i \in N, \tag{3.8b}$$

$$\text{y} = \{0, 1\}. \tag{3.8c}$$

Clearly, (3.8) also decomposes for each node $i \in N$ and, for a given r, can be solved greedily solving each of the following subproblems which we refer to as $BSP_i(r)$:

Objective Function:

$$z^i_{BSP}(r) = \max \sum_{a \in \delta^+(S)} (V_a u_a + \sum_{S: a \in \delta^+(S)} r_S) y_a \tag{3.9a}$$

Constraints:

$$\sum_{a \in \delta^+(i)} y_a \leq b_i, \tag{3.9b}$$

$$y_a \in \{0,1\} \tag{3.9c}$$

and observing that

$$z_{BSP}(r) = \sum_{S \in \mathbf{\Omega}} r_S - \sum_{i \in N} z^i_{BSP}(r) . \tag{3.10}$$

For any $r \geq 0$ solving (3.9) we obtain $z_{BSP}(r)$ giving a valid lower bound on z_{BSP}. We will seek to adjust r so that this bound improves, which corresponds to encouraging the y variables resulting to satisfy the connectivity constraints. We propose to adjust the r values using subgradient optimization, dual ascent, or some other appropriate method. In order to adjust r, and find (truncated) subgradients of $z_{BSP}(r)$, we will need to determine violated connectivity constraints, i.e., given a solution, y^*, for BSP(r), we seek the set(s) $S \in \mathbf{\Omega}$ such that $\sum_{a \in \delta^+(S)} y^*_a < 1$. (Note that y^* is a 0,1 vector.) Although several approaches to this problem are possible, we expect that the most efficient approach will be to seek a path from $o(k)$ to $d(k)$ for each commodity k in the network induced by y^*. Arguments for this are similar to those given in Section 3.3.4, although it is possible that if one fails to find a path from among the R physical routes considered, there is still one in the wider network, and a more expensive shortest path calculation would have to be made. An alternative would be to modify the connectivity constraints to force a path to exist amongst the physical routes considered. Let $\boldsymbol{R}^k$ denote the set of all physical routes considered for commodity k and let $(N^{k,\sigma}, A^{k,\sigma})$ denote the subnetwork of the blocking network induced by physical route $\sigma \in \boldsymbol{R}^k$ for commodity k. For $S \subset N$, define $\delta^+_{k,\sigma}(S) \subseteq A^{k,\sigma}\pi$ to be the set of arcs $(i, j) \in A^{k,\sigma}$ with $i \in S$ and $j \notin S$. We could then rewrite the connectivity requirements as:

$$\sum_{a \in \Delta^+_k(S)} y_a \geq 1, \qquad \forall k \in K, \forall S \in \mathbf{\Omega}, o(k) \in S, d(k) \notin S \tag{3.11}$$

where

$$\Delta_k^+(S) = \bigcup_{\sigma \in \boldsymbol{R}^k} \delta_{k,\sigma}^+(S). \tag{3.12}$$

The procedure for finding a violated constraint of this type could then be easily integrated with the heuristics to determine good feasible solutions. The only possible disadvantage of this approach is that the dimension of the multipliers *r* will increase, as we need a multiplier r_S^k for each *k* and *S* pair. However, since the dimension of (3.7) was already exponentially large, so it is hard to imagine that this would be a serious disadvantage.

3.3.6 Solution Approach Summary

We summarize the discussion so far. We propose a two-level Lagrangian relaxation approach. At the top level, the constraints linking the *x* and *y* variables are relaxed. This top-level Lagrangian multiplier problem will be solved by subgradient optimization or some other appropriate method. At the second level, we have the BSP with connectivity constraints, and the FSP to solve. The former can be solved by Lagrangian relaxation of the connectivity constraints, leading to decomposed very rapidly solved subproblems. It is not expected that this Lagrangian dual can be solved to optimality: instead some effort should be invested in achieving connectivity, but not at the expense of speed. The FSP can be solved by Lagrangian relaxation of the volume constraints, and the resulting subproblem solved by solving shortest path problems for each commodity. Since the volume constraints are not likely to be binding, and are not hard constraints, we would expect to only adjust the Lagrange multipliers at each iteration only a small number of times. Since at both the levels the Lagrangian relaxations provide lower bounds, we will maintain a valid lower bound at all times.

Although our approach as proposed above have many merits above the approaches of Barnhart et al.[2000] and Newton et al. [1998], once implemented, did not give satisfactory performance on the fronts of quality of lower bounds and time taken. We observed that the lower bounds obtained by the lagrangian relaxation approach were too loose and it was taking hours of computing time. The reason for unsatisfactory performance might be attributed to the size of the problem. As railroad companies are more concerned in solving the problem to near-optimality in less time and Lagrangian relaxation based approaches were not working satisfactorily, we thought it more appropriate to develop heuristics to solve the problem. In following sections, we describe heuristics to solve blocking problem using very large scale neighborhood search technique.

3.4 Very Large Scale Neighborhood Search based Heuristics

Motivated by the success on applying VLSN search for airline scheduling and locomotive scheduling problems, we developed a VLSN search algorithm for railroad blocking problem. VLSN search algorithm search improving solutions in very large neighborhood and blocking problem is also very large scale optimization problem. Therefore, VLSN search algorithm for blocking problem is required to be very time and space efficient. We have used network-flow based optimization techniques to find local optimal solution for blocking problem. This algorithm is highly scalable, robust and produces high quality solution in reasonable computational time. Our algorithm is very flexible and can easily number of additional constraints as outlined in section 3.5.

The railroad blocking problem can be modeled as network design problem as we need to construct a blocking network and to route shipments on the resulting network. With respect to the blocking network, we can state blocking problem as constructing a

blocking network honoring the blocking capacity constraints and routing commodities on the blocking network honoring car handling capacity constraints. In this section we describe a very large scale neighborhood (VLSN) based local search heuristic, which produces a local optimal solution for the blocking problem with respect to network design formulation. The neighborhood search heuristic uses network flow based techniques to implicitly enumerate very large number of neighbors. Figure 3-2 describes our VLSN search algorithm for railroad blocking problem.

```
algorithm VLSN-Blocking;
begin
    construct an initial feasible solution x of the blocking problem;
    while the solution is not locally optimal do
        begin {one pass}
            consider each node in the blocking network one by one and re-optimize
            the blocks emanating from this node;
        end; {one pass}
    end;
end;
```

Figure 3-2. The VLSN search algorithm for the blocking problem

This algorithm is a neighborhood search algorithm. It starts with a feasible solution of the blocking problem and iteratively improves the current solution by replacing it by its neighbor. This algorithm has two important subroutines: to construct the initial feasible solution; and to re-optimize the blocking arcs emanating from a node. In the rest of this section, we will first describe the construction heuristic to generate initial solution and then neighborhood search heuristics to get local optimal solution.

3.4.1 Construction Heuristic

A construction heuristic produces a feasible solution to the optimization problem from scratch by assigning values to one or more decision variables at a time. The railroad blocking problem is also a decision problem, where we select the potential blocks to

construct the blocking network and route the commodities on the resulting blocking network without violating the blocking capacity constraints. Therefore, the construction heuristic should be able to choose blocks to be made out of all potential blocks and to route the commodities. As the number of blocks to be made at a node is restricted and large number of commodities is to be routed on these blocks, finding a feasible solution for the blocking problem is itself a difficult problem. Therefore, an efficient algorithm is needed which can produce a feasible solution and at the same time should produce a good quality solution.

As commodities can be classified at any yard, a feasible path for any commodity can pass through any number of classification yards. Therefore, to ensure the generation of a feasible solution, a directed cycle (Hamiltonian Cycle) containing all yards is constructed in which starting from any yard, it will visit all the yards only once and returns back to the starting yard. To route any commodity from its origin to its destination, we first connect the origin of the commodity to nearest yard and connect the destination from nearest yard having available blocking capacity. As there exists a path from the yard, to which origin is connected, to the yard from which destination is connected, this procedure always ensure a feasible solution to the blocking problem. The problem of finding a Hamiltonian Cycle (HC) in a graph is a well known *NP*-Complete problem and we have used heuristic to find HC.

To get a good quality feasible solution we use nearest neighbor approach to connect origin to the yard and to connect destination from a yard. We also use greedy approach to make direct blocks for a commodity from its origin to its destination, if blocking capacity is available at the origin or at any intermediate node at any stage. Before making a new

block for a commodity, we first check whether a blocking path with better cost as compared to cost of the path if new block is created exists. If found any such path, we do not make new block and assign the commodity to blocking path found. Our construction heuristic, essentially perform following steps.

Step 0: Find a HC connecting all the classification yards.

Step 1: If all the commodities are routed, stop; otherwise go to Step 2.

Step 2: Find the best blocking path in the existing network for a commodity not having any assigned blocking path. If found any such path, compare the cost if a new blocking path is constructed (using nearest neighbor approach) without violating the blocking capacity. Assign the better path to the commodity and update the blocking capacity of nodes in the path.

We tested many variations of above algorithms and observed that the final solution obtained from our local search heuristic is quite insensitive to the quality of initial solution. We next describe our neighborhood search heuristic.

3.4.2 Neighborhood Search Algorithms

In the starting feasible solutions, we get the sets of blocks made at each node in the blocking network and blocking paths for each of the commodity. With respect to this solution, we define neighborhood as all the solutions obtained by changing blocking policy at a node and keeping blocks made at all other nodes fixed. The algorithm selects nodes in sequence, drop all the blocks currently made at the node selected, re-route all the commodities which currently assigned to blocking path containing dropped blocks, and re-optimizes the blocking policy at the node selected. If we could not find any alternate blocking path for any commodity required to be re-routed, we assign very large cost of transportation to the commodity.

Re-optimizing blocking policy at a yard (say p) is a large-size optimization problem. This problem can be stated as selecting a set of blocks out of all possible blocks

at yard *p*, honoring the blocking capacity constraint at *p*, and re-routing commodities such that improvement in objective function is maximum. If we use c_{pi}^{k} to denote the benefit of routing commodity *k* to use potential blocking arc (*p, i*), this problem can be formulated as Integer Programming problem as given below.

Objective Function:

$$\max \sum_{k \in K} \sum_{i \in N \setminus \{p\}} c_{pi}^{k} x_{pi}^{k} \tag{3.13a}$$

Constraints:

$$\sum_{i \in N \setminus \{p\}} y_{pi} \le b_p , \tag{3.13b}$$

$$\sum_{k \in K / \{p\}} x_{pi}^{k} \le | K | \, y_{pi}, \quad \forall i \in N , \tag{3.13c}$$

$$y_{pi} \in \{0,1\}, \quad \forall i \in N , \tag{3.13d}$$

$$x_{pi}^{k} \in \{0,1\}, \quad \forall k \in K, \forall i \in N . \tag{3.13e}$$

In above formulation, binary decision variable y_{pi} takes value 1 if blocking arc (*p, i*) will be built, and 0 otherwise. Binary decision variable x_{pi}^{k} takes value 1 if commodity *k* will be routed using blocking arc (*p, i*). The objective function (3.13a) maximizes the total benefit of building new blocks at yard *p*. Constraint (3.13b) enforces that the number of blocks built at yard *p* will not exceed the blocking capacity of yard *p*. A commodity can be routed using blocking arc (*p, i*) only when the blocking arc will be built, which is stated in constraints (3.13c).

Solving above problem optimally is very difficult problem. To better understand the difficulty of the problem, we briefly analyze here the number of binary decision

variables. For a typical U.S. railroad, such as CSX Transportation, blocking capacity of a node is around 10-50, and there are around 600 classification yards and 2400 destinations to which block can be made. Hence the optimization algorithm should be able to choose best 20 blocks out of around 3,000 possible blocks. Once blocks are made, we need re-route the commodities. As there are around 80,000 commodities, needless to say optimally re-routing these commodities is very difficult problem. With the data provided by the railroads, we could not find optimal solution for this problem even after spending hours of computer time. We re-optimize the blocking policy at every node in each pass (explained later) and there are around 3000 nodes in the network, even 1 second of computer time per node results in hours of computer time for railroad problem, which is not acceptable by the railroads.

We have developed an algorithm for re-optimizing blocking policy at a node (say *p*) which uses following maximum savings scheme.

- For each potential blocking arc at the node *p*, compute the savings in total cost if that blocking arc is built and commodities are re-routed keeping the newly built blocking arc in consideration.
- Select the blocking arc with the maximum savings in transportation cost and build it.
- Re-route some commodities to pass through newly built blocking arc.
- Repeat the process until no more saving is possible by building new block is possible or the no more blocking capacity is available.

In above scheme, most time consuming task is to compute the saving by creating a new blocking arc. We use a scheme based on the shortest path algorithm to compute the savings. Our scheme very efficiently calculates the shortest blocking path between two nodes whenever a new arc is added to or deleted from the blocking network. With the use of efficient data structure our algorithm takes around 0.01 computer second for re-

optimizing blocking policy at a node as against hours of computer time taken by commercial available optimization software.

In above algorithm, we re-optimize blocking policy at the nodes in increasing sequence of node sequence (say there are n nodes indexed 1, 2, …, *n*). Therefore, While re-optimizing blocking policy at node *i* we assume that some blocks are already made at node *i*+1, *i*+2, …, *n*. However, when the blocking policy at nodes *i*+1, *i*+2, …, *n* are re-optimized (and changed), the blocks made at node *i* may not be optimal and there may exists further scope of improvement using maximum saving scheme. To minimize this deficiency of neighborhood search, we perform passes over the nodes in the blocking network and re-optimize them. In one pass of the algorithm, starting with the solution obtained in preceding pass, we re-optimize the blockings at all the nodes in network. In first pass of the algorithm starting feasible solution is that produced by construction algorithm. The computational experiments on the data provided by the railroads indicate that there are substantial improvements in the solution in first 3-4 passes and solutions converge to the local optimal solution in 10 passes.

3.5 Additional Constraints

For the practical and day to day implementation of the solution obtained by above algorithms, railroad companies also have many operational and functional constraints. These constraints are mainly imposed for historical, customer satisfaction or trade union related reasons. To make the model practical, we have also incorporated features in VLSN search algorithms to meet following additional constraints.

- Generally train scheduling follows the blocking plan; however, railroad companies may want a blocking plan compatible with existing train schedule with the objective of minimum train swaps. Our algorithm meets this requirement by minimizing the number of blocks made on the route where no train exists.

- Decision makers may want to specify some blocks which must be made irrespective of cost involved. This may be due to historic reason or commitment to key customers. Similarly, railroad company may not want some blocks, irrespective of lucrative-ness of these blocks. Our algorithm honors this requirement by always making must blocks and by not making the unwanted blocks.

- There may be some blocks which are desirable to make, i.e., railroad company may show preference for some of the blocks. Our algorithms meet this requirement by not making these blocks in the case when the cost of making these blocks affects the objective function very adversely.

- The railroad company may specify a list of commodities which must flow on pre-specified blocking paths. This may again due to historic reason or commitment to customers. This implies that there must be a blocking path for pre-specified path and we always assign given commodities on such paths.

- To check the viability or to make incremental change, railroad company may want to change blocking plan only at a set of nodes in place of overhauling complete blocking plan. This may be required to gain confidence and properly managing (implementing) the change.

- Railroad company may specify the fraction of blocking, which can be generated by the algorithm. For example, decision makers may say that they want to change blocking plan only for 10 % of the existing blocks. This is again required if railroad companies want to make incremental change in the blocking plan.

- The shipment routes as generated by blocking plan must meet the plate clearance and weight limit of the physical network.

- Decision makers may want decision support systems to answer questions like when to add additional blocking or additional car handling capacities to have maximum impact on the solution quality.

3.6 Computational Results

In this section we present the performance of our VLSN neighborhood search algorithms for the real life data provided by two major U.S. railroad companies, CSX Transportation and BNSF Railways. We will first describe the problem characteristics for these railroads and then present the performance of our algorithms. All the performance data are normalized and do not represent the actual performance of railroads.

CSX Transportation has a large railway network consisting of around 3,000 nodes: around 600 are classification yards. CSX Transportation moves over 500,000 general merchandize shipments annually from their origins to their destinations. The total number of cars in these shipments is around 6 million. To move these shipments, total number of blocks can be made at the nodes is around 6000.

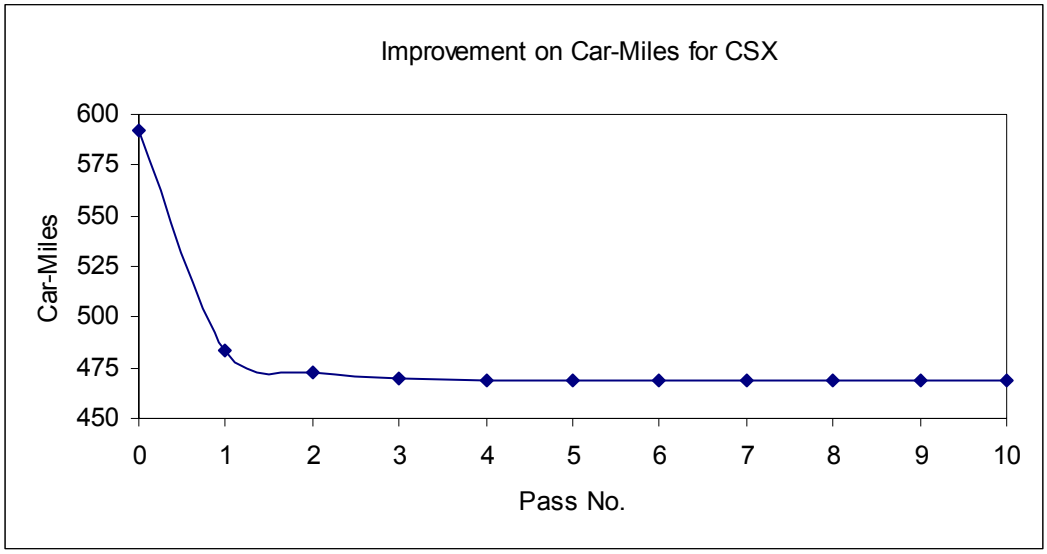
BNSF Railways is another major railroad in freight transportation. The physical railroad network of BNSF consists of around 1,200 nodes, including 300 classification yards. BNSF moves over 200,000 general merchandize shipments (containing around 3 million cars) annually from their origins to destinations. The restriction on total number of blocks can be made is around 1500.

We have measured the performance of our algorithms against the performance of these railroads with respect to two objectives: average car miles and average number of intermediate handlings. Average car miles indicate the average distance in miles traveled by each car and average number of intermediate handlings indicates the number of classification done for each car at the yards in between origin and destination. Table 3-2 gives the comparative analysis of our solution. It can be observed that our solution improves both the car miles and intermediate handlings substantially. Table 3-2 also gives the time taken in solving the blocking problem on a Pentium IV PC. This time is very reasonable and quite acceptable by the railroads.

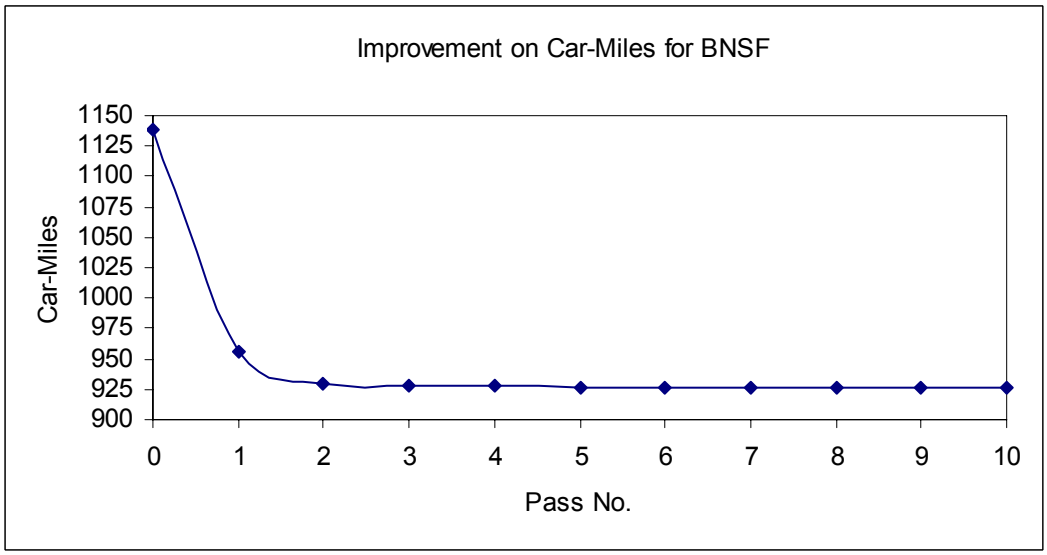
Table 3-2. Comparison of our solutions with the solutions used by railroads

	CSX Transportaion		BNSF Railways	
	Their Solution	Our Solution	Their Solution	Our Solution
Average Car-Miles	487	469	932	926
Average Intermediate Car-Handlings	1.5	0.95	1.08	0.83
Computational Time	N/A	15 minutes	N/A	5 minutes

To understand the algorithmic behavior, we did some additional analysis as presented in Figure 3-3 and 3-4. These figures give the incremental improvement in the average car-miles and intermediate handlings as we perform passes. Pass number 0 represents the starting solution value and pass number 10 gives the final solution value. It can be observed that there is substantial improvement in performance in first 3-4 passes and solution converges in around 10 passes.

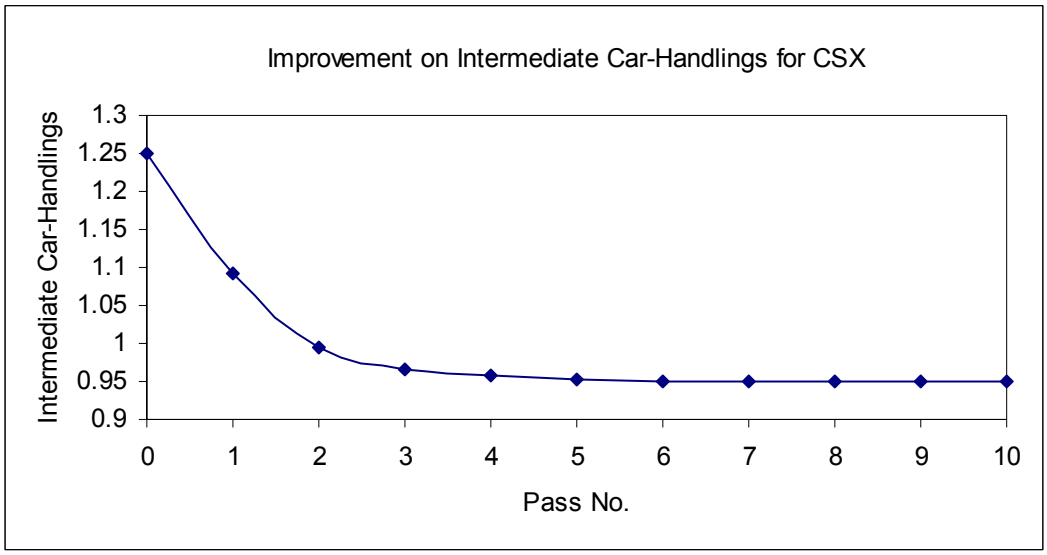


(a)

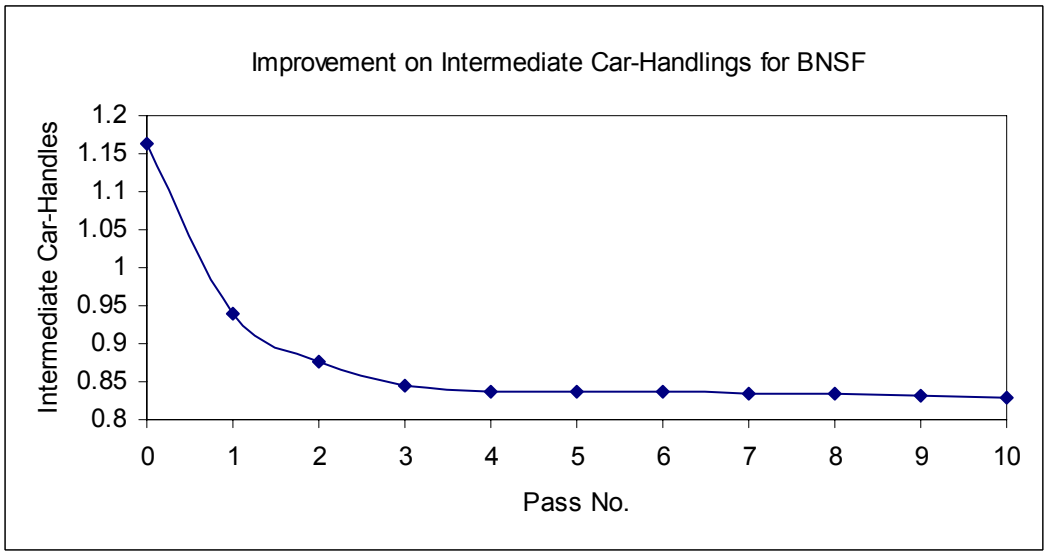


(b)

Figure 3-3. Improvement in average car-miles. (a) CSX Transportation, (b) BNSF Railways



(a)

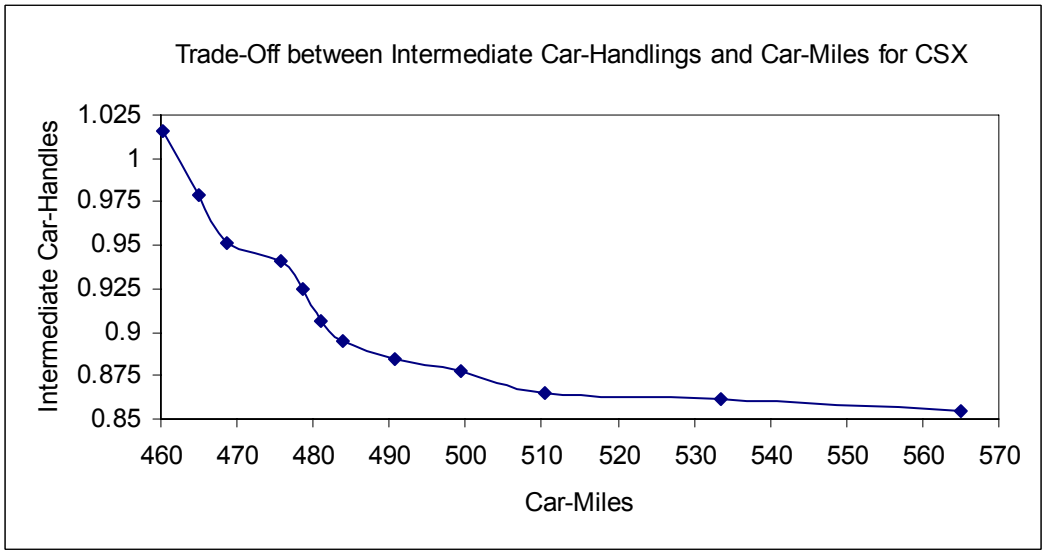


(b)

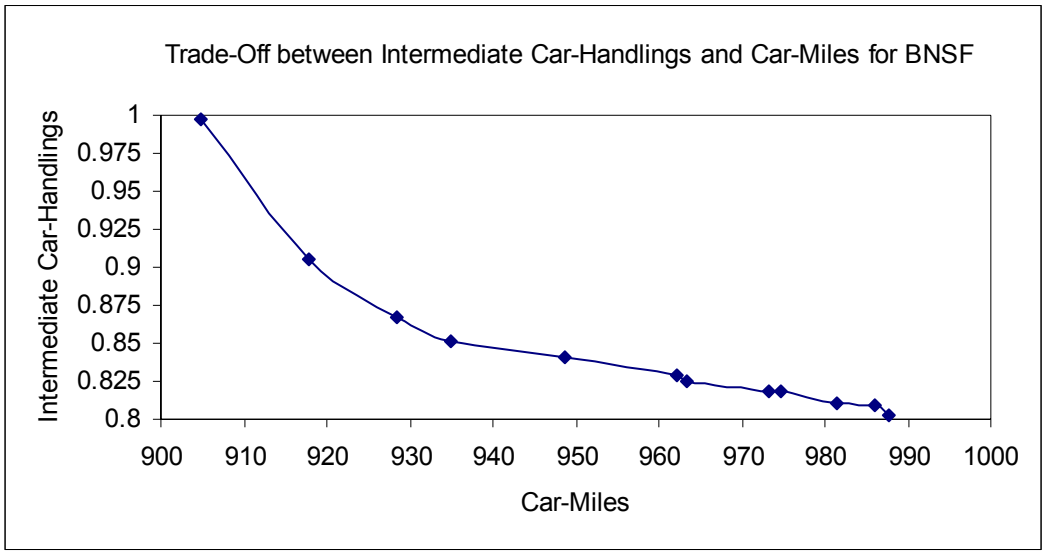
Figure 3-4. Improvement in intermediate car-handlings. (a) CSX Transportation, (b) BNSF Railways.

The objective function of railroad blocking problem consists of two cost terms: car miles, and car intermediate handlings. We can tradeoff between two terms by assigning different relative weights, i.e., we can specify preference for more improvement in car miles or in intermediate handlings. Figure 3-5 presents the tradeoff curve for these two

cost terms. The decision maker can select the most suitable point on this curve and use the solution corresponding to this point.



(a)



(b)

Figure 3-5. Tradeoff curve between of intermediate car-handlings and car-miles. (a) CSX Transportation, (b) BNSF Railways

3.7 Conclusions

In this chapter, we have formulated and described the algorithmic approaches to solve one of the most important railroad scheduling problems called blocking problem.

For a railroad this is a large scale and very difficult optimization problem. Approaches to solve IP formulation of the blocking problem were unable to produce solution in reasonable amount of computer time. We developed a network flow based very large scale neighborhood search heuristics for the problem. Our algorithms not only produce very good quality solution in few minutes of computer time, but also give flexibility in incorporating many important additional constraints. The algorithms developed are only applied to general merchandize type of shipment and further work is required to be done to develop an integrated model to solve the blocking problem with multi-type shipments.

CHAPTER 4
SOLVING THE MULTI-CRITIERIA COMBINED THROUGH AND FLEET ASSIGNMENT PROBLEM

4.1 Introduction

The airline industry has been a pioneer in using IE/OR techniques to solve complex business problems related to the schedule planning of the airline. Given a flight schedule, an airline's schedule planning group needs to decide the itinerary of each aircraft and each crewmember so that the total revenue minus the total operating costs is maximum and all the operational constraints are satisfied. The quality of the schedule is also measured in terms of other attributes such as schedule reliability during operations. The entire planning problem is too large to be solved to optimality as a single optimization problem using present day technology. Hence, it is typically divided into four stages (see, for example, Barnhart and Talluri [1997] and Gopalan and Talluri [1998]): (i) fleet assignment; (ii) through assignment; (iii) maintenance routing; and (iv) crew scheduling. These problems are solved sequentially where the optimal solution of one problem becomes the input for the following problem. There has been significant effort spent in modeling and solving these individual problems using advanced optimization models. The economies of scale at a large airline like United Airlines are such that a relatively minor improvement in contribution results in considerable improvement in the bottom line. As a result, airlines have benefited immensely from the advances in modeling these problems

The next frontier in the optimization of schedule planning is in solving an integrated optimization problem that will consider the entire planning problem mentioned above and include other downstream issues that affect the overall schedule quality. Basically, the planning problem is a multi-criteria optimization problem, i.e., there are many objectives that have different metrics, different priorities and different constraints. A sequential approach to solve such problems has a major drawback in that the solution at each stage does not take into account the considerations of subsequent stages. This results in overall suboptimal solutions. For example, if a fleet assignment is performed without considerations of optimizing crew scheduling, it becomes an arbitrary input for the crew scheduling process. On the other hand, if the fleet assignment incorporates crew issues, then it is likely to provide a better starting point to the crew scheduling optimization, resulting in overall economic benefits for the airline.

The airlines are making a lot of effort to develop models to solve such integrated optimization problems for schedule planning. The objective is to break down the functional silos that exist in order to manage the complexity of the planning process by using advances in optimization technology and computing power. At United Airlines, there are two distinct strategies being pursued. The first strategy is to develop explicit joint optimization models with combined objective functions, combined set of constraints and combined data. Typically, these joint models are too large to be solved to optimality or near-optimality, suggesting that heuristics may be needed. Moreover, some downstream criteria cannot be represented easily in a form consistent with explicitly modeling the problem. For example, airline reliability is an important criterion in schedule planning that is related to the schedule structure, but it is very hard to model it

in terms of a typical optimization problem. The second strategy exploits the nature of the planning problem. Instead of one optimal solution, there are typically many solutions that are close to the optimal in terms of contribution. However, these solutions can have very distinct characteristics on other criteria such as crew required, potential for through flights, schedule reliability, ground manpower requirements, *etc*. This implies that intelligent search techniques, when coupled with advanced optimization modeling, hold a lot of promise in solving multi-criteria schedule planning problems. As a result, United Airlines initiated collaboration with MIT and University of Florida to explore solution techniques that exploited the nature of the overall schedule planning problem. The strategy was to build upon the foundation that already existed for modeling the individual stages and develop a robust and generic methodology that could be easily scaled up for other downstream criteria.

Ahuja et al. [2001c] has developed an integrated model for solving the combined through and fleet assignment (ctFAM) model, which itself integrates the following two models: (i) the fleet assignment model (FAM), and (ii) the through assignment model (TAM). The computational result reported gives considerable improvement over the solution obtained by solving fleet assignment and through assignment problems sequentially. We next briefly describe these three models.

Fleet Assignment Model (FAM)**:** In FAM, planes of different fleet type are assigned to flight legs to minimize the assignment cost and subject to the following three types of constraints: (i) covering constraints: each flight leg must be assigned exactly one plane; (ii) flow balance constraints: for each fleet type, the number of planes landing at a city must be equal to the number of planes taking off from the city; and (iii) fleet size

constraint: for each fleet type, the number of planes used must not exceed the number of planes available. Abara [1989] and Hane et al. [1995] give an MIP formulation for FAM. Subsequently, Clarke et al. [1996] and Subramanium et al. [1994] provide extensions to incorporate additional operational constraints related to maintenance and crew scheduling. Barnhart et al. [1998] describes a column generation approach to solve the integrated fleet assignment, through assignment, and maintenance routing problem. Their approach was too time intensive for the entire planning problem, but was suitable for smaller problems such as international flight schedules. An alternative model for fleet assignment based on origin-destination demands was studied by Kniker et al. [2000].

Through Assignment Model (TAM): A through connection is a connection between an inbound flight leg and an outbound flight leg at a station which ensures that the same plane flies both legs. Both flights legs in a through connection get the same flight number in the airline's flight schedule. Since a through connection allows passengers to remain onboard instead of changing gates at busy airports, passengers are willing to pay a premium for such connections; this premium is termed as the through benefit. TAM takes as an input a list of candidate pairs of flight legs that can make through connections with corresponding through benefits, and identifies a set of most profitable through connections. Observe that we can make through connections only between flights flown by the same fleet type; hence the fleet assignment limits the possible through connections. In the current implementations, TAM takes as an input the fleet assignment, identifies inbound and outbound flights at each city flown by the same fleet type, and determines through connections (that must be a subset of the candidate pairs) to maximize the through benefit. This problem can be solved as a bipartite matching

problem. However, in practice the solution must satisfy some additional constraints, which yields a constrained bipartite assignment problem that can be solved using MIP techniques. We refer the reader to the papers by Bard and Hopperstad [1987], Barnhart et al. [1998], Gopalan and Talluri [1998], and Jarrah and Reeb [1997] for additional details on the TAM.

The Combined Through and Fleet Assignment Model (ctFAM): The through assignment depends on the fleet assignment in that a through connection requires that both its flight legs have the same fleet type. In the current systems, FAM does not take into account the through benefits, and may yield fleet assignments with limited through assignment possibilities. TAM cannot change the fleeting in order to get a better through assignment. In our model, ctFAM solves the integrated model and simultaneously determines fleet assignments and through connections. The integrated model offers opportunities to obtain better solutions compared to the current sequential approach.

In this chapter, we shall describe the extension of the model developed by Ahuja et al. [2001c] by adding two additional optimization criteria (i) ground manpower planning, and (ii) crew planning. In our multi-criteria approach we exploit the observation that there are many solutions of fleet assignment problem, which are close to optimal objective function value. We shall describe the cost drivers and functions that we use for the crew scheduling and ground manpower scheduling stages of the overall problem. We consider two multi-criteria problems. In the first problem, we consider the ground manpower scheduling as the second objective along with primary objective of fleet and through assignment. In the second problem, we consider the crew scheduling as the second objective. Using the neighborhood search approach, we provide airline planners

with several good solutions that may be evaluated for later planning stages. We restrict ourselves to bi-criteria problems, however, algorithms developed in this chapter can be readily generalized to ctFAM problems with more than two criteria.

We now present a brief overview of our neighborhood search algorithm for the multi-criteria ctFAM. Neighborhood search algorithms are typically quite efficient and scalable. Often, we can solve problems with only a linear increase (or a small polynomial increase) in the computation time. Neighborhood search algorithms are often flexible enough to incorporate other constraints that are difficult to model through linear constraints. As in Ahuja et al. [2001c], we define neighbors of a given solution by performing "*A*-*B* swaps" for two specified fleet types *A* and *B*. An *A*-*B* swap consists of changing fleet types of some legs from *A* to *B* and of some legs from *B* to *A*. The swap also modifies connections so that all constraints remain satisfied. In each iteration, the neighborhood search algorithm selects any two fleet types, which we label as *A* and *B*, and performs a profitable *A*-*B* swap. In principle, one can locate profitable *A*-*B* swaps by solving the MIP formulation as restricted to flights of type *A* and flights of type *B*. However, the solution times for this approach were impracticably large for use in this heuristic, and so we relied upon a method using "*A*-*B* Improvement Graphs" to obtain profitable *A*-*B* swaps quickly in practice. (This is based on the technique described in Ahuja et al. [2001c] which, in turn, is based on the approach of Talluri [1996].) Our search for *A*-*B* swaps reduced to a search problem in the *A*-*B* improvement graph. Our model involves multiple criteria; accordingly, we maintain a subset of solutions that are non-dominated, that is, no solution dominates another on all criteria. When the algorithm

terminates, we output the non-dominated solutions that have "satisfactory" values for all the objectives, as specified by a user.

The rest of this chapter is organized as follows. In Section 4.2, we summarize the results in Ahuja et al. [2001c], which is the starting point for our research. We describe the multi-criteria ctFAM and details of the additional criteria (estimated cost functions for manpower and crew criteria) in Section 4.3. We describe our neighborhood search algorithms for the multi-criteria ctFAM in Section 4.4. We provide computational results in Section 4.5. Section 4.6 summarizes our contributions and gives future research directions.

4.2 Background

In this section, we summarize the results from Ahuja et al. [2001c]. We first briefly present their integer programming formulation of the ctFAM followed by their neighborhood search approach for the ctFAM.

4.2.1 Integer Programming Formulation for ctFAM

The ctFAM can be formulated as an integer multicommodity flow problem on a network called the *connection network*. We denote the set of flight legs by L, the set of fleet types by F, the set of stations by S, and the set of candidate through connections by T. For each flight leg $i \in L$, we let *arr-time*(i) and *dep-time*(i) denote the arrival and departure times of the flight, respectively, and *arr-city*(i) and *dep-city*(i) represent the arrival and departure cities of the flight, respectively. For each fleet type $f \in F$, *size*(f) is the number of planes of type f. For each flight leg $i \in L$ and $f \in F$, the contribution from assigning fleet type f to leg i is c_i^f. We denote by d_{ij}^f the through contribution of the connection between flight leg i and flight leg j (assuming *arr-city*(i) = *dep-city*(j)) when

both legs have the same fleet type f. Let *count-time* be an instant on the 24-hour time scale when there is no arrival or departure. *count-time* is used in the constraint that bounds the number of planes of each type.

The connection network is $G = (N, E)$, where the set of nodes $N = \{i: i \in L\}$ and the arc set $E = \{(i, j)$: *arr-city*(i) = *dep-city*$(j)\}$. The arc set includes all possible connections between arrival and departure flights at each station. Let $I(i) = \{(j, i) \in E: j \in N\}$ and $O(i) = \{(i, j) \in E: j \in N\}$ be the set of incoming and outgoing arcs for a node i. We define the set of overnighting arcs as $ON = \{(i, j) \in E$: *arr-time*(i) < *count-time* < *dep-time*$(j)\} \cup \{(i, j) \in E$: *dep-time*$(i)$ < *count-time* < *arr-time*$(i)\}$.

For each leg $i \in L$ and fleet type $f \in F$, a binary variable y_i^f takes value 1 if fleet type f is assigned to leg i, and 0 otherwise. For each arc $(i, j) \in A$ and fleet type $f \in F$, a binary variable x_{ij}^f takes value 1 if the connection (i, j) is selected and legs i and j have fleet type f, and 0 otherwise. Using this notation, ctFAM is the following integer programming problem:

Objective Function:

$$\min \quad z^1(x, y) = -\sum_{i \in \mathrm{N}} \sum_{f \in \mathrm{F}} c_i^f y_i^f - \sum_{(i,j) \in \mathrm{E}} \sum_{f \in \mathrm{F}} d_{ij}^f x_{ij}^f \tag{4.1a}$$

Constraints:

$$\sum_{f \in \mathrm{F}} y_i^f = 1, \quad \text{for all } i \in N, \tag{4.1b}$$

$$\sum_{(i,j) \in O(i)} x_{ij}^f = y_i^f, \quad \text{for all } i \in N \text{ and all } f \in F, \tag{4.1c}$$

$$\sum_{(i,j) \in I(j)} x_{ij}^f = y_j^f, \quad \text{for all } j \in N \text{ and all } f \in F, \tag{4.1d}$$

$$\sum_{(i,j)\in ON} x_{ij}^{f} \le size(f), \quad \text{for all } f \in F, \tag{4.1e}$$

$$x_{ij}^{f} \in \{0,1\}, \quad \text{for all } (i,j) \in E \text{ and for all } f \in F, \tag{4.1f}$$

$$y_{i}^{f} \in \{0,1\}, \quad \text{for all } i \in E \text{ and for all } f \in F. \tag{4.1g}$$

The objective function (4.1a) represents the negative of the contributions resulting from the fleet assignment and through assignment. Therefore, minimizing this function is equivalent to maximizing fleet and through assignment contribution. The constraints (4.1b) ensure that each flight leg is assigned exactly one fleet type. The constraints (4.1b), (4.1c) and (4.1d) together imply that each flight leg is assigned to another flight leg using a connection arc, and the two flight legs and the connection arc are assigned the same fleet type. The constraint (4.1e) ensures that the total number of planes of fleet type *f* in the assignment, which is the sum of the flows on arcs in *ON*, is no more than the available planes given by *size*(*f*).

4.2.2 Neighborhood Search Algorithm for ctFAM

The combined through and fleet assignment model is too large to be solved to optimality using the existing commercial-level software. Therefore, the airlines typically solve the fleet assignment problem first followed by the through assignment problem. This sequential approach is tractable but it is suboptimal. Ahuja et al. [2001c] proposed a neighborhood search approach to heuristically improve the solution obtained by the sequential method. The neighborhood structure used by Ahuja et al. [2001c] is based on the concept of *A-B swap*. Given a solution (x, y) to the integer programming formulation above, and two fleet types *A* and *B*, a solution (x', y') is called an *A-B neighbor* of (x, y) if it differs from (x, y) in the assignment of only two fleet types *A* and *B*. The operation of

obtaining an *A-B* neighbor is called performing an *A-B* swap. It is possible to identify the best *A-B* neighbor of a solution (x, y) by solving a restricted integer program; however, it is computationally very expensive. Ahuja et al. [2001c] dramatically reduced the running time by searching for improvements in a graph referred to as the *A-B* improvement graph, and which is denoted as $G^{AB}(x, y)$.

We next briefly describe the construction of the A-B improvement graph, $G^{AB}(x, y)$, for solution (x, y) and fleet types *A* and *B*. The node set of the graph $G^{AB}(x, y)$, denoted by $N(G^{AB}(x, y))$, is the set of flight legs that are assigned fleet type *A* or *B* in the solution (x, y), i.e., $N(G^{AB}(x, y)) = \{i \in N: y_i^A = 1 \text{ or } y_i^B = 1\}$. We will be searching for a constrained cycle C in $G^{AB}(x, y)$ (that is, a directed cycle in $G^{AB}(x, y)$ satisfying some additional constraints). The presence of a node *i* in C indicates that flight i switches its fleet type, either from *A* to *B* or from *B* to *A*. The cost c_{ij}^1 of the arc (i, j) is equal to the increase in the fleeting and through contributions resulting from the "changes attributed to the arc (i, j)." There are six types of arcs that can be added to the improvement graph. We provide a brief description of each of these cases next.

Type 1 Arcs:

An arc (i, j) is introduced in the A-B improvement graph if $(i, j) \in E$ such that $(i, j) \notin ON$ and $x_{ij}^A = 1$. The cost of the arc (i, j), c_{ij}^1, is the net increase in the objective function $z^1(x, y)$ from (i) the increase in the fleeting cost of flight *i* as it is changed from type *A* to type *B*, and (ii) the increase in the connection cost of arc (i, j) as the fleet type of i and j changes to *B*. The cost of the arc (i, j), c_{ij}^1 is equal to $(c_i^B - c_i^A) + (d_{ij}^B - d_{ij}^A)$. We do not include the effect of the change in the fleeting of flight j as it is modeled by the arcs that

emanate from node *j*. (If the cost of changing the fleet types of both the legs *i* and *j* were included in the cost of arc (i, j), then when we sum the cost of arcs in a directed cycle, we would be double counting the changes in the fleeting contributions.)

Type 2 Arcs:

An arc (j, i) is introduced in the *A-B* improvement graph if $(i, j) \in E$ such that $(i, j) \notin ON$ and $x_{ij}^B = 1$. Contrary to the case of type 1 arcs, we introduce the arc (j, i) instead of arc (i, j) although the cases are similar. Ahuja et al [2001c] describe the reason why we add the arc (j, i) instead of the arc (i, j). The cost of the arc (j, i) is given by $c_{ji}^1 = (c_j^A - c_j^B) + (d_{ij}^B - d_{ij}^A)$.

Type 3 Arcs:

An arc (i, l) is introduced in the *A-B* improvement graph if $x_{ij}^A = 1$, $x_{kl}^A = 1$, and if flight *i* can connect to flight *l* and flight *k* can connect to flight *j* (without increasing the left hand side term of the fleet size constraints (4.1e) for the fleet type *A* or *B*). If this arc is part of a directed cycle, it corresponds to the flight legs *i* and *l* changing from type *A* to type *B* with *i* connecting to *l*, and reconnecting flight *k* to flight *j*. The cost of the arc is given by $c_{il}^1 = (c_i^B - c_i^A) + (d_{il}^B + d_{kj}^A) - (d_{ij}^A + d_{kl}^A)$.

Type 4 Arcs:

An arc (j, k) is introduced in the *A-B* improvement graph if $x_{ij}^B = 1$, $x_{kl}^B = 1$, and if flight *i* can connect to flight *l* and if flight *k* can connect to flight *j* (without increasing the number of planes used for fleet types *A* and *B*). Notice that a type 4 arc is similar to a type 3 arc except that the direction of the arc is reversed. The cost of the arc is given by $c_{jk}^1 = (c_j^A - c_j^B) + (d_{il}^B + d_{kj}^A) - (d_{ij}^B + d_{kl}^B)$.

Type 5 Arcs:

An arc (i, k) is introduced in the *A-B* improvement graph if $x_{ij}^{A} = 1$, $x_{kl}^{B} = 1$, and if flight i can connect to flight l and if flight k can connect to flight j, (without increasing the number of planes used for fleet types *A* and *B*). If this arc is part of a directed cycle, then flight leg i switches from type *A* to type *B*, and connects to flight leg l, and flight k switches from type *A* to type *B*, and there is a connection from flight leg k. The cost is $c_{ik}^{1} = (c_i^B - c_i^A) + (d_{il}^B + d_{kj}^A) - (d_{ij}^A + d_{kl}^B)$.

Type 6 Arcs:

An arc (j, l) is introduced in the *A-B* improvement graph if $x_{ij}^{B} = 1$, $x_{kl}^{A} = 1$, and if flight i can connect to flight l and if flight k can connect to flight j (without increasing the number of planes used for fleet types *A* and *B*). This arc is similar to an arc of type 5, except that its direction is reversed. The cost of the arc is given by $c_{jl}^{1} = (c_j^A - c_j^B) + (d_{il}^B + d_{kj}^A) - (d_{ij}^B + d_{kl}^A)$.

For each node i, we define its "mate" as follows. If $x_{ij}^{A} = 1$ then $mate(i) = j$. If $x_{kl}^{B} = 1$ then $mate(i) = k$. A directed cycle W in the *A-B* improvement graph is said to be a valid cycle if it satisfies the property that for every node $i \in W$ either $mate(i) \notin W$ or (i, $mate(i)) \in W$. The following result was shown by Ahuja et al. [2001c].

Theorem 4-1. *Each valid cycle in the A-B improvement graph $G^{AB}(x, y)$ gives an A-B swap with respect to the solution (x, y). The cost of the cycle is equal to the change in the cost $z^1(x, y)$ of the solution (x, y) obtained by performing the A-B swap.*

Theorem 4-1 implies that an improved solution of the ctFAM can be found by identifying a negative cost valid cycle. Unfortunately, identifying a negative cost valid cycle in a graph is an *NP*-Complete problem (Ahuja et al. [2001a]), however, can be

solved quite efficiently in practice. This problem can be formulated as an integer programming problem and using the integer programming software CPLEX 7.0, we were able to obtain negative cost valid cycles in a fraction of a second.

A local improvement algorithm for ctFAM based on the *A*-*B* improvement graph works as follows. We first solve the FAM to obtain the fleeting solution x. Using the fleeting solution x, we solve a TAM to obtain a through solution y. (We solve both models using an integer programming solver.) We then use the solution (x, y) as the starting solution of the neighborhood search. We choose a pair (A, B) of fleet types and construct the corresponding *A*-*B* improvement graph. If the *A*-*B* improvement graph contains a negative cost valid cycle, we perform the corresponding *A*-*B* swap and update the *A*-*B* improvement graph to reflect changes in the current solution. If there are no negative cost valid cycles, we choose another pair of fleet types. The algorithm terminates if a solution (x, y) is found such that there are no cost-improving *A*-*B* swaps possible for any pair (A, B) of fleet types.

Ahuja et al. [2001c] implemented this local improvement algorithm and an extension of this algorithm using tabu search (see, Glover and Laguna [1997] for details on tabu search). Using this local improvement algorithm, they were able to substantially improve the combined through and fleet assignment contributions. Since this technique does not take into account subsequent planning stages such as crew scheduling and manpower scheduling, the solution obtained from the combined through and fleet assignment model may not be a good solution with respect to the crew and manpower considerations. We next discuss our multi-criteria model, which incorporates these considerations within through and fleet assignment decisions.

4.3 Multi-criteria ctFAM

In this section, we describe our multi-criteria version for ctFAM and the objective functions used for two of the planning stages: (i) ground manpower scheduling, and (ii) crew scheduling. United Airlines developed an approximation $z^1(x, y)$ for ground manpower scheduling as a linear function of a ctFAM solution (x, y). In addition, United Airline developed an approximation $z^2(x, y)$ for crew scheduling costs as a non-linear function of (x, y). Both approximations were developed after extensive testing and validation. It would have been very difficult to incorporate the non-linear function into an integer programming formulation. However the non-linearity is more amenable to approximation in a neighborhood search approach.

We partition the bi-criteria version of ctFAM problem in two problems. In the first problem, we solve ctFAM using objectives $z^1(x, y)$ and $z^2(x, y)$. In our second problem, we solve ctFAM using objectives $z^1(x, y)$ and $z^3(x, y)$. We henceforth refer to the first problem as MCP1, and the second problem as MCP2. In general, there does not exist a solution that will simultaneously minimize both the objectives in each of the two problems. Instead one has to search for Pareto-optimal solutions (Steuer [1986]). A feasible solution (x, y) for MCP1 is called a Pareto-optimal solution if there does not exist any other solution (x', y') which strictly dominates (x, y), that is, there is no (x', y') such that either (i) $z^1(x', y') \leq z^1(x, y)$ and $z^2(x', y') < z^2(x, y)$, or (ii) $z^1(x', y') < z^1(x, y)$ and $z^2(x', y') \leq z^2(x, y)$. Pareto-optimality for the problem MCP2 is defined in the same manner except that we replace z^2 by z^3.

Determining a Pareto-optimal solution for the bi-criteria ctFAM is itself a difficult problem. As a matter of fact, determining an optimal solution for the single criteria ctFAM is a difficult problem and Ahuja et al. [2001c] solved this problem heuristically as the MIP based exact algorithm was unable to solve the realistic instances for the United Airlines. Since we cannot solve the single criteria ctFAM, we obviously cannot solve the bi-criteria ctFAM. We therefore focused on determining non-dominated solutions of ctFAM; that is, those solutions that are not dominated by other solutions.

Our neighborhood search algorithms for determining non-dominated solutions of ctFAM search for solutions in such a way that the first objective remains nearly optimal but that the second criteria substantially improves. Ideally, we would generate all the non-dominated solutions (x, y) such that $z^1(x, y) \leq z^* + \alpha$, where α is a small value specified by United Airlines, and z^* is the optimal objective function value of ctFAM. The number of such solutions may be quite large and they are difficult to find but we did generate a large number of solutions as alternative inputs for the subsequent stages of airline scheduling. The alternatives permitted more flexibility in subsequent planning stages. We next describe the two objectives $z^2(x, y)$ and $z^3(x, y)$.

4.3.1 Ground Manpower Scheduling

The ground manpower scheduling objective function, $z^2(x, y)$, determines the requirement of the manpower on the ground. This function depends on the number of aircraft of each class arriving into a station, and the number of overnight connections of each class at the station. The aircraft in each class require a different level of ground manpower and hence affect the total ground manpower requirement at a station

differently. Let n_s^f and o_s^f, respectively, denote the number of arrivals and overnight connections for the fleet type *f* at a given station *c* in the current solution. Then,

$$z^2(x, y) = \sum_{s \in S} \sum_{f \in F} c_s^f n_s^f + d_s^f o_s^f \ ,$$

where the constants c_s^f and d_s^f are obtained using analysis of the past data. We now express the terms n_s^f and o_s^f as functions of the variables *x* and *y*. For a station *s*, let $Ls = \{i \in L: arr\text{-}city(i) = s\}$ and $ON^s = \{(i, j) \in ON: arr\text{-}city(i) = s\}$ denote the sets of incoming flight legs and overnight connection arcs between legs at the city *s*, respectively. The terms n_s^f and o_s^f can be re-written as a linear function of the variables *x*, *y* associated with the arrivals and connections at the station as follows:

$$n_s^f = \sum_{i \in L^s} y_i^f \quad \text{and} \quad o_s^f = \sum_{(i,j) \in ON^s} x_{ij}^f \ , \tag{4.2}$$

We thus observe that $z^2(x, y)$ is a linear function of the variables *x* and *y*. Simplifying the expression of z^2 above, it can be shown that:

$$z^2(x, y) = \sum_{i \in N} \sum_{f \in F} \bar{c}_i^f y_i^f + \sum_{(i,j) \in E} \sum_{f \in F} \bar{d}_{ij}^f x_{ij}^f \ , \tag{4.3}$$

where $\bar{c}_i^f = c_{arr-city(i)}^f$, and $\bar{d}_{ij}^f = d_{arr-city(i)}^f$ if $(i, j) \in ON^{arr\text{-}city(i)}$ and 0 otherwise.

The requirement of ground manpower at a station is also influenced by other terms that are independent of the fleet and through assignments, such as the total number of arrivals, trade union agreements, and the duration of the workday at each station. But these terms only add a constant factor to the objective function term and we do not consider them while solving the multi-criteria ctFAM. The function $z^2(x, y)$ can be computed in time linear in the number of variables. Further, since $z^2(x, y)$ is a linear

function of (x, y), we can easily modify the algorithm of Ahuja et al. [2001c] for ctFAM by using $z^2(x, y)$ as the objective function instead of $z^1(x, y)$.

4.3.2 Crew Scheduling

In the crew scheduling problem, we need to decide the itineraries of crews in the flight schedule. Typically, the objective function and constraints for this problem are quite complex depending on the contractual agreements between the airline and the trade unions (Barnhart and Talluri [1997]). United Airlines developed a function $z^3(x, y)$ to estimate the cost of the optimal crew schedule using a given fleet and through assignment (x, y). We describe this function next. The function $z^3(x, y)$ involves five separate terms.

Suppose that a flight i1 arrives at a station s. We say that i1 is an illegal overnight in the current solution (x, y) if the following is true:

4. i_1 is on the ground at midnight.
5. there is no flight i2 departing from s that is crew compatible with i_1 and that departs at least Δ_1 time units after the arrival of i_1. (We say that two fleet types A and B are *crew compatible* if the same crew can be assigned to an aircraft of either fleet type.)

The total number of illegal overnights is an important driver of the crew costs and is used in a term in our function, which is a nonlinear function of (x, y).

Let i and i' be the arrivals at a station, connecting to flight legs j and j' respectively in the solution (x, y), i.e., $x_{ij}^{f} = 1$ and $x_{i'j'}^{f'} = 1$ for some fleet types f and f'. We say that arrivals i and i' have a swap opportunity if f' and f are crew compatible and the departures j and j' occur after flight legs i' and i respectively. These conditions allow the crew of flight i' to go on flight j and crew of flight i to go on flight j'. The presence of swap opportunities increases the set of possible routings for the crews. The total number

of swap opportunities in the current solution (x, y) is also a non-linear function of the variables.

We call a connection a tight turn if the duration of the connection is less than a given parameter Δ_2. The function $z^3(x, y)$ also depends on the total number of tight turns in the ctFAM solution. For each flight leg, the number of hours needed to go from its departure station to the arrival station is called *crew block hours*. The objective function $z^3(x, y)$ also depends on the total number of crew block hours for the flight legs that are assigned a set of crew compatible fleet types. Lastly, the objective function depends on the total number of aircraft used in the ctFAM solution too.

We let *IO* be the number of illegal overnights, *SO* be the number of swap opportunities, *TT* be the number of tight turns, *BH* be the total number of crew block hours, and *N* be the total number of aircraft used in the solution (x, y) for ctFAM. The objective function $z^3(x, y)$ for the crew scheduling is given by:

$$z^3(x, y) = k_1 \cdot TT + k_2 \cdot BH + k_3 \cdot N + k_4 \cdot IO + k_5 \cdot SO\,, \tag{4.4}$$

where k_1, k_2, k_3, k_4, and k_5 are predetermined constants.

We now describe how each of the five terms can be computed from the variables x and y. We first express the linear terms, *TT*, *BH*, and *N* as functions of x and y. Let $E^2 = \{(i, j) \in E$: *dep-time*(j) – *arr-time*$(i) > \Delta_2\}$. Let b_i^f denote the block hours for a flight leg i when it is assigned fleet type f. The terms *TT*, *BH*, and *N* are given by:

$$TT = \sum_{(i,j)\in \mathrm{E}^2} \sum_{f\in \mathrm{F}} x_{ij}^f\,, \tag{4.5}$$

$$BH = \sum_{i\in \mathrm{N}} \sum_{f\in \mathrm{F}} b_i^f y_i^f\,, \tag{4.6}$$

$$N = \sum_{(i,j)\in ON} \sum_{f\in F} x_{ij}^f\,. \tag{4.7}$$

Although each of the above terms appears to be linear, the nonlinearities arise because the sets used in the summations depend in nonlinear ways on x and y. We next describe the term IO. Let $IO^s = |\{i \in N: \textit{arr-city}(i) = s \text{ and } i \text{ is an illegal overnight}\}|$ denote the number of illegal overnight arrivals at station s. The term IO can be written as:

$$IO = \sum_{s \in S} IO^s \ . \tag{4.8}$$

Let $SO^s = |\{(i, i'): \textit{arr-city}(i) = \textit{arr-city}(i') = s \text{ and } i, i' \text{ have a swap opportunity}\}|/2$ denote the number of swap opportunities at station s. Then the term SO can be written as:

$$SO = \sum_{s \in S} SO^s \ . \tag{4.9}$$

4.4 Neighborhood Search for Multi-criteria ctFAM

In this section, we describe our neighborhood search approach for the multi-criteria ctFAM. Our algorithms for MCP1 and MCP2 are motivated by the computational investigations of our neighborhood search algorithm for ctFAM reported in Ahuja et al. [2001c]. As described in Section 4.2.2, this algorithm first solves FAM to obtain the fleeting solution x, then solves TAM to obtain the through solution y, and then uses neighborhood search algorithm to improve the fleeting-through solution (x, y). We observed that the the neighborhood search algorithm substantially improves the through contribution of the solution while marginally worsening the fleet assignment contribution. So, an optimal solution of FAM is a mediocre starting point with respect to TAM, but there are FAM solutions that are nearly optimal, and are far better with respect to the TAM objective. Extending this observation would suggest the following: *an optimal solution of ctFAM may not be a good solution with respect to the second criteria (ground manpower scheduling or crew scheduling) but there may exist ctFAM solutions that are nearly optimal and are very good with respect to the second criteria*.

Our approach uses the preceding observation and builds upon the framework of Ahuja et al. [2001c] for ctFAM with the single objective. We select a pair *A* and *B* of fleet types and perform cost improving *A*-*B* swaps. We incorporate additional objectives into the neighborhood search by associating multiple cost terms with each arc in the *A*-*B* improvement graph, one for each objective. Using these additional cost terms, we look for a valid cycle that minimizes some weighted combination of these arc costs or just the individual arc costs itself. Depending on the choice made, the search could be directed towards improving some weighted combination of objectives or a single objective. We next discuss how we determine the arc costs corresponding to the two additional objectives we introduced in Section 4.3.

4.4.1 Arc Costs in the A-B Improvement Graph

In this section, we define the costs for arcs in the *A*-*B* improvement graph for the two additional objectives defined in Section 4.3.

Ground Manpower Scheduling:

Recall from Section 4.2 that a solution of ctFAM is represented by (x, y) where x represents a fleeting solution and y represents a through solution. We associated arc costs in the *A*-*B* improvement graph $G^{AB}(x, y)$ with respect to the solutions (x, y) such that the cost of any valid cycle in the improvement graph equals the change in the cost of the solution (x, y) due to the corresponding *A*-*B* swap.

We observe from (4.3) that the function $z^2(x, y)$ is a special case of the function $z^1(x, y)$ with the costs c_i^f and d_{ij}^f replaced with $\overline{c}_i^f$ and $\overline{d}_{ij}^f$, respectively. Ahuja et al. [2001c] provides definitions of the arc costs in the *A*-*B* improvement graph for the function z^1 such that the cost of a valid cycle is equal to the change in the function z^1 after

performing the corresponding *A-B* swap. Using this observation, we define the costs of the six types of arcs in the *A-B* improvement graph described in Section 4.2.2 by replacing c_i^f and d_{ij}^f with $\overline{c}_i^f$ and $\overline{d}_{ij}^f$, respectively. For example, the cost of an arc (i, j) of Type 1 in Section 4.2.2 for the function $z^2(x, y)$ is denoted by c_{ij}^2 and is equal to $(\overline{c}_i^B - \overline{c}_i^A) + (\overline{d}_{ij}^B - \overline{d}_{ij}^A)$. The cost of a valid cycle with arc costs c^2 is equal to the change in function z2 after performing the corresponding *A-B* swap. The time to compute the cost of an arc is the same as that for z^1, which is $O(1)$.

Crew Scheduling:

The function $z^3(x, y)$ for the crew scheduling problem is nonlinear, so the arc cost definitions of Ahuja et al. [2001c] cannot be used in a straightforward manner as done in ground manpower scheduling. In fact, it is not possible to define arc costs such that the sum of costs along a valid cycle is equal to the change in the function after performing the corresponding *A-B* swap. We define arc costs such that the sum of the costs along a valid cycle is an approximation of the change in the function. We describe these arc costs next.

The function $z^3(x, y)$ is a weighted sum of the five terms: *TT*, *BH*, *N*, *IO*, *SO*, which are defined in Section 4.3. The increase in function z^3 corresponding to an arc in the *A-B* improvement graph is a weighted sum of the increases in these terms. We calculate the cost of an arc for function z^3 by computing the increase in each term and taking their weighted combination. Let c_{ij}^2 denote the cost of an arc (i, j) with respect to z^3. This cost is computed as $c_{ij}^2 = k_1 \cdot c_{ij}^{TT} + k_2 \cdot c_{ij}^{BH} + k_3 \cdot c_{ij}^{N} + k_4 \cdot c_{ij}^{IO} + k_5 \cdot c_{ij}^{SO}$ where the costs $c_{ij}^{TT}, c_{ij}^{BH}, c_{ij}^{N}, c_{ij}^{IO}, c_{ij}^{SO}$ denote the changes in *TT*, *BH*, *N*, *IO*, and *SO* resulting from arc (*i*,

j). We next describe how to compute these changes for an arc in the *A-B* improvement graph.

We note that the terms *TT*, *BH*, and *N* can each be treated as a special case of the function z^1 because the sets N, F, E^2, ON are independent of the solution (x, y). For example, the term *TT* in (4.5) can be obtained from z1 by setting $c_i^f = 0$ for each $i \in N$ and $f \in F$ and $d_{ij}^f = 1$ if $(i, j) \in E^2$ and 0 otherwise. We compute the costs c^{TT}, c^{BH}, and c^N in the same way as arc costs c^1 described in Section 4.2.2 for function $z^1(x, y)$. For each arc, the costs c^{TT}, c^{BH}, and c^N can be computed in time $O(1)$ similar to c^1. From Theorem 1, the cost of a valid cycle with respect to arc costs c^{TT} (c^{BH} or c^N) is equal to the change in *TT* (*BH* or *N*) by performing the corresponding *A-B* swap. Since the cost of a valid cycle is the sum of the cost of its arcs, it follows that the cost of a valid cycle with respect to a weighted combination of c^{TT}, c^{BH} and c^N is equal to the increase in the same weighted combination of *TT*, *BH*, and *N* by performing the corresponding *A-B* swap.

The terms *IO* and *SO* are nonlinear and cannot be expressed as special cases of z^1, so we use a different definition for c^{IO} and c^{SO}, which we describe next. Recall from expressions (4.8) and (4.9) in Section 4.3.2 that the terms *IO* and *SO* can each be written as a sum of terms IO^s and SO^s , which are functions of arrival, departure, and the connection variables at the station *s*. Next, we observe that the changes indicated by each of the six types of arcs in the *A-B* improvement graph result in modifying the connection variables *x* corresponding to connections at a single station. For example, the arc (i, j) of Type 1 in Section 4.2.2 results in changing connection variables x_{ij}^A and x_{ij}^B corresponding to the connection (i, j) at station *arr-city*(i) (=*dep-city*(j)). Therefore, we

can associate a station with each arc. We set the cost c_{ij}^{IO} (c_{ij}^{SO}) of the arc equal to the increase in the function IO^s (SO^s) for the station s corresponding to the arc (i, j) assuming that there will be no other changes to the connection variables at that station. Since a valid cycle may contain multiple arcs that modify connection variables at the same station, the cost of a valid cycle with respect to c^{IO} (c^{SO}) may not be equal to the increase in *IO* (*SO*) from the corresponding *A-B* swap. However, the following restricted result holds, which we state without proof.

Theorem 4-2. *For any valid cycle in the A-B improvement graph such that no two arcs in the cycle have the same station associated with them, the cost of the cycle with respect to arc costs c^{IO} (c^{SO}) is equal to the increase in IO (SO) from the corresponding A-B swap.*

Computing the functions IO^s or SO^s takes time $O(D^2)$, where D is the maximum number of departures (and arrivals) at a station. However, we can compute the change in each of these functions from arc (i, j) in time $O(D)$. We describe these computations for *IO* and *SO* next.

Recall that an arrival flight leg at a station is an illegal overnight if it makes an overnight connection and it has no legal friends. In our implementation, we maintain the number of legal friends for each flight leg in the current solution. Therefore, the number of illegal overnights at a station can be obtained by counting the number of arrivals that make an overnight connection at the station and have zero legal friends. This can be done in $O(D)$ time given the number of legal friends for each arrival. We can update the number of legal friends at the station s after performing the changes corresponding to arc (i, j) in $O(D)$ time using the following observation:

The legal friend relation between a pair of arrival and departure at a station depends solely on the fleet assignment variables corresponding to them. Therefore, if the fleet

assignment variables of the pair are not modified, their legal friend relation is also not affected.

In order to update the number of legal friends at the station s, we only consider pairs (l_1, l_2) of arrival and departure at s such that the fleet assignment of either l_1 or l_2 or both is modified by the arc (i, j). There are three possible disjoint cases for each pair:

1. If the departure l_2 is a legal friend of l_1 before the changes but not after the changes, we decrease the number of legal friends of l_1 by 1.
2. If the departure l_2 is not a legal friend of l_1 before the changes but is legal friend after the changes, we increase the number of legal friends of l_1 by 1.
3. In all other cases, we do not modify the number of legal friends of l_1.

Since each arc (i, j) modifies the fleet assignment of at most two arrivals and two departures, the number of pairs considered by us is $O(D)$. For any pair (l_1, l_2) of an arrival and a departure, we can check in $O(1)$ time if l_2 is a legal friend of l_1. Therefore, we can update the number of legal friends at a station in $O(D)$ time after the changes from an arc in the *A-B* improvement graph.

In the case of swap opportunities, we observe that for any pair of arrivals, their swap opportunity depends solely on their fleet assignments and connection assignments. For an arc (i, j) in the *A-B* improvement graph, we can compute the change in SO^s by only considering the change in swap opportunity for those pairs (l_1, l_2) of arrivals where at least one of the flight leg has its fleet or connection assignment modified by the arc (i, j). If the pair (l_1, l_2) has a swap opportunity before the changes of arc (i, j) but not afterwards, we decrease SO^s by 1. If the pair has a swap opportunity after the changes from arc (i, j) but not before it, we increase SO^s by 1. Using an argument similar to that for illegal overnights, this computation takes $O(D)$ time and gives the value of SO^s after the changes corresponding to (i, j).

4.4.2 Algorithms for Multi-Criteria Search

We developed two neighborhood search based algorithms to identify non-dominated solutions for the bi-criteria ctFAM. Our algorithms start with a ctFAM solution, and at each iteration performs an *A*-*B* swap for some pair of fleet types *A* and *B* to obtain a new ctFAM solution. The algorithms examine the set of solutions obtained in each iteration and output a subset of these solutions that are not dominated by each other. The primary difference in our algorithms is in the choice of the *A*-*B* swap in each iteration and the number of neighborhood search iterations performed. We next describe each of these in more detail.

Approach 1:

In our first approach, we perform neighborhood search with respect to a single objective: $\lambda z^1(x, y) + (1-\lambda)z^2(x, y)$ where $0 \leq \lambda \leq 1$. Observe that if we set $\lambda = 1$, then the ctFAM objective becomes the sole objective and if we set $\lambda = 0$, then the ground manpower scheduling objective becomes the sole objective. By varying λ between 0 and 1, we give different relative weights to the two objectives. We choose different discrete uniformly spaced values of λ in the interval 0, 1 and apply neighborhood search for each of these values. For a specific value of λ, we set the cost of an arc (i, j) in the improvement graph as $c_{ij} = \lambda c_{ij}^1 + (1-\lambda) c_{ij}^2$. We apply the standard neighborhood search, as outlined in Section 4.2.2 and always perform cost-decreasing swaps. We keep track of all the solutions enumerated by the algorithm and store the non-dominated solutions. Figure 4-1 gives a formal description of the algorithm.

The algorithm in Figure 4-1 is a local improvement algorithm and repeatedly improves the starting solution by performing cost-decreasing swaps. We also

implemented a tabu search algorithm where we occasionally perform cost-increasing swaps. We implemented a simple version of the tabu search algorithm based on short-term memory only. In this algorithm, we consider each pair of fleet types *A* and *B* and perform a specified number of *A*-*B* swaps. We stop when the current solution admits no cost-decreasing swaps for any pair of fleet types. Tabu search does not stop just because there is a local optimum. We use the integer programming formulation given in Ahuja et al. [2001c] to find valid cycles in the improvement graph. We refer to the local improvement version of this approach as LOCAL1 and the tabu search version as TABU1.

algorithm *multi-criteria-search*;
begin
1. let (x, y) be a ctFAM solution and $\lambda \leftarrow 1$;
2. let $PO \leftarrow \{(x, y)\}$ denote the set of solutions found;
3. let z1 and z2 be the two objective functions to be minimized;
4. **while** $\lambda \geq 0$ **do**
5. **begin**
6. **repeat**
7. choose a pair (A, B) of fleet types;
8. create the improvement graph $G^{AB}(x, y)$ and compute the arc costs z^1 and z^2;
9. set the cost of an arc (i, j) in GAB(x, y) to be cij $\leftarrow \lambda c_{ij}^1 + (1-\lambda) c_{ij}^2$;
10. **while** $G^{AB}(x, y)$ contains a negative cost valid cycle W **do**
11. **begin**
12. let (x', y') be obtained by the *A*-*B* swap corresponding to the valid cycle W;
13. **if** $\lambda z^1(x', y') + (1-\lambda)z^2(x', y') \leq \lambda z^1(x, y) + (1-\lambda)z^2(x, y)$ **then**
14. **begin**
15. set $(x, y) \leftarrow (x', y')$;
16. update $G^{AB}(x, y)$ and the arc costs c_{ij};
17. **if** (x, y) is not dominated by any solution in PO **then**
18. add (x, y) to PO and remove all solutions from PO dominated by (x, y);
19. **end;**
20. **else** terminate the while loop;
21. **end;**
22. **until** no improving A-B swap is found for any pair (A, B);
23. set $\lambda \leftarrow \lambda - \varepsilon$;
24. **end**;
end.

Figure 4-1. Local improvement based algorithm for Approach 1.

We described above our algorithm for the bi-criteria ctFAM where the ground manpower scheduling was our second objective. This algorithm easily applies to the case when crew scheduling is our second objective. We simply replace c^2 by c^3 and $z^2(x, y)$ by $z^3(x, y)$. Recall from Section 4.4.1 that the cost of a valid cycle in the improvement graph is not always equal to the change in the corresponding solution due to the corresponding *A-B* swap. It is possible that the valid cycle has a negative cost but the corresponding *A-B* swap has a positive cost. Therefore, before performing the *A-B* swap, we compute its exact cost and if it is negative, we perform the *A-B* swap; otherwise, we do not perform it.

Approach 2:

In our second approach, we first obtain an optimal solution of the ctFAM with $\lambda z^1 + (1-\lambda)z^2$ as the objective function with $\lambda = 1$ which is equivalent to making the first objective as the sole objective. This gives us a solution (x^*, y^*) with the value of the first objective equal to z^*. We now set $\lambda = 0$, which is equivalent to making the second objective as the sole objective, and apply neighborhood search algorithm with (x^*, y^*) as the starting solution. As the neighborhood algorithm proceeds, we add an additional constraint. We require that the current solution (x, y) always satisfies the constraint $z^* \leq z^1(x, y) \leq z^*+\alpha$ which we refer to as the range constraint. We look for valid cycles satisfying the range constraint using integer program based on the following formulation:

Objective Function:

$$\min \sum_{(i,j)\in E'} c_{ij} w_{ij} \tag{4.10a}$$

Constraints

$$\sum_{\{j:(j,i)\in E'\}} w_{ji} - \sum_{\{j:(i,j)\in E'\}} w_{ij} = 0, \quad \text{for all } i \in N', \tag{4.10b}$$

$$\sum_{\{j:(i,j)\in E'\setminus\{(i,mate(i))\}\}} w_{ij} + \sum_{\{j:(mate(i),j)\in E'\}} w_{mate(i),j} \le 1, \quad \text{for all } i \in N', \tag{4.10c}$$

$$\sum_{(i,j)\in E'} c_{ij}^{1} w_{ij} \le z^{*} + \alpha - z^{1}(x,y), \tag{4.10d}$$

$$w_{ij} \in \{0,1\}, \quad \text{for } (i,j) \in E'. \tag{4.10e}$$

We solve (4.10) by CPLEX which uses a branch and bound approach to obtain an optimal solution. The branch and bound algorithm proceeds by enumerating feasible solutions of (4.10). We select the best feasible solution of (4.10) enumerated by the branch and bound algorithm among a specified number of solutions (for example, 100 solutions), and use the corresponding *A*-*B* swap to improve the current solution. As in Approach 1, we keep track of all the non-dominated solutions found during the neighborhood search. When the algorithm terminates, all these solutions are given as output. We also considered a tabu search version of Approach 2 similar to that for Approach 1. We refer to the local improvement version of Algorithm 2 by LOCAL2 and the tabu search version by TABU2.

4.5 Computational Results

In this section, we present the computational results of our algorithms for the multi-criteria ctFAM. We programmed our algorithms in C programming language and tested them on a Pentium 4 1.4 GHz machine with 512MB RAM and Linux operating system. We used CPLEX 7.0 to solve the integer programs arising in the search of valid cycles and to find the initial ctFAM solution.

We obtain the starting ctFAM solution for our algorithms by sequentially solving the FAM and TAM. For our algorithms LOCAL1 and TABU1 from Approach 1, we use the value of λ varying from 0 to 1 with an increment of 0.01. We thus consider 101

values of λ. In Approach 2, we use α = 5,000 for both the algorithms, LOCAL2 and TABU2 which implies that we are willing to give up to $5,000 in fleet and through contribution to achieve better solutions with respect to the objective function for crew scheduling or ground manpower scheduling. In the tabu search algorithms, TABU1 and TABU2, we forbid changing the fleet type of a flight leg for the next 10 iterations if it is changed in the current iteration. We perform 100 iterations of the tabu search for each choice of the fleet type pair (*A*, *B*).

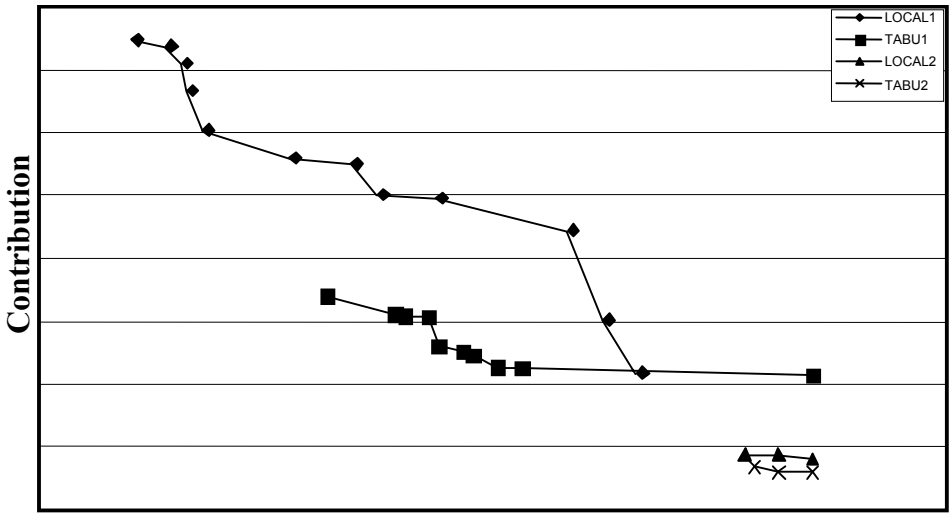


Figure 4-2. Non-dominated solutions for MCP1.

We tested the four algorithms, LOCAL1, TABU1, LOCAL2, and TABU2 with the settings mentioned above. We used the data provided by United Airlines for ctFAM and the functions z^2 and z^3. We set up two bi-criteria problems, MCP1 and MCP2, as described in Section 4.3. The solutions generated by our algorithms for the first and the second problem are shown in Figure 4.2 and Figure 4.3, respectively. Since both the problems are minimization problems, the closer a solution is to the left hand bottom of the graphs in Figures 4-2 and 4-3 the better it is.

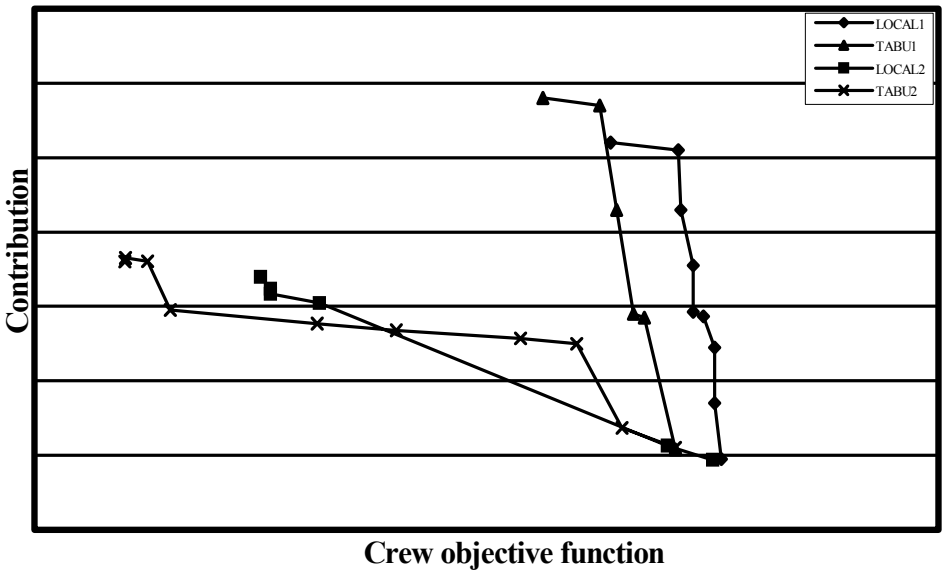


Figure 4-3. Non-dominated solutions for MCP2.

The following conclusions can be drawn from these figures.

- The number of non-dominated solutions found by all four algorithms is rather small.
- The solutions obtained by Approach 2 are better than those obtained by Approach 1.
- The set of non-dominated solutions obtained by the tabu search versions of both approaches are slightly better than the solutions obtained corresponding local improvement versions.

4.6 Conclusions

The use of neighborhood search techniques has shown significant promise in solving the next frontier of airline planning problems. The next generation airline planning problems are focused on the integration between business areas such as aircraft scheduling and crew scheduling. Since explicit joint optimization becomes computationally intractable, an attractive alternative is to formulate multi-criteria optimization problems that are solved using large-scale neighborhood search. The methodology provides a robust and accurate approach to achieve integration while managing the complexity in an effective manner.

The research work for this project was a collaborative effort between MIT, University of Florida and United Airlines. The objective was to solve the combined fleeting and through assignment problem using neighborhood search and develop a generic template for solving other joint optimization problems. The outcome of the research has been very promising. It has been shown that there are numerous fleet assignment solutions that are in the neighborhood of the "optimal" solution achieved from traditional fleet assignment models. Hence, there is a lot of opportunity to score similar solutions on other attributes that are scored using different criteria and/or metrics. We intend to implement this methodology in the near future.

CHAPTER 5
SOLVING THE COMBINED THROUGH AND FLEET ASSIGNMENT PROBLEM WITH TIME WINDOWS

5.1 Introduction

In this chapter, we develop a very large-scale neighborhood (VLSN) search algorithm for the combined through and fleet assignment model with time windows (ctFAM-TW).

In Chapter 4, we have described FAM, TAM, and ctFAM in details. Ahuja et al. [2001c] formulated the ctFAM as a mixed integer programming (MIP) problem; however, its size precluded solving it by the commercial MIP software. They thus developed a very large-scale neighborhood (VLSN) search algorithm that obtains a starting solution by solving FAM followed by TAM and repeatedly improving it. Their approach proceeds by swapping the fleet assignment of two flight paths flown by two different plane types so that the solution is feasible after the swaps and the total cost decreases. An important feature of this approach is that the size of the neighborhood is very large and it is implicitly enumerated using the concept of the *improvement graph*. This approach provides substantial cost savings over the sequential approach of solving FAM and TAM.

The ctFAM-TW model adds feature of time windows to the ctFAM. In this model, each flight leg has an associated time window with its departure time, and the flight can depart any time within its time window. For example, a flight leg may have its departure time as 8AM ±10 minutes, which would allow the model to set the departure time of this

flight to any value between 7:50AM to 8:10AM. (The arrival time of the flight at its destination is obtained by adding the flight time to the departure time at the origin city.) The ctFAM-TW needs to determine (i) the departure time of all flight legs; (ii) assignment of fleet types to various flight legs (consistent with the departure times chosen), and (iii) through connections between the flight legs flown by the same fleet type.

Allowing time windows for flight legs allows greater opportunities for connections between flight legs and can result in the savings of both the fleet assignment and through assignment costs. Consider, for example, the two flight legs whose arrival and departure times are give in Table 5-1. The flight 118 leaves Boston at 8 AM, arrives at Chicago and becomes available to be assignment to the next flight (called ready time) at 10AM. The flight 215 leaves Chicago at 9:55 AM for Denver, and since it leaves 5 minutes before the ready time of the plane assigned to flight 118, the same plane cannot be assigned to the two legs. However, if we associate a time window of ±10 minutes with both flight legs, then we can delay the departure time of flight 215 by 5 to 10 minutes and the same plane can fly both the legs. Thus, allowing time windows for flight legs creates additional opportunities for connections.

Table 5-1. An example of extra through connection opportunity if time windows are allowed

Flight No.	Origin	Destination	Departure Time	Ready Time
118	BOS	ORD	08:00 AM	10:00 AM
215	ORD	DEN	09:55 AM	12:00 Noon

In this chapter, we generalize the results in the paper on ctFAM for ctFAM-TW. We give a mixed integer programming formulation of the ctFAM-TW, which is a multicommodity flow problem on a graph, called the *connection graph*, with additional

side constraints. This integer programming problem is too large to be solved to optimality or near-optimality using current commercially available software. We next present our neighborhood search algorithm for ctFAM-TW which starts with a feasible solution and repeatedly improves it. In this neighborhood search algorithm, we define neighbors of a given solution by performing profitable *A-B swaps* for two specified fleet types *A* and *B*. An *A-B* swap consists of changing fleet types of some legs from *A* to *B* and of some legs from *B* to *A* so that all constraints remain satisfied. Identifying a profitable *A-B* swap is not a trivial problem because the number of possible *A-B* swaps is exponentially large. We describe a method using *A-B improvement graphs* which allows us to obtain profitable *A-B* swaps quickly in practice. The *A-B* improvement graph is constructed in a manner that each negative cost directed cycle in the graph satisfying some constraints defines a profitable *A-B* swap. Our local improvement algorithm obtains a local optimal solution for ctFAM-TW in 30 minutes on the data provided by United Airlines and resulted in revenue improvement of $78 million on an annual basis.

This chapter is organized in 5 sections. Section 5.2 describes the mathematical formulation of ctFAM-TW. In Section 5.3, we present our VLSN search algorithm to solve this problem. Computational results are given in Section 5.4, and Section 5.5 concludes this chapter.

5.2 Mathematic Formulation

In this section, we present a mixed integer linear (MIP) formulation of ctFAM-TW. We formulate ctFAM-TW as a flow problem on a graph, called *connection graph*. We first give the notation for ctFAM-TW, then the description of the connection graph, and followed by the MIP formulation.

5.2.1 Notation

We use the following notation:

- L: The set of flight legs which needed to be assigned planes. We will use the indices *i, j, k, l, r, s, t* and *u* to represent specific flight legs.
- F: The set of all fleet types. We use the index f to represent a specific fleet type.
- T: The set of all candidate through connections. Each through connection is specified by a pair (i, j) of flights legs.
- *fleet-size*(f): The number of planes of fleet type f available for assignment.
- *turn-time*(f): The minimum time needed to prepare a plane of fleet type f to be assigned to the next flight leg after it arrives at the airport. The *turn-time* of different fleet types can vary from 30 minutes to 45 minutes.
- *dep-time*(l): The departure city for flight leg l.
- *arr-city*(l): The arrival city for flight leg l.
- *scheduled-dep-time*(l) : The scheduled departure time for flight leg l.
- *flying-time*(l, f) : The duration of flight leg l between its departure at *dep-city*(l) and arrival at *arr-city*(l) when flown by fleet type f. The difference of *flying-time* when a flight is assigned with different fleet types can be as much as 30 minutes.
- *time-window-width*(l) : The time window width for flight leg l, within which the departure time of the flight leg can be shifted. We assume that the time window for flight leg i is centered at *scheduled-dep-time*(l) . Thus, the earliest allowable departure time for flight leg i is calculated as [*scheduled-dep-time*(l) – *time-window-width*(l) / 2], and the latest allowable departure time can calculated as [*scheduled-dep-time*(l) + *time-window-width*(l) / 2]. We typically consider time window width to be 20 minutes.
- *dep-time-interval*(l) : The time interval between two consecutive allowable departure time for flight leg l. We assume that the allowable departure times for flight leg l are uniformly distributed within the time window at some discrete intervals. For example, if the scheduled departure time of a flight leg l is 8:00, the time window width is 10 minutes and departure time interval is 2 minutes, then this flight leg can depart at the following times: 7:50, 7:52, 7:54, 7:56, 7:58, 8:00, 8:02, 8:04, 8:06, 8:08, 8:10.
- c_l^f : The contribution obtained by assigning a plane of fleet type f to flight leg l.

- $d^{f}_{(i,j)}$: The contribution obtained from making through connection (i, j) when both the flight legs i and j have the same fleet type f.

5.2.2 Connection Graph

We now explain how to construct the *connection graph*, which will be the basis of our integer programming formulation as well as our neighborhood search algorithm for ctFAM-TW. We denote the connection graph as $G = (N,E)$ where N denotes the node set and E denotes the arc set.

The node set $N = \{K(l) : l \in L\}$ is obtained by defining multiple copies of nodes for all flight legs $l \in L$. Each node in $K(l)$ is called a flight node copy of flight leg l. Flight node copies of a flight represent the same flight leg with different departure times. We use $l_1, l_2, \ldots, l_v, l_w$ denote the flight node copies that represent flight leg l; and we use $dep\text{-}time(l_w)$ to denote the departure time for flight node copy l_w. Since the flying-time and turn-time are dependent on the fleet type, we need the following additional notation:

$$arr\text{-}time(l_w, f) = dep\text{-}time(l_w) + flying\text{-}time(l, f),$$

$$ready\text{-}timef(l_w, f) = arr\text{-}time(l_w, f) + turn\text{-}time(f).$$

The term $arr\text{-}time(l_w, f)$ denotes the arrival time of flight node copy l_w when it is assigned with fleet type f, and $ready\text{-}time(l_w, f)$ denotes the time when the plane of fleet type f, which flies flight leg l and departs at $dep\text{-}time(l_w)$, is ready to fly the next flight leg.

As a matter of fact, the ctFAM can be interpreted as a special case of ctFAM-TW, where every flight leg is represented by a single flight node copy. In this chapter, we assume that for every flight leg l with $time\text{-}window\text{-}width(l) > 0$, there are at least three flight node copies to represent the earliest allowable departure time, the scheduled

departure time, and the latest allowable departure time, respectively. The number of flight node copies for flight leg l (or the cardinality of $K(l)$) is equal to $\textit{time-window-width}(l) / \textit{dep-time-interval}(l)$. We will show that the MIP formulation of ctFAM-TW requires a decision variable for each fleet-type and flight-node-copy combination, and the number of decision variables can grow rapidly if we select large values of $\textit{time-window-width}(l)$ and/or small values of $\textit{dep-time-interval}(l)$. Thus, the selection of these two parameters would govern the tradeoff between modeling accuracy and computational difficulty.

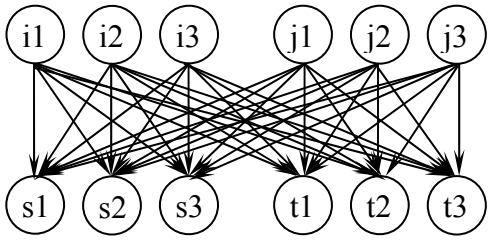


Figure 5-1. An example of part of the connection graph at a city with the inbound flight legs i and j, and outbound flight legs s and t.

The arc set $E = \{(i_w, j_v) : i_w \in K(i), j_v \in K(j), \textit{arr-city}(i) = \textit{dep-city}(j)\}$ consists of all possible connections between inbound and outbound flight node copies, when the arrival city of inbound flight leg i is the same as the outbound flight leg j. We give an example of connection graph in Figure 5-1.

A connection arc (i_w, j_v) is said to be a *through connection arc* if $(i, j) \in T$ and a *regular connection arc,* otherwise. We will use the following additional notation related to the connection graph:

- $I(i_w) = \{(j_v, i_w) \in E : j_v \in N\}$: The set of incoming arcs at flight node copy i_w in the connection graph.
- $O(i_w) = \{(i_w, j_v) \in E : j_v \in N\}$: The set of outgoing arcs at flight node copy i_w in the connection graph,

- *count-time*: A time instant on the 24-hour time scale when no plane might leave or arrives, that is, $count\text{-}time \neq arr\text{-}time(l_w)$ or $dep\text{-}time(l_w)$ of any $l_w \in N$. We will assume here that the $count\text{-}time$ is midnight.

We next define following subsets of the connection graph that will facilitate us to specify the fleet-size constraint of each fleet type.

$$E^S(f) = \{(i_w, j_v) \in E : ready\text{-}time(i_w, f) < count\text{-}time < dep\text{-}time(j_v)\},$$

$$N^S(f) = \{l_w \in N : dep\text{-}time(l_w) < count\text{-}time < arr\text{-}time(l_w, f)\}.$$

Arcs in the subset $E^S(f)$ are called *overnight connection arcs* of fleet type *f*. Every connection arc in $E^S(f)$ crosses the $count\text{-}time$, when the arc represents the connection of a plane of fleet type *f*. Nodes in the subset $N^S(f)$ are called *red-eye flight node copies.* Every flight node copy in $N^S(f)$ crosses the *count-time*, when the flight is flown by fleet type *f* is in the air. Note that the inequalities in the definition of $E^S(f)$ and $N^S(f)$ are based on the circular 24-hour time.

We further define the red-eye predictor $RE(l_w, f)$ and overnight predictor $ON(i_w, j_v, f)$ as follows:

$RE(l_w, f) = 1$ if $l_w \in N^S(f)$, and $RE(l_w, f) = 0$ otherwise,

$ON(i_w, j_v, f) = 1$ if $(i_w, j_v) \in E^S(f)$, and $ON(i_w, j_v, f) = 0$ otherwise.

5.2.3 Decision Variables

Our integer programming formulation uses two set of decision variables. The first set of decision variables, $y^f_{l_w}$, specifies the fleet assignment and departure times of flight legs; and the second set of decision variables, $x^f_{i_w j_v}$, specifies the (regular or through) connection assignments.

- $y_{l_w}^f$: This variable takes value 1 if the flight leg l is assigned a plane of fleet type f and departs at time equal to *dep-time*(l_w); otherwise it takes value 0.

- $x_{i_w j_v}^f$: This variable takes value 1 if both the inbound flight leg i and outbound flight leg j are flown by the fleet type f, depart at times *dep-time*(i_w) and *dep-time*(j_v), respectively, and we make a (regular or through) connection between the flight legs; otherwise it takes a value 0.

5.2.4 The Integer Programming Formulation of ctFAM-TW

We give below the integer programming formulation of ctFAM-TW.

Objective Function:

$$\max \quad z = \sum_{l \in L} \sum_{l_w \in K(l)} \sum_{f \in F} c_l^k y_{l_w}^f + \sum_{(i,j) \in T} \sum_{i_w \in K(i)} \sum_{j_v \in K(j)} \sum_{f \in F} d_{(i,j)}^f x_{i_w j_v}^f \tag{5.1a}$$

Constraints:

$$\sum_{l_w \in K(l)} \sum_{f \in F} y_{l_w}^f = 1, \quad \forall l \in L, \tag{5.1b}$$

$$\sum_{(i_w, j_v) \in O(i_w)} x_{i_w j_v}^f = y_{i_w}^f, \quad \forall i_v \in N \text{ and } f \in F, \tag{5.1c}$$

$$\sum_{(i_w, j_v) \in I(j_v)} x_{i_w j_v}^f = y_{j_v}^f, \quad \forall j_v \in N \text{ and } \forall f \in F, \tag{5.1d}$$

$$\sum_{(i_w, j_v) \in E^S(f)} x_{i_w j_v}^f + \sum_{l_w \in N^S(f)} y_{l_w}^f \le \textit{fleet-size}(f), \quad \forall f \in F, \tag{5.1e}$$

$$x_{i_w j_v}^f \in \{0,1\}, \quad \forall (i_w, j_v) \in E \text{ and } \forall f \in F, \tag{5.1f}$$

$$y_{l_w}^f \in \{0,1\}, \quad \forall l_w \in N \text{ and } \forall f \in F. \tag{5.1g}$$

We represent a feasible solution of ctFAM-TW as (x, y). The first term in the objective function (5.1a) represents the revenue generated by the fleet assignment, and the second term represents the contribution resulting from the through assignment. The constraint (5.1b) ensures that each flight leg is assigned exactly one fleet type, and departs at one particular time. The constraints (5.1c) and (5.1d) together with (5.1b)

imply that each flight leg is connected to another flight leg using a connection arc, and the two flight legs and the connection arc are assigned the same fleet type. The constraint (5.1e) ensures that the total number of planes of fleet type *f* in the assignment, which is the sum of flows flowing on arcs in $E^S(f)$ and flows passing through nodes in $N^S(f)$, is no more than the available planes given by *fleet-size*(*f*). Observe that to compute the total number of planes of a particular fleet type *f* used in a fleet schedule, we sum the flow of planes of that fleet type on *overnight* connection arcs and through *red-eye* flight node copies.

In the data supplied by United Airlines, there were 1,609 flight legs and 13 fleet types. If we use only 3 flight node copies for each flight leg, the resulting IP formulation had approximately 1 million integer variables, and 18,000 constraints, which is far beyond the capability of commercial IP solvers. We have therefore developed a neighborhood search algorithm for the ctFAM-TW which can solve it to near-optimality within a few minutes of computer time.

5.3 Neighborhood Search Algorithm for ctFAM-TW

A neighborhood search algorithm starts with a feasible solution and successively improves it. For any feasible solution of (x, y) of ctFAM-TW, let $\mathcal{N}(x, y)$ denote the set of neighboring solutions. There are three primary steps involved in designing a neighborhood search algorithm for ctFAM-TW: (i) creating an initial feasible solution (x, y); (ii) defining the neighborhood $\mathcal{N}(x, y)$ with respect to the solution (x, y); and (iii) searching the neighborhood $\mathcal{N}(x, y)$ to identify an improved solution. We shall now discuss these three steps in greater detail.

5.3.1 Obtaining an Initial Feasible Solution

Ahuja et al. [2001c] described a method to obtain an initial feasible solution of ctFAM by first solving fleet assignment model (FAM) and then solving the through assignment model (TAM). Recall that ctFAM is a special case of ctFAM-TW where all the flights depart at their scheduled departure time. Thus, the initial feasible solution for ctFAM can also serve as an initial feasible solution for ctFAM-TW.

5.3.2 *A-B* Solution Graph

The *A-B solution graph*, $S^{AB}(x, y)$, is a subgraph of the connection graph $G = (N, A)$. It is defined for a pair of fleet types A and B and with respect to the solution (x, y), and through connections y. We first define the flight legs that are represented in $S^{AB}(x, y)$ as the following:

$$L^A = \{l : l \in L \text{ and } \sum_{l_w \in K(l)} y^A_{l_w} = 1\}$$
$$L^B = \{l : l \in L \text{ and } \sum_{l_w \in K(l)} y^B_{l_w} = 1\}$$

The sets L^A and L^B denote the sets of flight legs that are assigned fleet type A and B, respectively, in the solution (x, y). We define the node set, $N(S^{AB}(x, y))$, and arc set, $A(S^{AB}(x, y))$, of the *A-B* solution graph $S^{AB}(x, y)$ as follows:

$$N(G^{AB}(x, y)) = \{K(l) : \sum_{l_w \in K(l)} y^A_{l_w} = 1 \text{ or } \sum_{l_w \in K(l)} y^B_{l_w} = 1\}$$
$$A(S^{AB}(x, y)) = \{(i_w, j_v) \in A, x^A_{i_w j_v} = 1 \text{ or } x^B_{i_w j_v} = 1\}$$

In other words, the *A-B* solution graph $S^{AB}(x, y)$ is the subgraph of G whose node set comprises of flight node copies of flight legs that are assigned fleet type A or B in the solution (x, y), and arc set comprises of connections between those flight node copies. We shall refer to a node l_w in $S^{AB}(x, y)$ as an *A-node* if flight leg l is assigned fleet type A, and *B-node* if the flight leg l is assigned fleet type B. Especially, we refer to the flight node

copy that represents the actual departure time of a flight leg as *real node*, and use $\overline{l}_w$ to denote the real node of leg l, i.e., $y^A_{\overline{l}_w} = 1$ or $y^B_{\overline{l}_w} = 1$. Consequently, nodes in $S^{AB}(x, y)$ other than real nodes are called *virtual nodes*. If l_w is a virtual node, there is $y^A_{l_w} = 0$ and $y^B_{l_w} = 0$. We shall refer to an arc in the *A-B* solution graph as an *A-arc* if $x^A_{\overline{i}_w\overline{j}_v} = 1$, and *B-arc* if $x^B_{\overline{i}_w\overline{j}_v} = 1$. Note that arcs in $S^{AB}(x, y)$ only connect real nodes.

Figure 5-2 shows a part of the solution graph $S^{AB}(x, y)$. We illustrate this case at a city where *i*, *j*, *k* and *l* are inbound flight legs, and *s*, *t*, *u* and *v* are outbound flight legs. We show *A* nodes and *A* arcs using solid lines, and *B* nodes and *B* arcs using dashed lines. Real nodes are represented by thick lines, and virtual nodes are represented by thin lines.

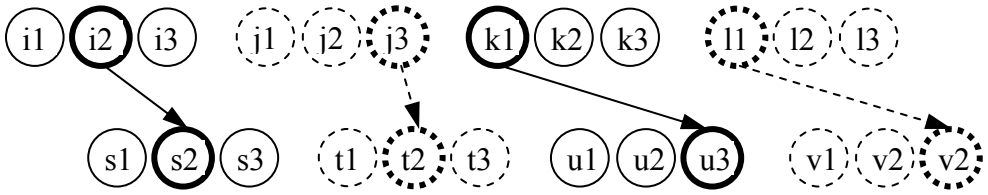


Figure 5-2. An example of the solution graph at a city with arrival flight legs *i*, *j*, *k*, *l*, and departure flight legs *s*, *t*, *u*, *v*.

5.3.3 *A-B* Swaps

Our neighborhood search structure uses the similar concept of *A-B* swaps described in Ahuja et al [2001c] to define neighboring solutions, with the extra feature that we allow flight legs to switch departing times. Given a feasible solution (x, y) of ctFAM-TW, and a pair of fleet types *A* and *B*, an *A-B* swap is defined as the follows. In the solution graph $S^{AB}(x, y)$, change some *A*-nodes to *B*-nodes, some *B*-nodes to *A*-nodes, switch some real nodes to virtual nodes and virtual nodes to real nodes, and also change

the *A*-arcs and *B*-arcs accordingly so that the modified solution is feasible and does not use more planes of fleet types *A* and *B* than used before the change.

In this chapter, we will use this *A-B* swap in a local improvement algorithm, where we always find profitable *A-B* swaps to improve the solution until no such swaps can be identified. We declare a solution (x, y) to be locally optimal when there is no profitable *A-B* swap for any combination of fleet types *A* and *B*. We illustrate an *A-B* swaps in Figure 5-3, where the flight legs *i* and *j* arrive at a certain city, and legs *s* and *t* depart at the same city Flight legs *i* and *j* depart at city1 and arrive at city 2. Figure 5-3 (a) shows the part of solution graph before swap takes place. And Figure 5-3 (b) shows a simple swap opportunity where flight leg *i* is re-fleeted from *A* to *B*, and re-scheduled at *dep-time*(i_3); flight leg *j* is re-fleeted from *B* to *A*, and re-scheduled at *dep-time*(j_2). Further, the connections are modified so that we only have connections between flight legs that are assigned the same fleet type.

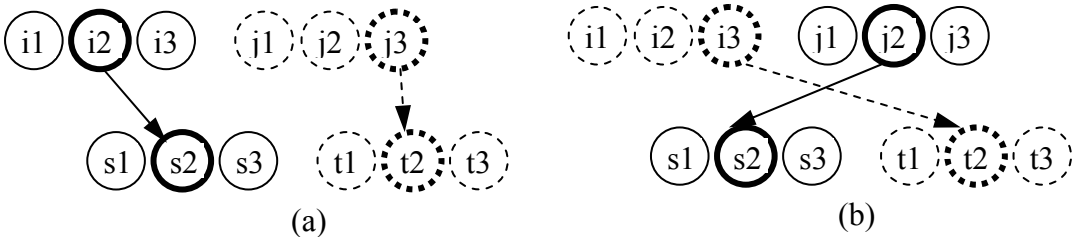


Figure 5-3. A simple example of *A-B* swap where flight legs *i* and *j* are re-fleeted and re-scheduled. (a) The solution network before the swap. (b) The solution network after the swap.

The example illustrated in Figure 5-3 is a very simple example of the *A-B* swaps. Our neighborhood structure allows much more complicated *A-B* swaps. However, there are two potential difficulties in identifying *A-B* swaps. First, identifying an *A-B* swap is a non-trivial problem. In addition, there may be too many *A-B* swaps and explicit search of these swaps to find a profitable swap may be computationally too expensive. We next

define the concept of *A-B* improvement graph which allows us to enumerate and identify profitable *A-B* swaps quite efficiently in practice.

5.3.4 *A-B* Improvement Graph

Before we discuss the construction of the improvement graph, we note that the *A-B* solution graph satisfies the following cycle-based property: The solution graph as restricted to the *A*-nodes is a union of node-disjoint cycles and isolated nodes. In other words, each real *A*-node l_w has exactly one outgoing real arc and exactly one incoming real arc, and both these arcs are incident on real nodes. In our swaps, we will be changing some *A* nodes to *B* nodes, some real nodes to virtual nodes, and vice-versa. We will construct our *A-B* improvement graph in such a way that an improving cycle leads to a new solution with the above cycle based property.

The *A-B improvement graph*, $G^{AB}(x, y)$, is derived from the *A-B* solution graph $S^{AB}(x, y)$. The node set of $G^{AB}(x, y)$ is identical to that of $S^{AB}(x, y)$. Each arc (i_w, j_v) in the *A-B* improvement graph $G^{AB}(x, y)$ signifies that we switch the fleet types of *i* and *j* from *B* to *A* or from *A* to *B* (whichever is applicable), depart the flight legs at *dep-time*(i_w) and *dep-time*(j_v), respectively, and reconnect the involved flight legs so that the connections between flights that are assigned the same fleet types. In our approach, we add an arc (i_w, j_v) to the improvement graph whenever this change can be feasibly made without increasing the total plane count at the city *arr-city*(*i*) if i_w is *A* node or *dep-city*(*i*) if i_w is a *B* node. We define the cost $c_{i_w j_v}$ of the arc (i_w, j_v) to be the negative of the change in the fleeting and through contribution resulting from the re-fleeting and reconnection.

The *A-B* improvement graph satisfies the property that each directed cycle in it, which satisfies some constraints called the *validity constraints,* corresponds to an *A-B* swap with respect to the solution (x, y), and the cost of the directed cycle equals the negate of the change in the fleeting and through connection contribution attributed to the swap. Consequently, a negative cost directed cycle satisfying the validity constraints gives a profitable *A-B* swap. We will subsequently refer to a directed cycle in $G^{AB}(x, y)$ satisfying validity constraints as a *valid cycle*.

There are six types of arcs that can be added to the improvement graph. The detailed explanation of these arcs is given next.

Type 1 Arcs:

Figure 5-4 gives an example of type 1 arc in the improvement graph. Consider an *A* arc (i_2, s_2) in the *A-B* solution graph, as shown in Figure 5-4 (a). We introduce the arc (i_3, s_1) in the improvement graph, as shown in Figure 5-4 (c), which corresponds to switching the plane types of both the flight legs *i* and *s* from *A* to *B*, and changing their departure time to *dep-time*(i_3) and *dep-time*(s_1) respectively. Figure 5-4 (b) shows the solution graph after the change takes place. Since both flight legs are still assigned the same fleet type, they maintain their connection. The cost of the arc, $c_{i_3 s_1} = (c_i^A - c_i^B) + (d_{is}^A - d_{is}^B)$, equals the sum of (i) the change in the fleeting contribution when plane type of flight leg *i* is changed from *A* to *B*, and (ii) the change in the through contribution due to the change in the fleet types. Notice that when computing $c_{i_3 s_1}$, we include the change in the fleeting contribution of flight leg *i* only but not flight leg *s*. We do it because including the change in the fleeting contribution of both the flight legs *i* and *s* will result in double counting of the fleeting costs when we sum the costs of arcs in a

valid cycle. We point out that we add the arc (i_2, s_2) to the improvement graph only if the corresponding change does not increase the number of planes of type *A* and *B*. Therefore, $RE(i_3, B) + RE(s_1, B) + ON(i_3, s_1, B) = 0$ must hold for this example.

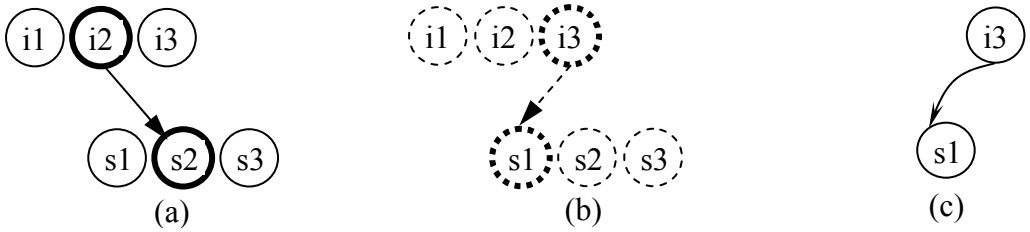


Figure 5-4. An example of the type 1 arc in the improvement graph. (a) Before the change. (b) After the change. (c) The corresponding arc in the improvement graph.

Type 2 Arcs:

We introduce a type 2 arc to the improvement graph for some *B* arcs in the solution graph. An example of type 2 arc, (t_3, j_1), is shown in Figure 5-5 (c). It corresponds to switching the plane types of both the legs *j* and t from *B* to *A*, changing their departure times to $dep\text{-}time(j_1)$ and $dep\text{-}time(t_3)$ respectively, and preserving then connection between the two flights. The part of solution graph before and after the change is shown in Figure 5-5 (a) and (b). The cost of the arc is calculated as $c_{t_3 j_1} = (c_t^B - c_t^A) + (d_{jt}^B - d_{jt}^A)$, which denotes the change in the fleeting contribution of flight leg *t* and through contribution of connection (*j*, *t*). Notice that contrary to the case of type 1 arcs, we introduce the arc (t_3, j_1) instead of (j_1, t_3). The arcs are reversed as per the discussion above. The condition of $RE(j_1, A) + RE(t_3, A) + ON(j_1, t_3, A) = 0$ should be satisfied so that fleet size of type *A* planes will not increase due to the change.

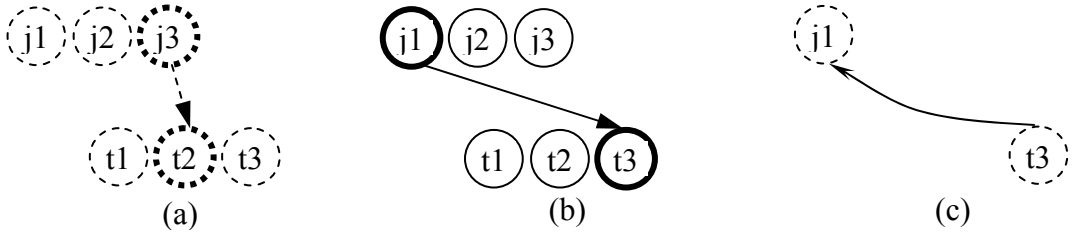


Figure 5-5. An example of the type 2 arc in the *A-B* improvement graph. (a) Before the change. (b) After the change. (c) The corresponding arc in the improvement graph.

Type 3 Arcs:

A type 3 arc is introduced to the improvement graph between two *A* nodes, similar to type 1 arc. But it is different from type 1 arc in that the corresponding flight legs are not connected in the solution graph. An example of type 3 arc (i_1, u_1) is given in Figure 5-6 (c). This arc signifies switch the fleet types of both legs *i* and *u* from *A* to *B*, and changing their departure time to *dep-time*(i_1) and *dep-time*(u_1) respectively. Since we can only make connections between flights flown by the same fleet type, this change requires changing the connections too. We thus need to reconnect flight legs *i* with *u* and *k* with *s*, as shown in Figure 5-6 (a) and (b). The cost of the arc (i_1, u_1), which is calculated as $c_{i_1u_1} = (c_i^A - c_i^B) + (d_{is}^A + d_{ku}^A - d_{iu}^B - d_{ks}^A)$, captures the change in the fleeting contribution of flight leg *i* and the change in the through contribution due to the reconnections. The following conditions must be satisfied to make this change feasible:

$$RE(i_2, A) + RE(u_3, A) + ON(i_2, s_2, A) + ON(k_1, u_3, A) \geq ON(k_1, s_2, A),$$
$$RE(i_1, B) + RE(u_1, B) + ON(i_1, u_1, B) = 0.$$

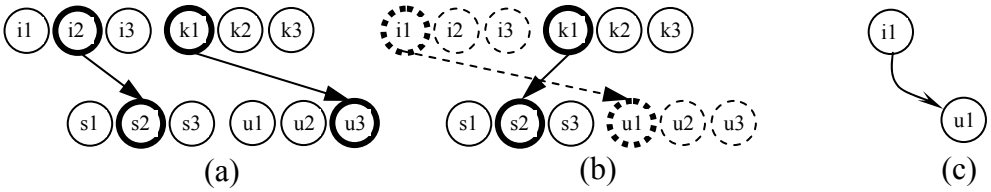


Figure 5-6. An example of the type 3 arc in the *A-B* improvement graph. (a) Before the change. (b) After the change. (c) The corresponding arc in the improvement graph.

Type 4 Arcs:

A type 4 arc is introduced to the improvement graph between two *B* nodes, for which the corresponding flight legs should meet at the same city without being connected. Figure 5-7 shows an example of type 4 arc and the related change in the solution graph. The arc (u_2, j_2) signifies switching the fleet types of both flight legs *j* and *v* from *B* to *A*, and change their departing time to *dep-time*(j_2) and *dep-time*(u_2) respectively. Also we reconnect flight legs *j* with *u* and *l* with *t* after the refleeting, as shown in Figure 5-7 (a) and (b). The cost of the arc (u_2, j_2), which is calculated as $c_{u_2 j_2} = (c_u^B - c_u^A) + (d_{jt}^B + d_{lu}^B - d_{ju}^A - d_{lt}^B)$, captures the change in the fleeting contribution of flight leg u and the change in the through contribution due to the reconnections. To ensure that the number of planes of type A and B do not increase, the following conditions must hold:

$$RE(j_2, A) + RE(u_2, A) + ON(j_2, u_2, A) = 0,$$
$$RE(j_1, B) + RE(u_1, B) + ON(j_1, t_2, B) + ON(l_3, u_1, B) \geq ON(l_3, t_2, B).$$

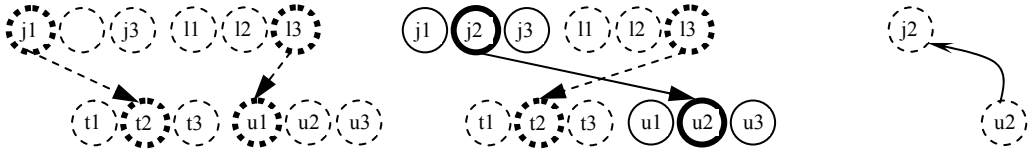


Figure 5-7. An example of the type 4 arc in the *A*-*B* improvement graph. (a) Before the change. (b) After the change. (c) The corresponding arc in the improvement graph.

Type 5 Arcs:

A type 5 arc is introduced to the improvement graph between an *A* node and *B* node, and the corresponding flight legs arrive at the same city. An example of type 5 arc, (i_2, j_3), is shown in Figure 5-8. This arc corresponds to switching the fleet type of leg *i* from *A* to *B* and of leg *j* from *B* to *A*, as shown in Figure 5-8 (a) and (b). In this example,

the departure times of both legs will remain the same. Changes in the fleet types require changing the through assignments too: leg *i* connects to leg *t*, leg *j* connects leg *s* after the change. The cost of the arc (i_2, j_3), which is calculated as

$c_{i_2 j_3} = (c_i^A - c_i^B) + (d_{is}^A + d_{jt}^B - d_{it}^B - d_{js}^A)$. To ensure that the number of planes of type A and B do not increase, the following must hold:

$$RE(i_2, A) + ON(i_2, s_2, A) \geq RE(j_3, A) + ON(j_3, s_2, A),$$
$$RE(j_3, B) + ON(j_3, t_2, B) \geq RE(i_2, B) + ON(i_2, t_2, B).$$

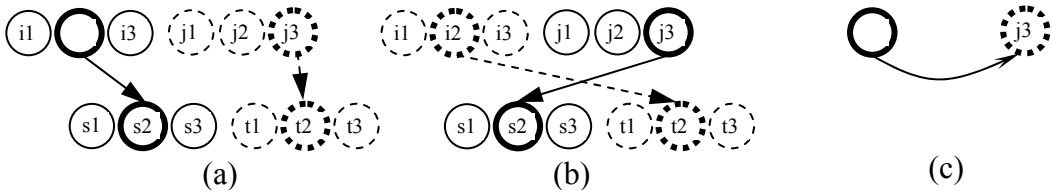


Figure 5-8. An example of the type 5 arc in the *A*-*B* improvement graph. (a) Before the change. (b) After the change. (c) The corresponding arc in the improvement graph.

Type 6 Arcs:

A type 6 arc is similar to a type 5 arc in the sense that it is also between an *A* node and *B* node but with its orientation reversed, and both the corresponding flight legs should have the same departure city. Figure 5-9 (c) gives such an example of type 6 arc (t_2, s_1), which signifies changing the fleet type of leg *s* from *A* to *B* and of leg *t* from *B* to *A*. At the same time, as shown in Figure 5-9 (a) and (b), after the change departure time of leg *s* will be switched to *dep-time*(s_1), and new connections will be made from leg *i* to leg *t* and from leg *j* to leg *s*. The cost of the arc (i_2, j_3) is calculated as

$c_{i_2 j_3} = (c_t^B - c_t^A) + (d_{is}^A + d_{jt}^B - d_{it}^A - d_{js}^B)$. To ensure that the number of planes of type A and B do not increase, the following conditions must hold:

$$RE(s_2, A) + ON(i_2, s_2, A) \geq RE(t_2, A) + ON(i_2, t_2, A),$$
$$RE(t_2, B) + ON(j_3, t_2, B) \geq RE(s_1, B) + ON(j_3, s_1, B).$$

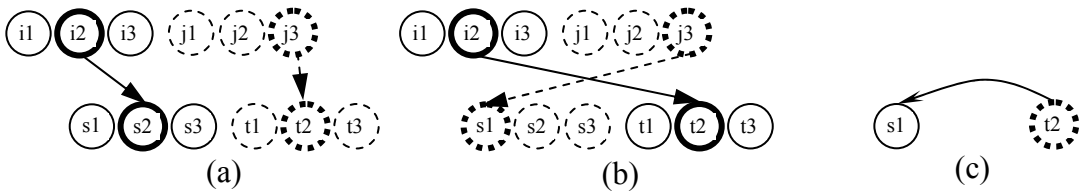


Figure 5-9. An example of type 6 arc in the *A-B* improvement graph. (a) Before the change. (b) After the change. (c) The corresponding arc in the improvement graph.

We will identify *A-B* swaps by defining valid cycles which we define next. For any feasible solution (x, y), we know that each flight leg i is connected to a unique leg j through the arc $(\overline{i}_w, \overline{j}_v)$ in the solution graph (recall that $\overline{i}_w$ and $\overline{j}_v$ are real nodes for leg i and j respectively), and is also connected to a unique leg k through the arc $(\overline{k}_v, \overline{i}_w)$. For each node i_w in the *A-B* improvement graph, we have the following definition:

$$CONNECT\text{-}FROM(i_w) = \{k_v : k_v \in K(k), \text{ leg } i \text{ is connected from leg } k \text{ at } dep\text{-}city(i)\}$$
$$CONNECT\text{-}TO(i_w) = \{j_v : j_v \in K(j), \text{ leg } i \text{ is connected to leg } i \text{ at } arr\text{-}city(i)\}$$

CONNECT-FROM(i_w) denotes the set of flight node copies that represent the flight leg from which leg i is connected at its departure city in the solution (x, y). *CONNECT-TO*(i_w) denotes the set of flight node copies that represent the flight leg to which leg i is connected to at its arrival city in the solution (x, y). If i_w is a A node, then we also call *CONNECT-TO*(i_w) as *MATE*(i_w), otherwise, we call *CONNECT-FROM*(i_w) as *MATE*(i_w).

Valid Cycles*: A directed cycle W in the A-B improvement graph is said to be a valid cycle if it satisfies the following property for each node $i_w \in W$: for all $j_v \in MATE(i_w)$, $j_v \notin W$ unless $(i_w, j_v) \in W$.*

Theorem 5-1. *Each valid cycle in the A-B improvement graph $G^{AB}(x, y)$ gives an A-B swap respect to the solution (x, y).*

Proof: This proof is similar to the proof given in Ahuja et al. [2001c] and is omitted.♦

The intuitive reason we do not allow valid cycles to contain both the node i_x and j_x that $j_v \in MATE(i_w)$ in the valid cycle unless $(i_w, j_v) \in W$ is as follows. The purpose of constructing the improvement graph is that a directed cycle in it defines a feasible *A-B* swap and that the cost of the cycle equals the negative of increase in the contribution resulting from the *A-B* swap. A directed cycle, which is not a valid cycle, cannot ensure this property. Consider, for example, a directed cycle W in the improvement graph contains a type 5 arc (i_2, j_3) (see Figure 5-4). According to our definition, s_1, s_2 and $s_3 \in MATE(i_2)$. The arc (i_2, j_3) signifies the changing of fleet type for flight leg j from B to A, and reconnects to leg s, which is connected from leg i before swap. If we allow the cycle W to visit s_1, s_2, or s_3, with any one of the six types of arcs, then the fleet type of flight leg s will be changed from A to B, and thus we will not be able to preserve the connection from leg j to s and its cost become incorrect. Therefore, if we make arc (i_2, j_3) part of the cycle, then we must disallow the nodes in $MATE(i_2)$, and due to the same reason we also must disallow nodes in $MATE(j_3)$. This difficult arises when we include arcs of type 3, 4, 5, or 6 in the cycle W. This difficulty does not arise when we make an arc of type 1 or type 2 to be the part of the cycle in which case in clued both the node i_w and one of its mate. Hence we introduce the "***unless***" clause in the definition of the valid cycle.

We will now give an illustrative example that a valid cycle in the improvement graph gives a feasible A-B swap. Consider the part of A-B solution graph shown in Figure 5-10 (a). When we construct the A-B improvement graph, it will contain the valid cycle W=(l1→i1→j3→h2→l1) shown in Figure 5-10 (b). This cycle denotes the *A-B* swap, which when performed, produces the solution shown in Figure 5-10 (c). Observe

that all the nodes in the cycle switch their fleeting types, and probably departure time. The arc (l1, i1) in the valid cycle W is a type 3 arc, this arc signifies that leg l and i switch their fleeting types, change departure time, and connect to/from each other. The next arc (i1, j3) in the cycle is a type 5 arc which changes fleeting types, departure time, and also connections. The next arc (j3, h2) is a type 2 arc; it only changes fleeting and departure time. Finally, the arc (h2, l1) is a type 6 arc, which causes legs h and l to be re-fleeted, re-scheduled, and re-connected. Figure 5-10 (c) shows the same part of the solution graph when the corresponding *A*-*B* swap has been performed.

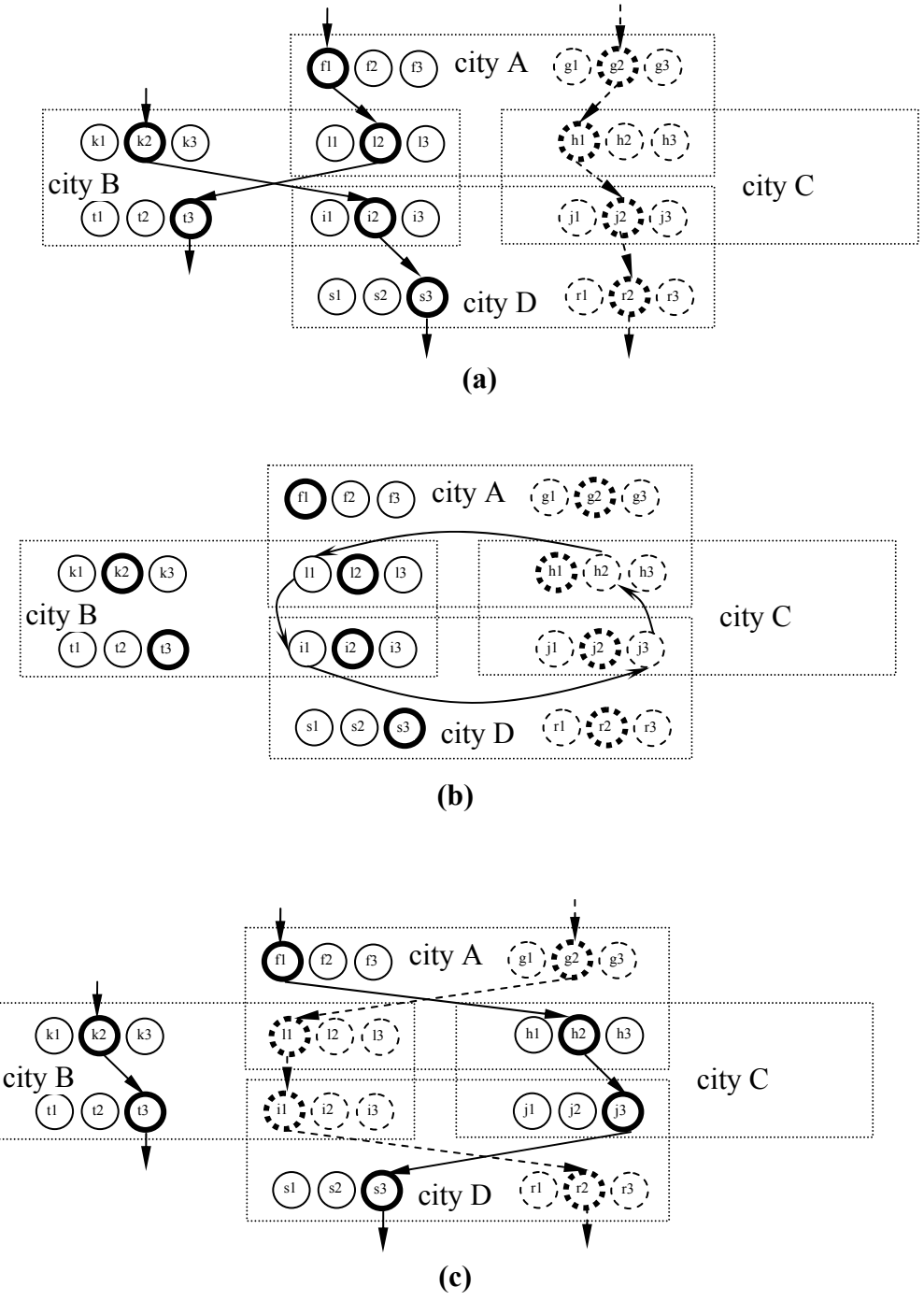


Figure 5-10. An illustrative example of valid cycle and *A-B* swap. (a) Solution network before the swap takes place. (b) A valid cycle in the A-B improvement graph. (c) Solution network after the swap takes place.

5.3.5 Identifying *A-B* swaps

In the last section, we have shown that we can identify *A-B* swaps by identifying valid cycles in the *A-B* improvement graph. However, identifying valid cycles in a graph is NP-complete problem (Ahuja et al. [2001]). Fortunately, this problem can typically be solved in a fraction of second using CPLEX in our benchmark case. We will next model the problem of finding a union of node-disjoint valid cycles as an integer problem.

We first introduce some notation related to the integer program. Let $N' = N(G^{AB}(x, y))$ denote the set of nodes and $E' = E(G^{AB}(x, y))$ denote the set of arcs in the *A-B* improvement graph. Further, we use $C^{AB} = \{(i, j) : x^{A}_{\bar{i}_w \bar{j}_v} = 1 \text{ or } x^{B}_{\bar{i}_w \bar{j}_v} = 1\}$ to denote the set of flight leg connections with fleet types *A* and *B* in the solution (x, y). We associate a binary variable $z_{i_w j_v}$ with each arc $(i_w, j_v) \in E'$. This variable takes value 1 if arc (i_w, j_v) is present in some valid cycle, and takes value 0 otherwise. We give the IP formulation next followed by its explanation.

Objective Function:

$$\min \sum_{(i_w, j_v) \in E'} c_{i_w j_v} z_{i_w j_v} \tag{5.2a}$$

Constraints:

$$\sum_{\{j_v : (j_v, i_w) \in E'\}} z_{j_w i_v} - \sum_{\{j_v : (i_w, j_v) \in E'\}} z_{i_w j_v} = 0, \quad \text{for all } i \in N' \tag{5.2b}$$

$$\sum_{\substack{\{(i_w, l_v):\\ i_w \in K(i), j_v \notin K(j)\}}} z_{i_w l_v} + \sum_{\substack{\{(i_w, j_v):\\ i_w \in K(i), j_v \in K(j)\}}} z_{i_w j_v} + \sum_{\substack{\{(r_w, j_v):\\ r_w \notin K(i), j_v \in K(j)\}}} z_{r_w j_v} \le 1, \quad \text{for all } (i, j) \in C^{AB} \tag{5.2c}$$

$$z_{i_w j_v} \in \{0,1\}, \quad \text{for all } (i_w, j_v) \in E' \tag{5.2d}$$

In the above formulation (5.2), the constraints (5.2b) and (5.2d) imply that the solution is a 0-1 circulation. This 0-1 circulation can be decomposed into unit flows along directed cycles. The constraints (5.2c) ensures that (i) for each flight leg $i \in L^A \cup L^B$, at most one flight node copy can present in those valid cycles; and (ii) the flow passing through each node i_w plus the flow passing through any of the nodes in $MATES(i_w)$ is at most 1 implying that the resulting flow will not pass through i_w and any node in $MATES(i_w)$ simultaneously. An exception to this rule occurs when flow takes place over the an arc (i_w, j_v) and $j_v \in MATES(i_w)$, in which case both the nodes i_w and j_v can be visited. It is easy to see that a feasible solution of (5.2) gives a set of valid cycles. If the improvement graph does not contain any negative cost valid cycles, then $z = 0$ will be an optimal solution of (5.2). If the improvement graph contains a negative cost cycle, then an optimal solution z^* of (5.2) will give a collection of valid cycles with the minimum total cost. Using flow decomposition (see, for example, Ahuja, Magnanti and Orlin [1993]), we can decompose z^* into a set of node-disjoint cycles. Each of these cycles either has a negative cost or zero cost. The negative cost cycles include an associated profitable *A-B* swap.

5.3.6 Neighborhood Search Algorithm

We are now in a position to describe our neighborhood search algorithm for ctFAM-TW. The algorithm first obtains an initial feasible solution of ctFAM-TW using the method described in Section 5.3.1. The algorithm then considers every pair of fleet types A and B and performs the following steps: It constructs the A-B solution graph $S^{AB}(x, y)$ followed by the A-B improvement graph $G^{AB}(x, y)$. It solves the IP (5.2) to

determine a set of negative cost valid cycles in $G^{AB}(x, y)$. If negative cost valid cycles are found, then it performs the corresponding *A*-*B* swaps, updates the solution graph and improvement graph, and resolves the IP (5.2) to identify another set of negative cost valid cycles. This process is repeated until the improvement graph has no negative cost valid cycles, at which point we consider another pair of fleet types A and B. The algorithm terminates when for any pair of fleet types A and B, the A-B improvement graph does not contain a negative valid cycle.

5.4 Computational Testing

In this section, we present computational results for our neighborhood search algorithm. We programmed our algorithm in the C++ programming language on a Pentium 2.4 GHz processor computer with 1 GB RAM and a Windows 2000 Professional operating system. We tested our algorithms on the data provided by United Airlines, and Table 5-2 gives the computational results. The first result with zero time window width gives the solution of ctFAM. It can be observed that there is large improvement in revenue generated even with 10 minute time window and 3 flight node copies.

Table 5-2. Improvements generated by ctFAM-TW neighborhood search algorithm

Time Window Width	No. of Flight Node Copies	CPU Time	Annual Revenue Improvement Generate
0 min	1	16 sec	$26.5 million
10 min	3	62 sec	$41.9 million
10 min	5	126 sec	$42.1 million
20 min	3	63 sec	$63.4 million
20 min	5	128 sec	$63.4 million
30 min	3	62 sec	$51.9 million
30 min	5	157 sec	$64.9 million

5.5 Conclusions

In this chapter, we study the ctFAM-TW which incorporates time windows with ctFAM. We give an integer programming formulation of ctFAM-TW, which,

unfortunately, is too large to be solved to optimality or near optimality using the state-of-art commercial MIP solvers. We extend the VLSN search algorithm for ctFAM and apply the algorithm on ctFAM-TW. This approach generates revenue improvement of millions of dollars according to the real-life data provided by a major U.S. airline.

CHAPTER 6
FURTHER REASERCH AND CONCLUDING REMARKS

6.1 Introduction

As described in Chapter 1, there are numerous transportation scheduling problems that need attentions from the OR community. We have attempted to solve a few of such problems. Our efforts are focused in finding good quality and implementable solutions in reasonable computational time. In this chapter, we first summarize the findings for the four transportation scheduling problems that we have studied and then briefly present the direction for the future research.

6.2 Summary of Findings

In Chapter 2, we developed model and algorithm to solve the locomotive scheduling problem. Our approach can offer considerable savings in cost, especially in terms of a significant reduction in the number of locomotives needed. Our model incorporates a set of constraints and business rules specified by the railroad industry. It is the first approach in literature to account for consist-busting, light travel, and the consistency of solutions in a unified framework.

In Chapter 3, we studied the railroad blocking problem. We developed integer programming (IP) and very large scale neighborhood search (VLSN) based algorithms for this problem. Our VLSN approach has been able to incorporate many practical constraints that are essential for an implementable blocking policy. Our algorithms achieve savings both in number of intermediate car-handlings and car-miles. The

solutions obtained by our algorithms have potential of monetary savings in millions of dollars for both CSX Transportation and BNSF Railways.

In Chapter 4, we developed VLSN search algorithms to solve the multi-criteria combined through and fleet assignment problem. The next generation airline planning problems are focused on the integration between business areas such as aircraft scheduling and crew scheduling. Since explicit joint optimization becomes computationally intractable, an attractive alternative is to formulate multi-criteria optimization problems. The VLSN search heuristic for this multi criteria ctFAM has provided a robust and accurate approach to achieve integration while managing the complexity in an effective manner. The outcome of the research has been proved very promising.

In Chapter 5, we extended the algorithms in Chapter 4 to solve ctFAM with additional feature of time windows (ctFAM-TW). It is observed that the flexibility of flight departure time allows more through connection opportunities, and leads to more efficient aircraft utilization and fleet assignment. By altering the departure times of flight legs, ctFAM-TW produces solutions with distinct advantage in monetary terms over the ctFAM solutions.

6.3 Proposed Research

In Chapter 2, we have used a sequential heuristic procedure for determining three sets of decision variables (i) light travels, (ii) train-train connections, and (iii) active and deadhead locomotives. Although we have achieved a certain level of integration in our sequential approach, further research is required to obtain a better coordination of these different set of variables. CSX Transportation is interested in incorporating locomotive fueling and servicing constraints along with constrains we have already incorporated in

our algorithm. CSX Transportation also intends to introduce decision support tools into its tactical management process.

In Chapter 3, we have developed algorithms to solve the railroad blocking problem for single traffic type (general merchandize). However, railroads transport freight of many types (for example, automotives, grains, coal, etc.). Further research is required to develop an integrated model for solving blocking problem for all traffic types. Development of a flexible and user friendly decision support system with numerous what-if analysis is highly desirable by railroads.

In Chapter 4, we have incorporated ground manpower planning and crew planning requirements in the objective function of ctFAM, and the results obtained by our algorithms are very conducive for the subsequent airline scheduling stages of ground manpower scheduling and crew scheduling. There are many such criteria that airlines want to integrate in their scheduling process. The inclusion of these criteria in objective function achieves a certain level of integration. Further research is required to develop scheduling models incorporating many such criteria. In our algorithms, we have considered two multi-criteria problems, and each problem considers one criterion as secondary objective. Research can be done to develop unified model which considers both secondary objectives simultaneously.

In Chapter 5, we have considered ctFAM with time windows and in Chapter 4 we have developed algorithms for multi-criteria ctFAM. Further work can be done to integrate these two models, i.e., development of algorithms to solve multi-criteria ctFAM with time windows.

LIST OF REFERENCES

Aarts, E. H. L., and J. K. Lenstra, J. K. (editors). 1997. Local Search in Combinatorial Optimization. John Wiley & Sons, New York, NY.

Abarra, J. 1989. Applying integer linear programming to the fleet assignment problem. *Interfaces* **19**, 20-28.

Ahuja, R. K., T. L. Magnanti, and J. B. Orlin. 1993. Network Flows: Theory, Algorithms, and Applications. Prentice Hall, NJ.

Ahuja, R. K., O. Ergun, J. B. Orlin, and A. P. Punnen. 2000. A survey of very large scale neighborhood search techniques. To appear in *Discrete Applied Mathematics*.

Ahuja, R. K., J. B. Orlin, and D. Sharma. 2001a. Multi-exchange neighborhood search algorithms for the capacitated minimum spanning tree problem. *Mathematical Programming* **91**, 71-97.

Ahuja, R. K., J. B. Orlin, and D. Sharma. 2001b. A composite very large-scale neighborhood structure for the capacitated minimum spanning tree problem. Submitted to *Operations Research Letters*.

Ahuja, R. K., J. B. Orlin, and D. Sharma. 2001c. A very large-scale neighborhood search algorithm for the combined through-fleet assignment model. Submitted to *INFORMS Journal on Computing*.

Association of American Railroads web site: http://www.aar.com.

Assad, A. A. 1981. Analytical models in rail transportation: an annotated bibliography. *INFOR* **19**, 59-80.

Assad, A. A. 1983. Analysis of rail classification policies. *INFOR* **21**, 293-314.

Bard, J. F., and C.A. Hopperstad. 1987. Improving through-flight schedules. *IIE Transactions* **19**, 242-251.

Barnhart, C., and K. T. Talluri. 1997. Airlines operations research. In *Design and Operation of Civil and Environmental Engineering Systems*, Edited by A. McGarity and C. ReVellepp, 435-469.

Barnhart, C., N. L. Boland, L. W. Clarke, E. L. Johnson, G. L. Nemhauser, and R. Shenoi. 1998. Flight string models for aircraft fleeting and routing. *Transportation Science* **32,** 208-219.

Barnhart, C., H. Jin, and P. H. Vance. 2000. Railroad blocking: a network design application. *Operations Research* **48**, 603-614.

Bodin, L. D., B. L. Golden, A.D. Schuster and W. Romig. 1980. A model for the blocking of trains. *Transportation Research* **14B**, 115-120.

Brannlund, U., P. O. Lindberg, A. Nou, and J. -E. Nilsson. 1998. Railway timetabling using Lagrangian relaxation. *Transportation Science* **32,** 358-369.

Campbell, K. C. 1996. Booking and Revenue Management for Rail Intermodal Services. PhD dissertation, Department of Systems Engineering, University of Pennsylvania, Philadelphia, PA.

Caprara, A., Fischetti, M., Toth, P., Vigo, D. and P. L. Guida. 1997. Algorithms for railway crew management. *Mathematical Programming* **79**, 125-141.

Chih, K. C., M.A. Hornung, M.S. Rothenberg, and A. L. Kornhauser. 1990. Implementation of a real time locomotive distribution system. In *Computer Applications in Railway Planning and Management,* T. K. S. Murthy, R. E. Rivier, G. F. List and J. Mikolaj (eds.) Computational Mechanics Publications, Southampton, U.K., 39-49.

Clarke, L. W., C.A. Hane, E. L. Johnson, and G. L. Nemhauser. 1996. Maintenance and crew considerations in fleet assignment. *Transportation Science* **30,** 249-260.

Cordeau, J.-F., P. Toth, and D. Vigo. 1998. A survey of optimization models for train routing and scheduling. *Transportation Science* **32,** 988-1005.

Crainic, T. G., M. Gendreau, and P. Dejax. 1990. Modelling the container fleet management problem using a stochastic dynamic approach. In *Operational research'90*, H. E. Bradley (ed.), Pergamon Press, New York, 473-486.

Farvolden, J. M. and W. B. Powell. 1994. Subgradient Methods for the service network design problem. *Transportation Science* **28**, 256-272.

Fischetti, M. and P. Toth. 1997. A package for locomotive scheduling, Technical Report DEIS-OR-97-16, University of Bologna, Italy.

Florian, M. G. Bushell, J. Ferland, G. Guerin, and L. Nastansky. 1976. The engine scheduling problem in a railway network. *INFOR* **14,** 121-138.

Forbes, M. A., J. N. Holt, and A. M. Watts. 1991. Exact solution of locomotive scheduling problems. *Journal of Operational Research Society* **42**, 825-831.

Glover, F., and M. Laguna. 1997. Tabu Search. Kluwer Academic Publishers, Norwell, MA.

Goldberg, A. V. 1995. Scaling algorithms for the shortest path problem. *SIAM Journal on Computing* **24**, 494-504.

Gopalan, R., and K. T. Talluri. 1998. Mathematical models in airline schedule planning: A survey. *Annals of Operations Research* **76**, 155-185.

GORMAN, M. F. 1995. An application of genetic and tabu searches to the freight railroad operation plan problem. INFORMS Spring 1995 Meeting.

Hane, C. A., C. Barnhart, E. L. Johnson, R.E. Marsten, G.L. Nemhauser, and G. Sigmondi. 1995. The fleet assignment problem: Solving a large-scale integer program. *Mathematical Programming* **70,** 211-232.

Haghani, A. E. 1987. Rail freight transportation: a review of recent optimization models for train routing and empty car distribution. *Journal of Advanced Transportation* **21**, 147–172.

Huntley, C. L., D. E. Brown, D. E. Sappington, and B. P. Markowics. 1995. Freight routing and scheduling at CSX transportation. *Interfaces* **25(3)**, 58-71.

Jarrah, A. I. Z., and J.C. Reeb. 1997. An optimization model for assigning through flights. Technical Document, United Airlines.

Jovanovic, D. and P. T. Harker. 1991. A decision support system for train dispatching: An optimization-based methodology, *Transportation Research Record* **1314***,* 31-40.

Keaton, M. H. 1989. Designing optimal railroad operating plans: Lagrangian relaxation and heuristic approaches. *Transportation Research* **23B**, 415-431.

Keaton, M. H. 1992. Designing railroad operating plans: A dual adjustment method for implementing Lagrangian relaxation. *Transportation Science* **26**, 263-279.

Kniker, T. S., C. Barnhart, and M. Lohatepanont. 2000. Itinerary-based airline fleet assignment. Submitted to *Transportation Science*.

Kraft, E. R. 1998. A Reservation-based Railway Network Operations Management System. PhD dissertation, Department of System Engineering, University of Pennsylvania, Philadelphia, PA.

Mao, C-K., and C. D. Martland. 1981. Utilization of railroad motive power: Train delay impacts and organizational considerations. *The Logistics and Transportation Review* **17**, 291-312.

Newton, H. N., C Barnhart, and P. H. Vance. 1998. Constructing railroad blocking plans to minimize handling costs. *Transportation Science* **32,** 330-345.

Nou, A., J. Desrosiers, and F. Soumis. 1997. Weekly locomotive scheduling at Swedish State Railways, Technical report G-97-35, GERAD, Ecole des Hautes Etudes Commercials de Montreal, Canada.

Nozick, L. K. and E. K. Morlok. 1997. A model for medium-term operations planning in an intermodal rail-truck service. *Transportation Research* **31A**, 91-107.

Powell, W. B., P. Jaillet, and A. Odoni. 1995. Stochastic and dynamic networks and routing. In *Network Routing*, M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser (eds.), North-Holland, Amsterdam, 141-295.

Smith, S. and Y. Sheffi. 1988. Locomotive scheduling under uncertain demand. *Transportation Research Record* **1251**, 45-53.

Shan, Y. 1997. Block-to-train assignment model. Technical Report. Operations Research Group, CSX Transportation, Jacksonville, FL.

Sherali, H. D., and A. B. Suharko. 1988 A tactical decision support system for empty railcar management. *Transportation Science* **32**, 306-329.

Steuer, R. E. 1986. Multi Criteria Optimization – Theory, Computation, and Application. Wiley, New York, NY.

Subramanium, R., R. P. Scheff Jr., J. D. Quillinan, D. S. Wiper, and R. E. Marsten. 1994. Coldstart: Fleet assignment at Delta Air Lines. *Interfaces* **24,** 104-120.

Talluri, K.T. 1996. Swapping applications in a daily fleet assignment. *Transportation Science* **31,** 237-248.

Van Dyke, C. D. 1986. The automated blocking model: aA practical approach to freight railroad blocking plan development. *Transp. Res. Forum* **27**, 116-121.

Van Dyke, C. D. 1988. Dynamic management of railroad blocking plans. *Tranp. Res. Forum* **29**, 149-152.

Yu, G. (ed.) 1998. Operations Research in the Airline Industry, Kluwer Academic Publishers, Boston, MA.

Ziarati, K., F. Soumis, J. Desrosiers, S. Gelinas, and A. Saintonge. 1997. Locomotive assignment with heterogeneous consists at CN North America. *European Journal of Operational Research* **97**, 281-292.

Ziarati, K., F. Soumis, J. Desrosiers, and M. M. Solomon. 1999. A branch-first, cut-second approach for locomotive assignment, *Management Science* **45**, 1156-1168.

BIOGRAPHICAL SKETCH

Jian Liu was born in Gansu, China. He is the youngest child and only boy among three children of the family. He moved to Xi'an, Shanxi, with his family when he was 14 and left for Nanjing, Jiangsu, at the age of 17. The summer he completed his high school he went to TsingHua University, Beijing. He earned his bachelor's degree in Dept. of Automation from the university in July 1997. Two years later he received his master's degree in Dept. of Electrical Engineering at the National University of Singapore, Singapore. He came to the University of Florida for his Ph.D study in the Department of Industrial and Systems Engineering in August, 1999. His doctoral research is concerned with developing the state-of-art algorithmic approaches in linear, integer, and network optimization to solve very difficult and large-scale combinatorial optimization problems which must satisfy a variety of practical constraints and business rules. Some of the awards he has received include the First Prize in the INFORMS sponsored student competition "Management Science in Railroad Applications" at the 2002 INFORMS Conference at San Jose, California; the University of Florida Award for Outstanding Academic Accomplishment; and the University of Florida Alumni Fellowship.