TRANSPORTATION SCIENCE
Vol. 42, No. 3, August 2008, pp. 263–278
ISSN 0041-1655 | EISSN 1526-5447 | 08 | 4203 | 0263

informs

DOI 10.1287/trsc.1070.0227

Per-Seat, On-Demand Air Transportation Part I: Problem Description and an Integer Multicommodity Flow Model

D. Espinoza
School of Industrial Engineering, Universidad de Chile, Santiago, Chile, daespino@dii.uchile.cl

R. Garcia
DayJet Corporation, Boca Raton, Florida 33431, renan.garcia@dayjet.com

M. Goycoolea
School of Business, Universidad Adolfo Ibáñez, Santiago, Chile, marcos.goycoolea@uai.cl

G. L. Nemhauser, M. W. P. Savelsbergh
H. Milton School of Industrial and Systems Engineering,
Georgia Institute of Technology, Atlanta, Georgia 30332
{george.nemhauser@isye.gatech.edu, martin.savelsbergh@isye.gatech.edu}

The availability of relatively cheap small jet planes has led to the creation of on-demand air transportation services in which travelers call a few days in advance to schedule a flight. A successful on-demand air transportation service requires an effective scheduling system to construct minimum-cost pilot and jet itineraries for a set of accepted transportation requests. We present an integer multicommodity network flow model with side constraints for such dial-a-flight problems. We develop a variety of techniques to control the size of the network and to strengthen the quality of the linear programming relaxation, which allows the solution of small instances. In Part II, we describe how this core optimization technology is embedded in a parallel, large-neighborhood, local search scheme to produce high-quality solutions efficiently for large-scale real-life instances.

Key words: air transportation; on-demand service; integer multicommodity flow
History: Received July 2007; revision received November 2007; accepted December 2007. Published online in *Articles in Advance* May 15, 2008.

1. Introduction

The United States Department of Transportation (DOT) reports that Americans make more than 405 million business trips of more than 50 miles each year (United States Department of Transportation 2003). Of these trips, 84% do not cross regional boundaries, and close to 20% involve travel of 250 to 1,000 miles.

Despite the length of the trips in the latter category, most of these trips are done by automobile. This phenomenon can be explained by examining the air travel alternative. To get from one regional airport to another, travelers typically have to connect through heavily congested "hub" airports, often located many miles from their origins and destinations. Missed connections and flight delays are common. Furthermore, travelers often have to drive many miles to get to or from an airport serviced by a scheduled airline. In fact, commercial flight service exists in only about 550 of the nation's 5,000 public-use airports, with a mere 67 of these airports accounting for 90% of domestic traffic. Add to this the fact that airlines offer limited schedules at regional airports, and the air travel alternative does not look very appealing.

As a consequence of these impediments, the DOT reports that the trend seems to be that more and more travelers prefer driving rather than flying for trips between 200 and 500 miles. This is in part due to the changes taking place in the commercial airline industry. Increased security at airports has resulted in longer waiting times, with the associated frustrations, and thus longer travel times. Furthermore, due to the huge losses suffered by the airlines in recent years (airlines worldwide lost $25 billion dollars and more than 400,000 jobs in 2002 and 2003), airlines have cut back and are operating with a reduced schedule, affecting the flexibility of the business traveler, especially when it concerns smaller regional airports.

Can this trend be reversed? Can an air travel alternative be developed that appeals to the regional business traveler? Some people believe so.

New developments in avionics and airplane manufacturing have brought about a new technology: the very light jet (VLJ), also called the microjet.

Weighing less than 10,000 pounds, these aircraft can carry up to five passengers, fly distances of over 1,000 miles, reach altitudes of 19,000–41,000 feet, and travel at speeds between 350–390 nautical miles per hour (almost twice the altitude and speed of current turboprop airplanes). Priced at slightly more than a million U.S. dollars, these jets cost about one-third of the price of the typical small jets sold today. Several manufacturers are taking orders for VLJs, with Eclipse Aviation Corp. (http://www.eclipseaviation.com) being the first, with deliveries in 2007.

The availability of relatively cheap small jet aircraft suggests a new air transportation business: dial-a-flight, an on-demand service in which travelers call one day or a few days in advance to schedule transportation. The advantages of such a system are obvious. This service gives regional travelers the option of using small jets that fly to and from less congested outlying airports, without packed parking lots, long lines at security checkpoints, flight delays, and lost luggage, that are closer to where they live and where they want to go. In fact, VLJs can land at nearly 5,000 of the 14,000 private and public landing strips in the United States. By charging a discount fare for sharing cabin space with other passengers, aggregation can greatly reduce costs while still ensuring a very convenient service.

The idea of a dial-a-flight service to satisfy regional demand is rapidly becoming a reality. In fact, even though VLJs are not yet available, air taxi services already exist today. Linear Air (http://www.linearair.com)[1] is providing air transportation in the northeastern United States. Alpha Flying (http://planesense.aero/whoweare.htm), Avantair (http://www.avantair.com), CitationShares (http://www.citationshares.com), and FlexJet (http://www.flexjet.com) offer fractional ownership programs, in which one can buy flying time on a fleet of aircraft.

In October of 2007 DayJet Corporation (2005) began providing *per-seat, on-demand* air transportation services in the southeast. The business model of DayJet is different from the companies mentioned above in the sense that DayJet's offerings include individual seats (per-seat on-demand) as well as entire planes (per-plane on-demand) and therefore the potential of much lower fares. Will this business model be successful? Ed Iacobucci, founder and CEO of DayJet Corporation, believes that it is not only possible to successfully run a per-seat, on-demand air service business, but it is possible to do so charging an amount only slightly above nondiscount coach fares of commercial airlines. For the business to be profitable at these rates, each revenue flight segment should average a load of around 1.3 paying passengers. And, Iacobucci says, the key to attaining such load factors is optimization-based scheduling systems.

The dial-a-flight alternative has generated a lot of interest. Recent media coverage includes *The New York Times* (Fallows 2005), *The Wall Street Journal*, *USA Today*, *Business Week*, *Newsweek*, CNN, BBC, and much, much, more.

To effectively manage day-to-day operations at a per-seat, on-demand air service, several optimization-based scheduling components need to be employed. The two key components are: (1) an online accept/reject system to quickly inform passengers whether their air transportation requests can be serviced and at what price, and (2) an offline scheduling system to construct minimum cost pilot and jet itineraries for the next day once the reservation deadline has passed.

In this two-part paper, we discuss the components of an offline scheduling system developed for and in collaboration with DayJet Corporation. The online accept/reject system involves a very rapid (less than 15 seconds) heuristic search to see if a new request can be fit in. This is a proprietary DayJet system that, unfortunately, we cannot present here. Online accept/reject systems in other contexts have been studied, among others, by Benoist et al. (2001); Bent and Van Hentenryck (2004); Campbell and Savelsbergh (2005); and Van Hentenryck, Bent, and Vergados (2006).

In Part I, we introduce an integer multicommodity network flow model with side constraints for the dial-a-flight problem, and we develop a variety of techniques to control the size of the network and to strengthen the quality of the linear programming relaxation. The resulting technology allows the solution of small-size instances. In Part II, the core optimization technology is embedded in parallel local search scheme that makes our technology scalable so that high-quality solutions can be obtained efficiently for large-scale real-life instances.

The remainder of Part I is organized as follows. In §2 we formally introduce the dial-a-flight problem, and we discuss its relation to other pickup and delivery problems encountered in the literature. In §3 we place our research in context and identify the specific contributions. In §4 we describe a novel multicommodity network flow model for the dial-a-flight problem. In §5 we discuss innovative construction and aggregation algorithms to ensure the resulting optimization problem is of manageable size. Finally, in §6, we present a computational study demonstrating the viability of the proposed approach for small instances.

2. Problem Description

The *dial-a-flight* problem (DAFP) is concerned with the scheduling of a set of requests for air transportation during a single day. A request specifies an origin

[1] Websites last verified April 29, 2008.

airport, a destination airport, an earliest acceptable departure time at the origin, a latest acceptable arrival time at the destination, and the number of passengers and their weight. Although service requirements could be handled differently, e.g., by specifying an earliest acceptable departure time and a maximum trip time or a latest acceptable arrival time and a maximum trip time, or simply a maximum trip time, DayJet's market research indicates that an earliest departure and a latest arrival time is what is preferred by business travelers. A fleet of jet airplanes is available to provide the requested air transportation. Each jet has a home base, a seat capacity limiting the number of passengers that can be accommodated, and a weight capacity limiting the weight that can be accommodated. Each jet is available for a certain period during the day and has to return to its home base at the end of the day. A set of pilots, stationed at the home bases of the airplanes, is available to fly the jets. A pilot departs from the home base where he is domiciled at the start of his duty and returns to the home base at the end of his duty. A pilot schedule has to satisfy FAA regulations governing flying hours and duty period; a pilot cannot fly more than 8 hours in a day and his duty period cannot be more than 14 hours. Pilots do not change aircraft during their duty. To ensure acceptable service, an itinerary for a passenger will involve at most two flights; i.e., at most one intermediate stop is allowed. Furthermore, if there is an intermediate stop, both flights have to be on the same jet (a safeguard to avoid waiting in case planes get delayed). A plane can deadhead (fly without passengers) to pick up passengers at another airport or to return home. The minimum time between an arrival at an airport and the next departure, called the turnaround time, is given for each airport. The objective is to minimize the costs, while satisfying all requests and respecting all constraints. A dispatcher has to decide which jets and pilots to use to satisfy the requests and what the jet and pilot itineraries will be, i.e., the flight legs and associated departure times.

We use the following notation to model the problem:

$\mathcal{A}$: the set of all airports.

$\mathcal{J}$: the set of all jets.

$\mathcal{R}$: the set of all transportation requests.

Without loss of generality, we assume that the requests are ordered, i.e., $\mathcal{R} = \{r_1, r_2, \ldots, r_s\}$. Each request $r \in \mathcal{R}$ has the following attributes:

ORIGIN(r): the origin airport where passengers are to be picked up.

DESTINATION(r): the destination airport where passengers are to be dropped off.

EARLIEST(r): earliest departure time.

LATEST(r): latest arrival time.

PASSENGERS(r): number of passengers flying together.

WEIGHT(r): total weight of passengers and luggage.

Each jet $j \in \mathcal{J}$ has the following attributes:

HOME_BASE(j): the home airport of the jet.

JET_BEGIN(j): the earliest time at which the jet can start.

JET_END(j): the latest time at which the jet can finish.

CAPACITY(j): the number of passenger seats in the jet.

MAX_FLY_TIME(j): the total daily flying time limit (in minutes) for the jet.

MAX_WEIGHT(j): the weight limit (in pounds) for the jet.

For airports $a, b \in \mathcal{A}$ we consider the quantities:

FLIGHT_COST(a, b): the cost of flying between airports a and b.

FLIGHT_TIME(a, b): the flying time (in minutes) between airports a and b.

TURN_AROUND(a): the turnaround time (in minutes) at airport a; i.e., the number of minutes which must elapse between an arrival and departure of a specific jet at airport a.

$$\text{RELOCATE\_TIME}(a,b): \begin{cases} 0 & \text{if } a = b, \\ \text{FLIGHT\_TIME}(a,b) & \\ \quad + \text{TURN\_AROUND}(b) & \\ & \text{otherwise.} \end{cases}$$

We assume that time is measured in minutes; i.e., a time instant is an integer in $[0, 1{,}440]$. We often use a specific discretization of the planning horizon. The set of all time instants in the discretization will be denoted by $\mathcal{T}$. The set $\mathcal{T}_j(a) \subseteq \mathcal{T}$ represents the time instants in which jet $j \in \mathcal{J}$ will be allowed to take off from airport $a \in \mathcal{A}$. The discretization can (and often will) be different for each jet $j \in \mathcal{J}$. For $h = \text{HOME\_BASE}(j)$, we assume that JET_BEGIN(j) and JET_END(j) are in $\mathcal{T}_j(h)$. For each $t \in \mathcal{T}$, $j \in \mathcal{J}$, and $a \in \mathcal{A}$ define $\lceil t \rceil_{j,a} = \min\{t' \in \mathcal{T}_j(a)\colon t' \geq t\}$ and $\lfloor t \rfloor_{j,a} = \max\{t' \in \mathcal{T}_j(a)\colon t' \leq t\}$; whenever the jet and the airport are clear from the context, we may simply write $\lceil t \rceil$ and $\lfloor t \rfloor$.

The dial-a-flight problem is an example of a pickup and delivery problem. Pickup and delivery problems have received a fair amount of attention in the vehicle routing literature. Savelsbergh and Sol (1995) and Desaulniers et al. (2002) provide overviews of pickup and delivery problems. The class of vehicle-routing problems with pickups and deliveries dealing specifically with passenger transportation is known under the name of dial-a-ride problems. When dealing with passenger transportation, service-related constraints and objectives take on a more prominent role. These

are typically aimed at controlling "user inconvenience" in terms of ride time (the time between pickup and delivery), waiting time (time spent in the vehicle when it is not in motion), and deviations from desired pickup and delivery times. A discussion of modeling issues in dial-a-ride problems and an overview of proposed algorithms can be found in Cordeau and Laporte (2003). Even though dial-a-flight and dial-a-ride have many common characteristics, there are also some notable differences. The dial-a-ride problem often arises in social services contexts, e.g., transportation of the elderly, whereas the dial-a-flight problem is encountered exclusively in business settings. As a result, there tends to be less flexibility in the specification of requests, especially in terms of the desired service level, in dial-a-ride environments. Furthermore, in dial-a-ride environments, requests often have a common destination or a common origin (e.g., elderly citizens needing to visit a hospital, or wanting to go to the mall), whereas in dial-a-flight environments this rarely happens. Other relevant dial-a-ride papers include Dumas, Desrosiers, and Soumis (1991); Savelsbergh and Sol (1998); Xu et al. (2001); Ropke and Pisinger (2006); Cordeau (2006); Borndorfer et al. (1997); and Bent and Van Hentenryck (2005).

Fractional ownership is another new development in the airline industry (Keskinocak and Tayur 1998; Martin, Jones, and Keskinocak 2003; Hicks et al. 2005; Yao et al. 2005). Fractional ownership programs provide share sizes from one-sixteenth with 50 flying hours per year to one-half with 400 flying hours per year. Usually, a partial owner requests a flight by specifying a departure station, a departure time, an arrival station, and an arrival time, only days or hours ahead of time. The management company must assign a crew and an available aircraft to serve this flight. While scheduling all the requested flights, the management company tries to minimize total operational costs. Fractional ownership leads to per-plane, on-demand air transportation, as opposed to per-seat, on-demand air transportation considered in this paper. The business model and resulting scheduling problems are quite different, because the success of per-seat, on-demand air transportation depends on the ability to effectively aggregate requests, i.e., use the same flight leg to (partially) satisfy multiple transportation requests. Moreover, in fractional ownership planes are typically requested for at least several hours at a time so the number of requests that need to be dealt with are at least an order of magnitude less than in the per-seat, on-demand scenario.

3. Motivation and Contribution

Although the dial-a-flight problem can be viewed as a dial-a-ride problem in which the vehicles are jets, the current methodology that is available for these problems is not adequate for the dial-a-flight problem. This is due to two reasons. First, in the dial-a-flight problem "user inconvenience" is not only controlled by imposing a maximum transit time, as is typically done in dial-a-ride problems, but also by imposing that passengers make at most one intermediate stop between their origin and destination. The second reason is the sheer size of the problems that must be solved. DayJet is aiming to have over 300 jets operating daily within two years. Thus, on a typical day, about 3,000 accepted reservations must be scheduled. State-of-the-art dial-a-ride and vehicle-routing algorithms are well short of tackling such large problems in a short period of time.

There are two natural ways of modeling the offline schedule optimization problem. First, it can be viewed as an integer multicommodity flow problem on a time-space network in which all jets form a commodity. This involves a large number of commodities and a huge number of nodes because the desired level of time granularity is minutes (see Cordeau et al. 2007 for a description of this model). More importantly, a prohibitively large number of forcing constraints is required to model that requests can only be served on an arc when there is a jet assigned to that arc as well. The alternative is a column generation/branch-and-price approach in which a column corresponds to a feasible itinerary for one jet. This latter approach can produce a tighter linear programming bound at the expense of having an exponential number of variables. In our preliminary studies we considered both models. Neither produced the results for which we had hoped. The multicommodity flow model could solve small instances with fewer than four planes, but solution times grew exponentially with the number of planes so that even an instance with eight planes could not be solved in the desired time because of the exponential growth in the size of the network and the associated model. On the other hand, the column generation model could not even solve the linear programming relaxation for instances with 20 planes in a reasonable amount of time. The reason for this is that the subproblem for generating columns is a very difficult constrained shortest path.

This led us to develop a new multicommodity network flow formulation. Instead of the natural integer multicommodity flow on a time-space network, we opted for an integer multicommodity flow model on a time-activity network in which it is easier to handle the constraint that limits the number of intermediate stops to at most one. This two-part paper discusses this integer multicommodity flow model on a time-activity network and how it is used to provide high-quality solutions to real-life, large-scale instances of the dial-a-flight problem.

In Part I, we describe and analyze the integer multicommodity flow model. To achieve acceptable computational performance, three specialized techniques had to be developed. The first two reduce network size significantly and the third technique both reduces network size and tightens the linear programming relaxation. They are: (1) a customized network construction algorithm exploiting objective function characteristics; (2) a nonregular time-discretization increasing granularity around important time instants; and (3) a network aggregation algorithm. The network aggregation algorithm contracts nodes, thus incorporating information arising from side constraints into the network model. The resulting network, in which arcs correspond to partial routes that can be flown by a plane, can be considered as achieving some of the advantages of a column generation approach in which the variables correspond to a full route for a plane. Therefore, the aggregation gives us some of the benefits of a column generation formulation without having to design and implement a full-scale branch-and-price algorithm. The efficacy of the resulting algorithm, in terms of solution time and number of requests satisfied, is comparable to state-of-the-art algorithms used in other similar applications. Instances with fewer than 50 requests (four jets) are easy to solve, and instances with 80 or more requests (eight jets) are hard to solve. This is in line with recent computational results for the dial-a-ride problem (Cordeau 2006) and the capacitated vehicle-routing problem (Fukasawa et al. 2006; Lysgaard, Letchford, and Eglese 2004).

In Part II, we discuss the use of the integer multicommodity flow model for solving real-life, large-scale instances with several thousands of requests and several hundred jets. We present a parallel local search algorithm that optimizes subsets of planes. To achieve the desired computational performance, we give novel approaches to focus the search on neighborhoods that provide good opportunities for improvement, as well as using an asynchronous parallel implementation to explore a large number of neighborhoods. The neighborhood designs are based on parameters such as number of jets, time discretization level, and time of day. We develop customized metrics to select subsets of jets with a high probability of leading to an improvement. Our adaptive neighborhood selection scheme modifies the neighborhoods as the search progresses. Instances with about 3,000 requests and over 300 jets are handled routinely and effectively. While our results are not directly comparable to recent computational results for the pickup and delivery problem with time windows (Bent and van Hentenryck 2003; Ropke and Pisinger 2006), they show that our approach is capable of dealing with problems of at least the same size in comparable time.

4. A Multicommodity Network Flow Model

We begin by describing a discretized time-space network model representing feasible itineraries for a single jet airplane. The key events from the perspective of the jet are modeled in this network, e.g., which flights to make, when to make them, and which passengers to carry in each flight. Linking these networks together via constraints that impose that all requests must be satisfied yields a multicommodity network flow model with side constraints.

4.1. The Individual Jet Network

For each jet $j \in \mathcal{J}$, we define a *feasible itinerary* as a sequence of flights satisfying the following conditions:

(R1) The jet begins and ends the itinerary at $\textsc{home\_base}(j)$,

(R2) The jet begins the itinerary no earlier than $\textsc{jet\_begin}(j)$ and ends the itinerary no later than $\textsc{jet\_end}(j)$,

(R3) The jet never carries more than $\textsc{capacity}(j)$ passengers on a flight,

(R4) The jet never carries more than $\textsc{max\_weight}(j)$ weight on a flight,

(R5) The jet does not fly more than $\textsc{max\_fly\_time}(j)$ minutes during a day, and

(R6) The service requirements of passengers transported by the jet are met; that is, passengers are picked up at their origin and dropped off at their destination within their specified time window with at most one intermediate stop, and without changing airplanes during their trip.

The Individual Jet Network allows us to model any feasible itinerary for a given jet $j \in \mathcal{J}$ in terms of a flow between two nodes in the network. Nodes in this network represent key decisions taken from the perspective of the jet as it travels during the day between different airports. These decisions are of three types: standby, departure, and loading decisions. Deciding to remain in "standby" allows a jet to wait, idle, at an airport. A "departure" decision involves deciding where to fly next, and "loading" decisions involve choosing which requests to satisfy. In Figure 1 we illustrate possible sequences of such events as represented in an individual jet network.

We now give a formal description of the network model in terms of nodes and arcs.

4.1.1. Nodes. Let $\mathcal{R}[a, t, b]$ be the set of all requests $r \in \mathcal{R}$ with $\textsc{origin}(r) = a$ and $\textsc{destination}(r) = b$ that can be serviced by a direct flight departing from a at time t. That is, the set of all requests $r \in \mathcal{R}$ such that $t \geq \textsc{earliest}(r)$ and $t + \textsc{flight\_time}(a, b) \leq \textsc{latest}(r)$. Let $\mathcal{R}[a, t_1, c, t_2, b]$ be the set of all requests $r \in \mathcal{R}$ with $\textsc{origin}(r) = a$ and $\textsc{destination}(r) = b$ that

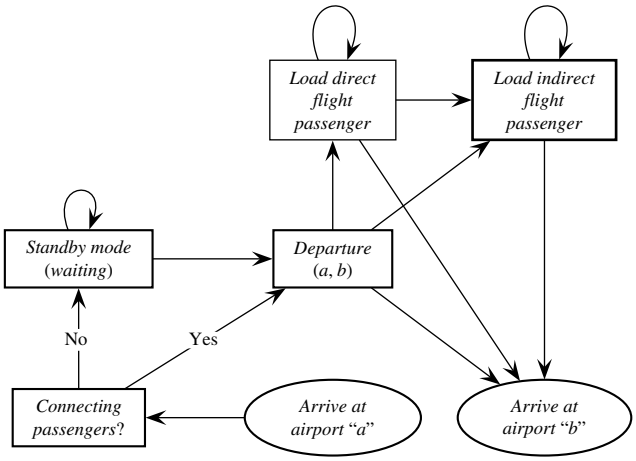

Figure 1 Key Events from the Perspective of an Airplane

can be serviced by an indirect flight through airport c departing from a at time t_1 and departing from c at time t_2, i.e., such that $t_1 \geq \text{EARLIEST}(r)$, $t_1 + \text{RELOCATE\_TIME}(a, c) \leq t_2$ and $t_2 + \text{FLIGHT\_TIME}(c, b) \leq \text{LATEST}(r)$. Furthermore, let $\mathcal{R}[a, t]$ be the set of all requests $r \in \mathcal{R}$ with $\text{ORIGIN}(r) = a$ that can be serviced by a flight starting at time t; i.e., $\mathcal{R}[a, t] = \bigcup_{b \in \mathcal{A}} \mathcal{R}[a, t, b] \cup (\bigcup_{c \in \mathcal{A}: c \neq a \wedge c \neq b,\, t' \geq t} \mathcal{R}[a, t, c, t', b])$.

Standby Nodes. For every airport $a \in \mathcal{A}$ and every instant $t \in \mathcal{T}_j(a)$ define a node $S_j(a, t)$. This node models that jet j is idle, or in standby mode, that is, without any passengers onboard, and ready to take off at airport a at time t.

Gate Nodes. For every pair of airports $a, b \in \mathcal{A}$ and every instant $t \in \mathcal{T}_j(a)$ define a node $G_j(a, t, b)$. This node models that jet j will depart from airport a to airport b at time t.

Direct Loading Nodes. For each $t \in \mathcal{T}_j(a)$, and each $r \in \mathcal{R}[a, t, b]$ node $DL_j(a, t, b, r)$ models that request r will be satisfied with a direct flight on jet j departing at time t.

Indirect Loading Nodes. For each $t_1 \in \mathcal{T}_j(a)$, $t_2 \in \mathcal{T}_j(c)$, and each $r \in \mathcal{R}[a, t_1, c, t_2, b]$ node $IL_j(a, t_1, c, t_2, b, r)$ models that request r will be satisfied indirectly by jet j flying from airport a to airport c at time t_1, and then flying from airport c to airport b at time t_2.

Terminal Nodes. Let

$$h = \text{HOME\_BASE}(j), \quad t_1 = \text{JET\_BEGIN}(j), \quad \text{and}$$
$$t_2 = \text{JET\_END}(j).$$

Let $T_j^b \equiv S_j(h, t_1)$ and $T_j^e \equiv S_j(h, t_2)$. Nodes T_j^b and T_j^e are called the *terminal nodes* for jet j.

4.1.2. Arcs.

Idle Arcs. For every airport $a \in \mathcal{A}$ and every pair of time instants $t_1, t_2 \in \mathcal{T}_j(a)$ such that $t_2 > t_1$ put an arc from $S_j(a, t_1)$ to $S_j(a, t_2)$.

Departure Arcs. For every pair of airports $a, b \in \mathcal{A}$ and every time instant $t \in \mathcal{T}_j(a)$ put an arc from $S_j(a, t)$ to $G_j(a, t, b)$.

Loading Arcs. For every pair of airports $a, b \in \mathcal{A}$, every time instant $t \in \mathcal{T}_j(a)$, and every service request $r \in \mathcal{R}[a, t, b]$ introduce an arc from $G_j(a, t, b)$ to $DL_j(a, t, b, r)$. Likewise, for every $a, b, c \in \mathcal{A}$, every $t_1 \in \mathcal{T}_j(a)$, $t_2 \in \mathcal{T}(j, c)$, and every $r \in \mathcal{R}[a, t_1, c, t_2, b]$ put an arc $G_j(a, t_1, b)$ to $IL_j(a, t_1, c, t_2, b, r)$.

Aggregation Arcs. For each pair of requests $r_1, r_2 \in \mathcal{R}[a, t, b]$ such that $r_1 < r_2$ introduce an arc from $DL_j(a, t, b, r_1)$ to $DL_j(a, t, b, r_2)$. For each request $r_1 \in \mathcal{R}[a, t_1, b]$ and request $r_2 \in \mathcal{R}[a, t_1, b, t_2, c]$ introduce an arc from $DL_j(a, t_1, b, r_1)$ to $IL_j(a, t_1, b, t_2, c, r_2)$. For each pair of requests $r_1, r_2 \in \mathcal{R}[a, t_1, b, t_2, c]$ such that $r_1 < r_2$ put an arc from $IL_j(a, t_1, b, t_2, c, r_1)$ to $IL_j(a, t_1, b, t_2, c, r_2)$. (Note that, by design, aggregation of direct flights happens before aggregation of indirect flights.)

Relocation Arcs. For each pair of airports $a, b \in \mathcal{A}$ and every pair of time instants $t_1 \in \mathcal{T}_j(a)$, $t_2 \in \mathcal{T}(j, b)$ such that $t_2 = \lceil t_1 + \text{RELOCATE\_TIME}(a, b) \rceil$, put an arc from $G_j(a, t_1, b)$ to $S_j(b, t_2)$.

Direct Flight Arcs. For each pair of airports $a, b \in \mathcal{A}$, each time instant $t_1 \in \mathcal{T}_j(a)$, and each request $r \in \mathcal{R}[a, t_1, b]$ put an arc from $DL_j(a, t_1, b, r)$ to $S_j(b, t_2)$, where $t_2 = \lceil t_1 + \text{RELOCATE\_TIME}(a, b) \rceil$.

Indirect Flight Arcs. For each triple of airports $a, b, c \in \mathcal{A}$, time instant $t_1 \in \mathcal{T}_j(a)$, time instant $t_2 \in \mathcal{T}_j(b)$, and each request $r \in \mathcal{R}[a, t_1, b, t_2, c]$ put an arc from $IL_j(a, t_1, b, t_2, c, r)$ to $G_j(b, t_2, c)$.

4.1.3. A Small Example.

Consider a single jet problem instance defined by airports $\mathcal{A} = \{a, b, c\}$, and assume that $\text{FLIGHT\_TIME}(a, b) = 3$, $\text{FLIGHT\_TIME}(b, c) = 3$, and $\text{FLIGHT\_TIME}(a, c) = 5$. Suppose that $\text{TURN\_AROUND}(\cdot) = 1$ for all airports and that we need to satisfy the requests given in Table 1. An example of the corresponding individual jet network is given in Figure 2. Figure 3 depicts two possible itineraries for the jet. In the first itinerary, the jet satisfies request r_1 by means of a direct flight from a to b and then deadheads from b to airport c. In the second itinerary, the jet satisfies requests r_1, r_2, and r_3. To do this, the jet picks up r_1 and r_2 at airport a, drops off r_1 and picks up r_3 at airport b, and then drops off r_2 and r_3 at airport c.

4.1.4. Observations.

Before describing the multicommodity network flow formulation, we highlight

Table 1 Flight Requests

r	ORIGIN(r)	EARLIEST(r)	DESTINATION(r)	LATEST(r)
r_1	a	1	b	5
r_2	a	1	c	8
r_3	b	4	c	9

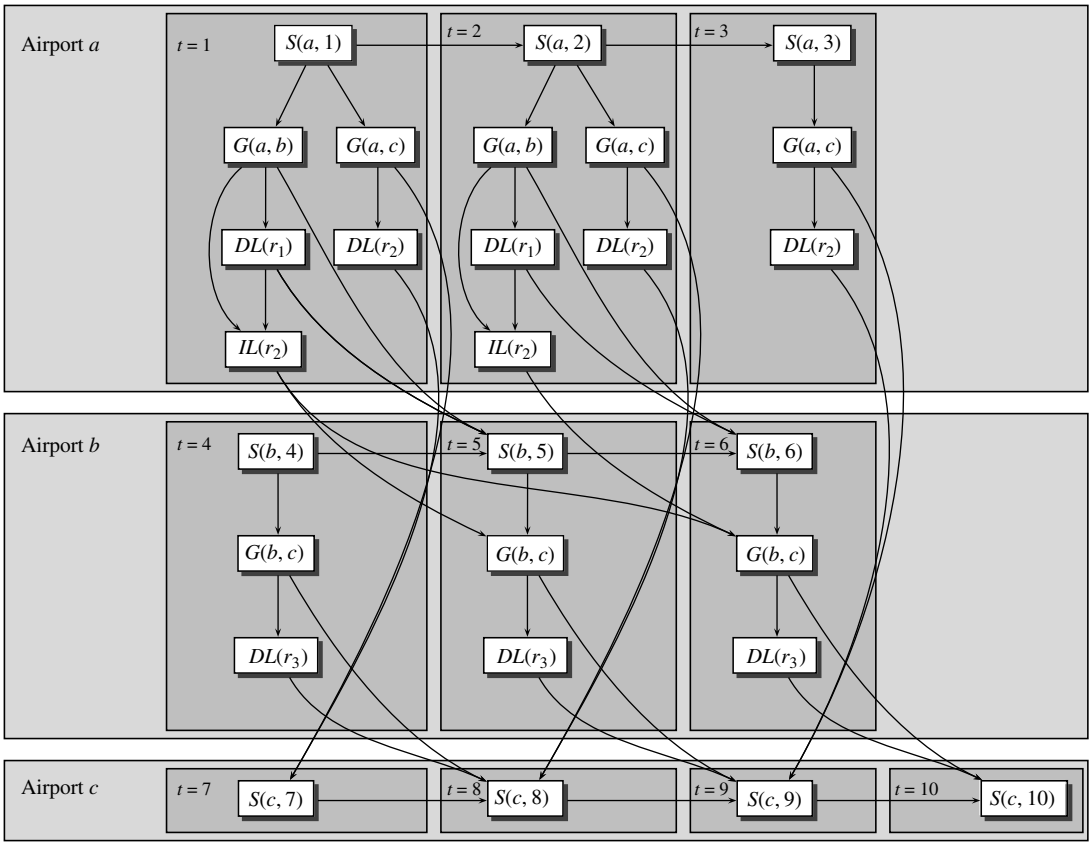

Figure 2 Individual Jet Network

some important characteristics of the individual jet networks. First, and most importantly, the individual jet networks are acyclic. Furthermore, every feasible itinerary for jet j corresponds to a path from node T_j^b to node T_j^e in the individual jet network, but the converse is not always true. That is, not every path going from T_j^b to T_j^e corresponds to a feasible itinerary. A path from T_j^b to T_j^e will always satisfy conditions (R1), (R2), and (R6) because these are explicitly modeled in the network. However, conditions (R3), (R4), and (R5) may not be satisfied by every path, and a path from T_j^b to T_j^e may satisfy a request more than once.

4.2. A Multicommodity Network Flow Formulation

A solution to the dial-a-flight problem consists of a set of feasible itineraries, one for each jet $j \in \mathcal{J}$, satisfying all of the service requests $r \in \mathcal{R}$ at minimum cost. We formulate this problem as a network flow-based integer program. For each jet $j \in \mathcal{J}$, we formulate a min-cost flow problem in which one unit of flow is sent from node T_j^b to T_j^e in the corresponding individual network, adding side constraints to ensure that each individual jet itinerary satisfies constraints (R3)–(R5). Then, a constraint is generated that links all of these network flow problems together by imposing that all requests must be satisfied exactly once.

Let V_j and E_j denote the set of all nodes and arcs in the individual jet network corresponding to each $j \in \mathcal{J}$. For each arc $e \in E_j$ define a binary variable,

$$x_e = \begin{cases} 1 & \text{if jet } j \text{ uses arc } e, \\ 0 & \text{otherwise.} \end{cases}$$

For each individual jet network (V_j, E_j) with $j \in \mathcal{J}$, we require that a single unit of flow must go from node T_j^b to node T_j^e by imposing the network flow conservation constraints (where tail(e) and head(e) denote the tail and the head of arc e)

$$\sum_{e \in E_j:\, \text{tail}(e) = T_j^b} x_e = 1 \quad \forall j \in \mathcal{J},$$

$$\sum_{e \in E_j:\, \text{head}(e) = T_j^e} x_e = 1 \quad \forall j \in \mathcal{J},$$

$$\sum_{e \in E_j:\, \text{head}(e) = v} x_e = \sum_{e \in E_j:\, \text{tail}(e) = v} x_e \quad \forall v \in V_j \backslash \{T_j^b, T_j^e\}\ \forall j \in \mathcal{J}.$$

Next, we consider the side constraints.

Capacity (R3). Consider airports $a, b \in \mathcal{A}$ and a time instant $t_1 \in \mathcal{T}$. To each arc $e \in E_j$ into a direct loading node $DL_j(a, t_1, b, r)$, into an indirect loading

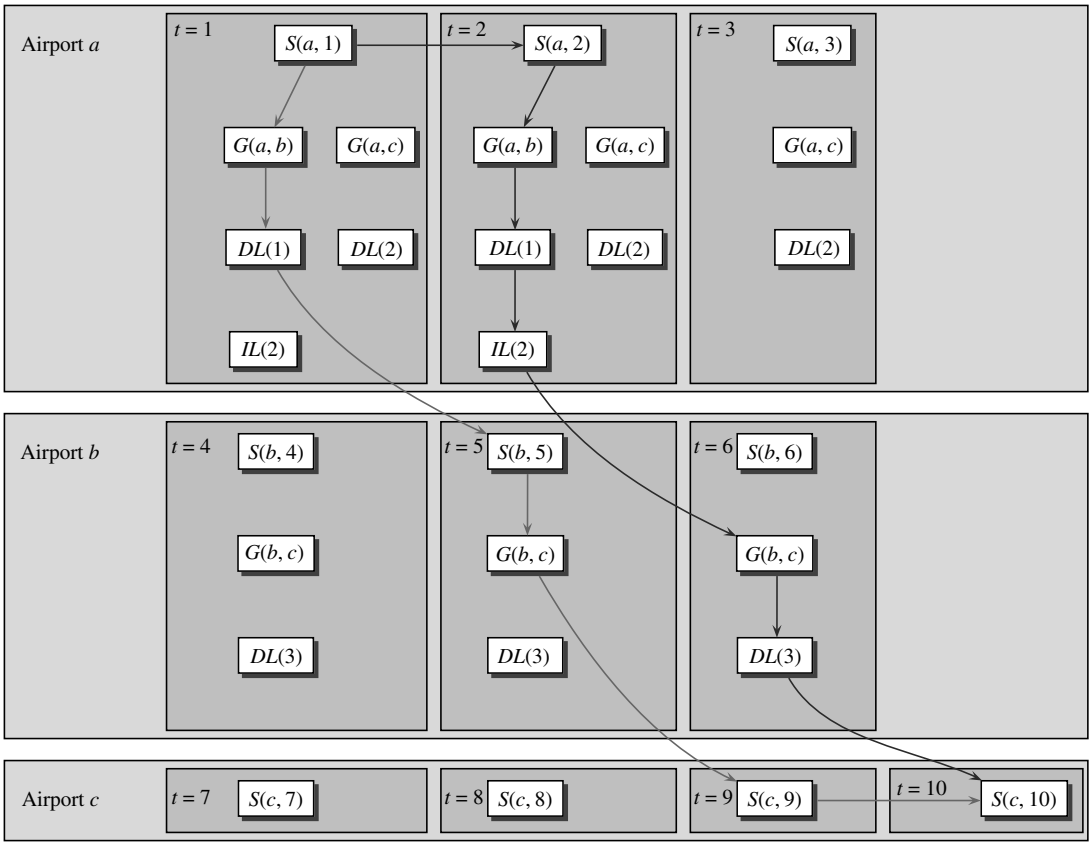

Figure 3 Two Possible Itineraries in the Jet Network

node $IL_j(a, t_1, b, t_2, c, r)$, and into an indirect loading node $IL_j(c, t_3, a, t_1, b, r)$ assign consumption $q_e^{a,b,t_1} = \text{PASSENGERS}(r)$, and to all other arcs $e \in E_j$ assign consumption $q_e^{a,b,t_1} = 0$. For each jet and each flight segment, we impose a jet capacity constraint

$$\sum_{e \in E_j} q_e^{a,b,t} x_e \le \text{CAPACITY}(j) \quad \forall a, b \in \mathcal{A},\ \forall t \in \mathcal{T},\ \forall j \in \mathcal{J}$$

limiting the number of passengers on the flight.

Weight (R4). Similarly, we assign consumption $w_e^{a,b,t_1} = \text{WEIGHT}(r)$ to each arc $e \in E_j$ into a direct loading node $DL_j(a, t_1, b, r)$, into an indirect loading node $IL_j(a, t_1, b, t_2, c, r)$, and into an indirect loading node $IL_j(c, t_3, a, t_1, b, r)$, and $q_e^{a,b,t_1} = 0$ to all other arcs $e \in E_j$. Then, for each jet and each flight segment, we impose a jet weight constraint

$$\sum_{e \in E_j} w_e^{a,b,t} x_e \le \text{MAX\_WEIGHT}(j)$$

$$\forall a, b \in \mathcal{A},\ \forall t \in \mathcal{T},\ \forall j \in \mathcal{J}$$

limiting the total weight on the flight.

Flying Time (R5). To each arc $e \in E_j$ into a gate node $G_j(a, t, b)$ assign flying time

$$f_e = \text{FLIGHT\_TIME}(a, b).$$

To all other arcs assign flying time $f_e = 0$. For each jet, we impose a flying time constraint

$$\sum_{e \in E_j} f_e x_e \le \text{MAX\_FLY\_TIME}(j) \quad \forall j \in \mathcal{J}$$

limiting the amount of time it is in the air.

We also need a constraint to link together all of the individual jet networks and to impose that all requests are satisfied.

Request Satisfaction. Consider a request $r \in \mathcal{R}$. To each loading arc $e \in E$ involving request r assign $s_e^r = 1$. To all other arcs assign $s_e^r = 0$. We impose the constraint

$$\sum_{j \in \mathcal{J}} \sum_{e \in E_j} s_e^r x_e = 1 \quad \forall r \in \mathcal{R},$$

which guarantees that each request must be satisfied exactly once.

Because each Individual Jet Network is acyclic, we need not be concerned about the existence of subtours (or closed directed cycles) in the solution. That is, every feasible solution to the constraints defined above must consist of $|\mathcal{J}|$ paths, one for each jet $j \in \mathcal{J}$ going from node T_j^b to T_j^e.

Finally, the objective function is defined as a linear function on the flights made by each individual jet.

Objective Function. To each arc $e \in E$ into a gate node $G_j(a, t, b)$ assign cost $c_e = \text{FLIGHT\_COST}(a, b)$. To all other arcs assign cost $c_e = 0$. These costs are used to specify the objective function of the problem, which is

$$\min \sum_{j \in \mathcal{J}} \sum_{e \in E_j} c_e x_e.$$

We have chosen to use the flying time f_e for arc e as a proxy for the cost c_e because operational costs primarily depend on fuel consumption. (Pilots are salaried employees.) It is a proxy because fuel consumption is not a linear function of the flying time. Planes use more fuel during take off and landing, and fuel use also depends on the weight of the plane (determined by the people and fuel aboard).

Aggregating Jets. The multicommodity network flow model presented above has a time-discretized network for *each* jet and models feasible jet itineraries as paths through these networks. Observe that for all jets $j \in \mathcal{J}$ with $\text{HOME\_BASE}(j) = a$, $\text{JET\_BEGIN}(j) = t_1$, and $\text{JET\_END}(j) = t_2$ the networks (V_j, E_j) are identical. If we define jet class $\mathcal{J}[a, t_1, t_2]$ to be the set of jets with $\text{HOME\_BASE}(j) = a$, $\text{JET\_BEGIN}(j) = t_1$, and $\text{JET\_END}(j) = t_2$, then rather than modeling the itineraries of jets in jet class $\mathcal{J}[a, t_1, t_2]$ as paths in separate networks, we can model them as a flow through a single network. This reduces the number of the networks as well as the symmetry in the formulation and may therefore make the problem easier to solve. The disadvantage of working with a flow formulation instead of a path formulation is that side constraints can only be imposed in aggregate form. For example, we can only impose that the combined flying time of the jets in a jet class does not exceed a certain limit. We can no longer limit the flying time of an individual jet. There are ways to deal with these issues, but all of them reduce the potential benefits of the flow formulation. Initial computational experiments with the flow formulation showed only minor improvements. Therefore, the flow formulation has not been pursued in any depth.

5. Constructing the Network

The network flow formulation described in the previous section gets very large very quickly. However, by being careful and intelligent it is possible to create a much smaller formulation that still contains an optimal solution. In addition to being smaller, the formulation may also result in a tighter linear programming relaxation. The creation of this smaller formulation proceeds in two phases. In the first phase, we carefully construct the individual jet networks, trying to prevent the inclusion of nodes and arcs that will not be part of an optimal solution. In the second phase, we judiciously analyze substructures in the network (sometimes in combination with side constraints) to see if we can reduce its size.

5.1. The Rolling Forward Algorithm

To ensure the creation of an individual network of acceptable size, we exploit the following three observations:

(P1) Feasibility Considerations May Lead to the Elimination of Arcs. Each jet is constrained by the earliest time at which it can depart from its home base and the latest time by which it must return. Also, each jet has constraints limiting its flying time and the number of passengers and weight aboard at any time. Therefore, for any given jet $j \in \mathcal{J}$, many of the arcs in the individual jet network as defined in the previous section cannot actually be used in any feasible path going from T_j^b to T_j^e. For example, consider an airport $a \in \mathcal{A}$ and time instants $t_1 < t_2 \in \mathcal{T}_j(a)$. For an idle arc connecting standby nodes $S_j(a, t_1)$ and $S(a, t_2)$ to be usable, the following sequence of events must be possible: The jet must be able to fly from its home base to airport a and complete the turnaround by time t_1, fly back to its home base from airport a taking off at time t_2, while satisfying the flying time limit and staying within its duty period. As another example, consider two airports $a, b \in \mathcal{A}$, a time instant $t \in \mathcal{T}_j(a)$, and two requests $r_1, r_2 \in \mathcal{R}[a, t, b]$. For an aggregation arc connecting direct loading nodes $DL_j(a, t, b, r_1)$ and $DL_j(a, t, b, r_2)$ to be usable, the following sequence of events must be possible: The jet must be able to fly from its home base to airport a and complete the turnaround by time t, fly to airport b, turn around, and have enough time to return to its home base from airport b within its duty period. In addition, the combined number of passengers and weight of requests r_1 and r_2 cannot be more than the passenger and weight limit, respectively, and the maximum flying time cannot be exceeded.

(P2) Optimality Considerations May Lead to the Elimination of Nodes and Arcs. Given that costs are a function of the flights in an itinerary, flights have to occur for one of the following reasons: (a) to pick up a passenger, (b) to drop off a passenger, or (c) to return to the home base. Because we are assuming that a jet can fly between any pair of airports, there is no need to make two consecutive flights without any passengers aboard.

(P3) Objective Function Characteristics May Lead to the Elimination of Nodes and Arcs. If the cost of a flight does not depend on its departure time, then the costs of itineraries that are identical except for flight departure times are the same. In such situations, it suffices to construct individual jet networks in which at least one of these itineraries can be found. We exploit this observation by only including flight arcs (direct and indirect) in the individual jet networks corresponding to earliest possible departures.

Next, we describe how these observations can be used to construct a network for jet $j \in J$ consisting of

nodes $\hat{V}_j$ and arcs $\hat{E}_j$ contained in V_j and E_j, respectively, that still contains at least one optimal solution. Because this algorithm constructs the resulting graph in chronological order with respect to the time at which events take place, we call it the rolling forward algorithm.

The algorithm is initialized by setting $\hat{V} := \{T_j^b\}$, $\hat{E} := \emptyset$, and $t = 0$. For any given $t \in \mathcal{T}$, let $\hat{V}[t]$ consist of the nodes in $\hat{V}$ corresponding to events taking place at time t. Whenever a node is added to the set $\hat{V}$, it is labeled *unprocessed*. The algorithm iterates forward through time by selecting unprocessed nodes in $\hat{V}[t]$ and processing them. When $\hat{V}[t]$ contains several nodes, they are processed in the following order: standby nodes, gate nodes, direct loading nodes, and finally, indirect loading nodes. Processing a node v involves identifying all possible events that can directly follow the event represented by node v, creating nodes representing these subsequent events (unless the network already contains nodes representing these events), and adding arcs connecting v to these (new) nodes. Once all nodes in $\hat{V}[t]$ are processed, t is set to the next $t' > t$ such that $\hat{V}[t']$ is nonempty. The algorithm proceeds in this way until all nodes in $\hat{V}$ are processed. Because the analysis of possible subsequent events is done using observations (P1), (P2), and (P3), every node v in the constructed network will be such that it is in some feasible path from T_j^b to T_j^e. Note that when processing nodes in $\hat{V}[t]$, only nodes in $\hat{V}[t']$ with $t' \geq t$ are added.

To each airport, time instant pair (a, t), with $a \in A$ and $t \in \mathcal{T}$, we may possibly add labels indicating that there are passengers arriving at their destination, or that there are passengers arriving and connecting to their final destination(s). When the algorithm begins, the pair (HOME_BASE(j), JET_BEGIN(j)) is labeled as *final-stop*. All other pairs (a, t) are unlabeled, but this can change as nodes are added to $\hat{V}_j$. The possible labels are *final stop*, and *middle stop* (b), for all $b \in \mathcal{A}$. Note that a pair (a, t) may have several labels.

The processing of a node depends on the airport a, the time t, the labels defined at (a, t), the time discretization, and the type of node. The details of the processing of nodes, even though important for the overall success of the solution approach, are tedious and repetitive. Therefore, they are presented in the appendix.

5.2. Time Discretizations

Executing the optimization algorithm with $\mathcal{T}_j(a)$ equal to all integer points in [0, 1,440], corresponding to the minutes in a 24-hour day, typically results in the generation of an excessively large network, even if the network is constructed carefully using the rolling forward algorithm. In this section, we discuss methods for generating customized discretizations strictly contained in [0, 1,440]. This means that for each jet $j \in \mathcal{J}$ and for each airport $a \in \mathcal{A}$ we need to define the set $\mathcal{T}_j(a)$ to be suitably small without sacrificing our goal of finding high-quality solutions. We will describe some heuristics that gave the best empirical results. The starting point for all of our discretization schemes will be to choose an integer parameter $\Delta \in \{1, \ldots, 1{,}440\}$, and, using this value, define a regular-interval time discretization $(0, \Delta, 2\Delta, \ldots, k\Delta)$, where $k\Delta \leq 1{,}440$ and $(k+1)\Delta > 1{,}440$. To these time instants we add three other types of time instants that capture key moments during the day:

1. For each request $r \in \mathcal{R}$ and for $j \in \mathcal{J}$ that can feasibly satisfy request r, add the earliest time t_r^j at which jet j can pick up the passengers in request r to $\mathcal{T}_j(a)$.

2. For each jet $j \in \mathcal{J}$ and each take off time t at airport a in the best-known feasible schedule (usually generated by a heuristic before starting the optimization process), add time instant t to $\mathcal{T}_j(a)$. This ensures that the best-known feasible schedule can be represented in the resulting network.

3. For each airport, add additional time instants (typically integer multiples of $\lceil \Delta/2 \rceil$) during congested periods of the day, i.e., periods of the day with a high number of anticipated takeoffs.

5.3. Aggregation

Even though the network formulation is judiciously constructed, it may still be large, may have many side constraints, and may have a weak linear programming relaxation. To reduce the size of the formulation and to increase the strength of the linear programming relaxation, we perform various aggregations.

The idea of aggregation is very natural and can be used in any network. It is especially useful when there are tight side constraints. We will introduce our aggregation procedures for general networks and then discuss how these procedures apply to the dial-a-flight network.

Consider a network $\mathcal{N}$ with nodes V and arcs A. Assume that there are k resources (which are consumed additively), and let vector $r_a \in \mathbb{R}_+^k$ represent the amount of resources consumed when traversing arc $a \in A$. Let $R \in \mathbb{R}_+^k$ represent the total resources available, and for each $a \in A$ let x_a be a binary variable indicating whether or not we use arc a. Assume that we wish to find a min-cost flow of one integer unit from a node $s \in V$ to a node $t \in V$, such that the resource usage constraint

$$\sum_{a \in A} r_a x_a \leq R$$

is satisfied.

For each node $v \in V$, let $\delta(v)$ represent the set of all arcs incident to v. Define $\delta^-(v)$ ($\delta^+(v)$) as the set

of all arcs in $\delta(v)$ whose head (tail) is v and let $r^-(v)$ ($r^+(v)$) be such that for each $i \in 1, \ldots, k$, $r_i^-(v)$ ($r^+(v)$) represents the minimum consumption of resource i required to get from node s to node v (node v to node t).

An iteration of the aggregation algorithm works as follows:

1. Choose a node $v \in V$.
2. Delete node v and all arcs in $\delta(v)$ from $\mathcal{N}$.
3. For each arc $e \in \delta^-(v)$ and each arc $f \in \delta^+(v)$ define an arc ef such that its tail coincides with the tail of e and its head coincides with the head of f. Define $r_{ef} = r_e + r_f$. If $r^-(\text{tail}(e)) + r_{ef} + r^+(\text{head}(f)) \le R$, then add edge ef to $\mathcal{N}$.

Consider a network $\mathcal{N}$ with nodes $\{A, B, C, D, E, M\}$ connected to each other as depicted in Figure 4(a). Assume that aggregation algorithm chooses node M. In Figure 4(b) we see what the resulting network looks like if the resource constraints do not allow us to eliminate any arcs in Step 3 of the aggregation algorithm. In Figure 4(c) we see what the resulting network may look like if the resource constraints do allow us to eliminate arcs in Step 3 of the aggregation algorithm (in this case, the elimination of arcs (A, C) and (B, D)).

If the resource constraints are loose and few arcs are eliminated in Step 3 of the aggregation algorithm, the size of the network may greatly increase. In fact, at each iteration of the aggregation algorithm, one could potentially be adding a quadratic number of arcs relative to the number of arcs removed. As a precaution, we choose a parameter $\kappa \in \mathbb{N}_+$, and the aggregation algorithm only deletes nodes $v \in V$ such that $|\delta^-(v)||\delta^+(v)| \le |\delta^-(v)| + |\delta^+(v)| + \kappa$. Even for $\kappa = 0$, the aggregation algorithm can greatly reduce the number of nodes and arcs by eliminating path-like and forklike structures, which seem to be very common in sparse networks such as the individual jet networks. For $\kappa > 0$, significant network-size reductions can be obtained when resource limits are tight, causing many arcs to be eliminated in Step 3 of the aggregation algorithm. We have observed that choosing direct and indirect loading nodes in Step 1 of the aggregation algorithm can be very effective due to the restrictive seat and weight capacity of the jets.

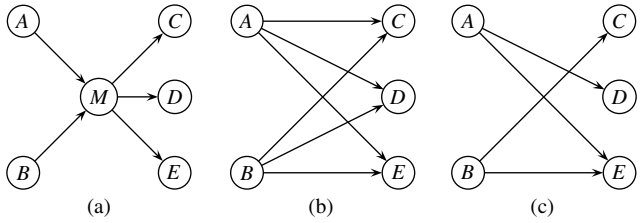

Figure 4 The Aggregation Algorithm

We define two arcs $e_1, e_2 \in A$ to be *parallel* if there is no directed path in $\mathcal{N}$ containing both e_1 and e_2. A set of arcs $F \subseteq A$ is said to be parallel if every pair of arcs in F is parallel. Furthermore, we define the *support* of constraint (R_i) $\sum_{e \in A}(r_e)_i x_e \le R_i$ to be $A_i = \{e \in A\colon (r_e)_i \ne 0\}$. Without loss of generality, we may assume that every arc $e \in A_i$ is such that $(r_e)_i \le R_i$, else the arc can be removed from the network.

Now observe that if the supporting arcs A_i of a constraint (R_i) are parallel, then constraint (R_i) can be dropped. This is due to the fact that when the set of arcs A_i is parallel, any path from T_j^b to T_j^e includes at most one arc e from A_i, and that arc will satisfy $(r_e)_i \le R_i$.

This observation can be exploited to eliminate side constraints. Consider a set of nodes $U \subseteq V$ with the following properties: (1) every arc $e \in A_i$ for some (R_i) has head $(e) \in U$, and (2) there does not exist a directed path starting and ending in U. Then, if one aggregates every node in U, constraint (R_i) can be eliminated. This is true because when we aggregate every node in U, every pair of arcs generated by the aggregation will be parallel, and these arcs define the support of constraint (R_i) in the aggregated network.

In the individual jet networks, capacity and weight side constraints can be eliminated with relative ease by successive aggregation iterations. Recall from §4.2 that there is one capacity constraint for each set of indices $a, b \in \mathcal{A}$, $t \in \mathcal{T}$, and $j \in \mathcal{J}$, and for each of these index sets, the edges with positive coefficients in the corresponding constraints are such that they are incident to a few specific nodes in the network.

5.4. Multiple Shifts

In the application developed for DayJet, there are two pilot shifts per day, one morning shift and one afternoon shift (shift changes happen only at the home base). More specifically, two time intervals are specified. Together, the two time intervals cover the operating hours in a day. Each time interval has a beginning time and an ending time. The first time interval ends some time after the second time interval begins; i.e., the time intervals overlap. The first time interval has to contain the morning shift and the second time interval has to contain the afternoon shift. The change in the problem is that now a jet must return to its home base sometime between the beginning of the second time interval and the ending of the first time interval so that there may be a pilot change. Pilot changes are assumed to be quick; i.e., they do not take any longer than the turnaround time at the home base and are allowed to occur while there are connecting passengers on board. After pilots are swapped, it is required that the jet returns to the home base before the second shift ends. Also, the maximum flying time constraint is imposed for each individual

shift because the FAA regulation is intended to limit the total daily flying hours of individual pilots.

To deal with the shift change, it suffices to use two copies of the individual jet network for each jet (one for each shift) and link each of these by arcs going through the home base in the time window during which pilot changes must take place. The same basic ideas used to build the rolling forward algorithm can be modified to take into account the pilot changes. One must simply be careful to enumerate all possible cases by which the shifts can change at the home base. Given that the methodology is exactly the same, and that the notation and case analysis required for the two-shift case is much more extensive, we will not go into details. However, given the relevance of the two-shift model to the actual application, we will report computational results for two shifts rather than just one.

6. Computational Results

To assess the value of the proposed multicommodity network flow formulation and the effectiveness of the rolling forward algorithm, the flexible discretization approach, and the aggregator, we performed a series of computational experiments on instances provided by DayJet.

DayJet provided three sets of 23 instances. Each instance consists of a list of service requests covering 17 airports in the southeastern United States. In addition, DayJet provided, for each instance, the best feasible schedule generated by their internally developed and proprietary heuristic, which they developed to determine whether requests could be accepted. In the first set of instances, there is a fleet of eight jets and an average of 81.36 requests, with an average of 101.10 passengers. In the second set of instances, there is a fleet of six jets and an average of 65.73 requests, with an average of 80.95 passengers. Finally, in the third set of instances, there is a fleet of four jets and an average of 43.40 requests, with an average of 53.56 passengers.

All the computational experiments were conducted on a cluster of 2.4 GHz Dual AMD 250 computers with 4 GB of RAM each. The source code was written in the C programming language, compiled with gcc version 3.4.3, and executed in Red Hat Enterprise Linux ES, release 4. All linear and integer programs were solved using CPLEX version 9.0.

Given the fact that a feasible schedule is available for each instance, our primary interest is evaluating whether our technology is able to obtain improved solutions in a reasonable amount of time. Even though our technology is capable of finding an optimal solution and proving its optimality, it is unlikely that this can be accomplished in an acceptable amount of time for larger instances; therefore, our focus is on improving given feasible schedules. This focus is reflected in the presentation of computational results, where we show improvements over the best-known feasible schedule. Starting with a feasible solution definitely helps the integer program because the feasibility problem is itself difficult, and without the upper bound provided by a feasible solution, the search tree might be larger.

The goal of our computational experiments is to gain an understanding of (1) the quality of the formulation obtained by the rolling forward algorithm, (2) the impact of the discretization scheme that is used, (3) the impact of the network aggregation techniques, and, foremost, and (4) whether the technology can significantly improve the solutions obtained by the DayJet heuristic in a reasonable amount of time.

The results of the computational experiments are presented in Tables 2–4. A time limit of four hours was imposed on the solution time of each integer program with optimality tolerance set to 0.01%. Linear programs at root nodes were solved using the barrier algorithm. Pseudo-reduced cost branching was used, and the up branch was always explored first. CPLEX heuristics were turned off. Finally, the value of the schedule produced by the DayJet Corporation heuristic was provided as an initial upper bound.

For the base experiment, we constructed the jet networks using the rolling forward algorithm with the most basic time discretization (i.e., $\Delta = 1{,}440$, which means only the "special" time instants) and without using the aggregation algorithm. The results are shown in Table 2, which gives the problem set used (PSET), the fleet size (JETS), the average percentage improvement over the initial solution after four hours of computation (IMPROVE), the average percentage gap between the best upper and lower bounds after four hours of computation (LP-GAP), the number of instances for which we were able to find an improving solution (FEAS), and the number of instances for which we were able to complete the search (OPT).

We observe that, even with this basic formulation, we are able to obtain improved schedules for many instances and that the average reduction in flying time is substantial, between 6% and 8%. On the other hand, the remaining integrality gaps after four hours of computing are still quite large, especially for the larger instances.

Next, we evaluate the impact of the aggregation algorithm and analyze how this impact varies based

Table 2 Results for the Base Experiment

PSET	JETS	IMPROV	LP-GAP	FEAS	OPT
1	8	6.16	26.20	11	0
2	6	7.88	12.81	20	3
3	4	6.02	4.61	17	15

Table 3 Aggregation

PSET	AGG	IMPROV	LP-GAP	ROWS	COLS	NZS	FEAS	OPT
1	0	6.16	26.20	100.00	100.00	100.00	11	0
	2	8.87	18.86	41.77	72.10	91.30	15	0
	4	11.52	13.79	36.13	73.21	99.48	20	1
	6	11.84	12.85	34.15	75.42	106.71	21	1
	8	12.90	10.95	32.75	78.80	116.26	22	2
	10	13.68	9.59	32.08	81.57	124.03	22	2
	12	13.77	9.20	31.60	84.49	132.03	22	2
	14	14.62	7.85	31.27	87.17	139.32	23	3
	16	14.05	8.27	30.96	90.35	147.97	22	3
	"noGL"	15.73	2.99	29.21	293.95	914.64	22	7
2	0	7.88	12.81	100.00	100.00	100.00	20	3
	2	8.41	9.76	41.62	70.82	88.73	22	6
	4	9.12	7.48	36.80	71.62	95.63	23	8
	6	9.50	6.3	35.20	73.34	101.35	23	9
	8	9.57	5.42	34.01	76.21	109.63	23	12
	10	9.58	5.18	33.45	78.49	115.96	23	14
	12	9.67	4.91	33.09	80.61	121.70	23	13
	14	9.77	4.64	32.78	82.60	127.30	23	15
	16	9.74	4.40	32.57	84.96	133.76	23	14
	"noGL"	10.00	2.11	31.36	158.19	399.62	23	16
3	0	6.02	4.61	100.00	100.00	100.00	17	15
	2	6.02	4.17	42.41	70.97	88.44	17	16
	4	6.02	3.36	38.40	71.40	93.96	17	16
	6	6.02	2.88	37.03	72.70	98.24	17	16
	8	6.02	2.15	36.15	74.74	104.12	17	16
	10	6.02	2.02	35.72	76.03	107.71	17	17
	12	6.02	1.67	35.36	78.11	113.21	17	17
	14	6.02	1.67	35.16	79.44	116.70	17	17
	16	6.02	1.57	35.01	80.74	120.38	17	17
	"noGL"	6.02	1.01	34.40	101.64	189.45	17	17

on how aggressive we delete nodes. For each problem set, we test several different values of the parameter κ, which controls how aggressive we are with deleting nodes. In addition, we introduce a special level of aggregation, denoted by "noGL," in which we contract all gate and loading nodes, regardless of their in and out degrees.

Table 4 Time Limit

		5 minutes			15 minutes			30 minutes		
PSET	AGG	IMPROV	FEAS	OPT	IMPROV	FEAS	OPT	IMPROV	FEAS	OPT
1	0	0.00	0	0	0.26	1	0	3.05	5	0
	4	2.72	4	0	4.57	7	0	7.80	13	0
	8	3.48	4	0	6.54	10	0	10.03	18	2
	12	4.46	7	0	7.77	12	0	11.76	20	1
	16	3.97	6	2	8.19	14	3	10.77	17	3
	"noGL"	4.70	6	4	7.65	10	5	12.44	18	6
2	0	3.33	9	2	4.71	12	2	5.52	12	2
	4	5.97	15	3	6.99	18	4	8.22	18	4
	8	7.30	19	6	7.88	21	8	8.50	21	8
	12	7.30	19	9	8.24	22	10	9.35	22	10
	16	7.56	19	10	8.55	22	12	9.03	22	12
	"noGL"	8.68	21	12	9.93	23	13	9.96	23	13
3	0	4.43	17	12	4.45	17	13	4.45	17	15
	4	4.45	17	14	4.45	17	16	4.45	17	16
	8	4.45	17	15	4.45	17	16	4.45	17	16
	12	4.45	17	16	4.45	17	16	4.45	17	17
	16	4.45	17	16	4.45	17	16	4.45	17	16
	"noGL"	4.45	17	16	4.45	17	16	4.45	17	17

The results are shown in Table 3. In addition to the information provided for the base experiment, we include the level of aggregation (AGG), i.e., the parameter κ, the number of columns in the formulation after aggregation relative to the number of columns in the formulation before aggregation (COLS), the number of rows in the formulation after aggregation relative to the number of rows in the formulation before aggregation (ROWS), and the number of nonzeros in the formulation after aggregation relative to the number of nonzeros in the formulation before aggregation (NZS).

The impact of the aggregation algorithm is most apparent in the first problem set, i.e., for the larger instances. In fact, for this problem set, using the noGL level of aggregation results in an improved solution for 22 out of the 23 instances, compared to only 11 improved solutions without aggregation, and an average reduction of flying time of 15.73%, which is more than twice the reduction obtained without aggregation (6.16%). Furthermore, we see that the reductions in flying time increase gradually from 6.16% to 15.73% as we increase the level of aggregation and we observe a similar change, although more pronounced (from 26.2% to 2.99%), for the decrease in the integrality gap. The fact that the decrease in integrality gap is more pronounced indicates that aggregation not only improves the IP solutions found, but also the LP solutions found. This same phenomenon can be observed for the second and third problem sets. In the third problem set, with the smallest instances, there is the least amount of change. In fact, the aggregation does not increase the number of instances for which an improvement is found and the reduction in flying time remains the same. Only the integrality gap becomes smaller (reduces from 4.61% to 1.01%). This suggests that the quality of the schedules produced by the DayJet heuristic degrades as the instances become larger, thus leaving more room for improvement by our optimization technology. In fact, we believe that for the third problem set, we have produced the optimal schedule for all instances, which means that the DayJet heuristic found the optimal schedule for six instances. It is important to note that although the use of the aggregation algorithm results in improved performance, it comes at a price. The problem size increases, especially in terms of number of nonzeros, when the aggregation level increases, which affects the time it takes to solve linear programs.

The time it takes to solve linear programs during the search, especially the time it takes to solve the linear program at the root, may affect the time it takes to reach a first or substantially improved solution. This time is important because we are also

developing parallel local search technology to solve large-scale instances (see Espinoza et al. 2008). To get some insight into how quickly improved solutions are obtained, we conducted an experiment in which the running time of the search was limited to 5, 15, and 30 minutes. The results are given in Table 4.

The results show that even if we allow only a limited amount of time we are able to find improvements, but that there is a clear qualitative difference between the improved solutions found after 5, 15, and 30 minutes. Furthermore, there is no longer a gradual monotonic change in the number of instances for which an improvement is found and in the reduction of flying time when we increase the level of aggregation. In fact, for problem set 1, the largest number of instances for which an improvement is found occurs with aggregation level 12 instead of with noGL. This is probably due to the fact that a larger part of the search space can be explored due to faster linear programming solving.

In the final experiment, we analyze the importance of the selected time discretization. We chose aggregation level 10 for all experiments, and as before, a time limit of four hours was imposed. The results are presented in Table 5. The second column (DISC) specifies the discretization parameter Δ used.

At first glance, the results may appear counterintuitive because decreasing the level of granularity does not lead to improved solutions. In fact, the worst results are obtained with the finest level of discretization (note that for problem set 1 with discretization level 1, an improvement was found for only one of the 23 instances!), and the level of discretization does not seem to have much of an impact for discretization levels greater than or equal to 15 (for problem set 1, the improvements found range between 13.05 and 14.39). This demonstrates, most likely, that the set of "special" time instants added to the time instants of the discretization are crucial to obtaining high-quality schedules. The results also clearly demonstrate that the problem size explodes at discretization levels of five or less. Given a limited amount of computing time, it appears that a smaller formulation more than makes up for the added precision in finer discretizations and allows us to find improving solutions more often and with larger reductions in flying time.

Table 5 Discretization

PSET	DISC	IMPROV	LP-GAP	NZ	COLS	ROWS	FEAS
1	1	0.85	18.01	771.04	797.85	765.30	1
	5	9.97	14.83	238.64	242.31	239.10	15
	10	11.81	12.49	164.35	166.11	164.98	19
	15	13.88	9.66	141.94	142.80	142.36	22
	30	13.05	10.71	118.34	118.59	118.55	20
	60	13.56	9.93	108.24	108.36	108.40	21
	120	14.39	8.80	103.58	103.78	103.70	22
	240	13.89	9.43	101.90	102.10	101.96	21
	1,440	14.30	8.87	100.00	100.00	100.00	22
2	1	6.26	10.04	923.26	980.63	917.70	16
	5	8.96	6.67	277.48	287.79	279.81	21
	10	9.25	6.39	184.24	188.59	185.66	22
	15	9.93	5.25	155.15	157.58	155.93	23
	30	9.55	5.53	123.61	124.77	124.14	22
	60	9.52	5.38	111.06	111.33	111.22	23
	120	9.28	5.64	104.92	105.01	104.98	22
	240	9.68	5.11	102.59	102.73	102.64	23
	1,440	9.58	5.18	100.00	100.00	100.00	23
3	1	6.46	1.93	1,075.14	1,153.60	1,061.50	17
	5	6.45	1.88	324.88	340.28	328.39	17
	10	6.13	1.99	210.23	215.40	211.61	17
	15	6.13	2.06	174.21	176.74	175.20	17
	30	5.75	1.96	132.14	132.24	132.25	17
	60	5.71	2.03	115.61	114.87	115.22	17
	120	5.68	2.06	106.67	107.03	106.88	17
	240	5.68	2.02	103.97	104.01	103.95	17
	1,440	5.68	2.07	100.00	100.00	100.00	17

DayJet expects to operate with a fleet of 300 jets by 2008 with an expectation that the fleet should be several times larger by 2011. Therefore, DayJet needs schedule optimization technology that can efficiently handle instances of the dial-a-flight problem involving hundreds of jets and thousands of requests. The integer multicommodity network flow model with side constraints discussed above cannot be used directly to satisfy their needs. In Part II, we demonstrate that by embedding this core optimization technology in a parallel local search scheme, it is possible to produce high-quality solutions efficiently for large-scale real-life instances.

Acknowledgments

The authors would like to thank the DayJet corporation for financial support of this research and data, and the DayJet team consisting of Ed Iacobucci, Alex Khmelnitsky, Bruce Sawhill, Bob Spaulding, and Eugene Taits for many valuable and stimulating discussions.

Appendix

Processing Standby Nodes

Consider a jet j at airport a at time t. Assume that this jet has no passengers on board and that it is ready for takeoff. There are two possible decisions regarding the jet: either wait at airport a until some future time t' or fly to some other airport.

If the jet waits, then we may assume it is doing so for one of two reasons: (i) it is waiting to pick up some passenger whose origin is a, but whose earliest departure time is greater than t, or (ii) it is currently at its home base and will not be operated anymore during the day. Observe that we are applying (P3) when making these assumptions. There is no reason for jet j to wait if after doing so it will only carry passengers available at time t. If the jet is to only satisfy requests already available, it will do so immediately.

If the jet is to depart somewhere it will do so for one of three reasons: either (i) the jet will relocate empty to pick

up passengers at another airport, (ii) the jet will load some passengers at this airport and take them somewhere (either their destination, or some middle stop where additional passengers could be picked up), or (iii) the jet will pick up no more passengers during the day and will relocate empty to its home base. Note that here we can apply property (P2). Ideally, we would like to allow the jet to relocate empty only if it has just arrived with passengers on board. This is where the labels come into play. If (a, t) is labeled as a *final stop*, then there exists some itinerary in which jet j arrives with passengers to airport a and is next ready to fly at t. In this case, relocation without loading must be allowed at the node. Otherwise we can prohibit it.

Formally, this is done as follows. Consider a node $S_j(a, t)$ to be processed.

1. Add an idle arc to model that the jet may wait for passengers not yet available for pickup. Let t_2 be the next moment in time after t at which a new passenger is available for pickup in airport a. Formally, let $t_2 = \min\{t' > t\colon \mathcal{R}[t', a] \backslash \mathcal{R}[t, a] \neq \emptyset\}$. If t_2 exists, then add $(S_j(a, t), S_j(a, t_2))$.

2. If at the home base, add an idle arc connecting to the terminal node so as to allow the jet to retire for the day. That is, if a is the home base of the jet, then add $(S_j(a, t), T_j^e)$.

3. Add departure arcs to allow satisfaction of requests whose origin is airport a. That is, for every $b \in \mathcal{A}$ such that $\mathcal{R}[a, t, b]$ is nonempty, add $(S_j(a, t), G_j(a, t, b))$. For every $c \in \mathcal{A}$ and $t_2 \in \mathcal{T}$ such that $\mathcal{R}[a, t, b, t_2, c]$ is nonempty, add $(S_j(a, t), G_j(a, t, b))$.

4. Add departure arcs allowing the jet to relocate after passengers have been dropped off. That is, if (a, t) has a *final-stop* label, then add $(S_j(a, t), G_j(a, t, b))$ for every $b \in \mathcal{A}$.

Processing Gate Nodes

Consider a jet j that is at airport a at time t_1. Assume that it has been decided that this jet will immediately fly to airport b. The next decisions correspond to which passengers (if any) will be loaded.

If no passengers are to be loaded, then by (P2) we may assume that passengers have just been dropped off. In this case we may further assume that the jet is either (i) relocating to pick someone up at airport b, (ii) relocating to its home base to finish up for the day, or (iii) taking indirect passengers to their destination. If the jet is relocating (as opposed to dropping off indirect passengers) and not retiring for the day, it will not fly again from b until some passenger is picked up there; that is, we can assume the jet has no need to fly two consecutive relocation flights. Note that because of (P2) we know that case (i) can only happen if (a, t_1) has a *final-stop* label, and case (iii) can only happen if (a, t_1) has a *middle-stop*(b) label.

If passengers are to be loaded, then either (i) all of them have destination b, or (ii) there exists some airport c such that all of them have destination b or c. In case we decide to load passengers whose final destination is c, we must decide at what time to depart from b. We may decide to take off from b as soon as possible, or we may decide to wait to pick up some passenger that become available later in the day at b.

Formally, this is done as follows. Consider a node $G_j(a, t_1, b)$ to be processed. Let

$$t_2 = \lceil t_1 + \text{RELOCATE\_TIME}(a, b) \rceil,$$

i.e., the first time at which jet j will be available for takeoff from airport b after flying there from a at time t_1. Let $t_2^+ = \min\{t \in \mathcal{T}(j, b)\colon t \geq t_2$ and $\mathcal{R}[b, t]$ is nonempty$\}$, i.e., the first time (no earlier than t_2) at which a request becomes available for pickup at airport b; if no such time exists, set $t_2^+ = +\infty$.

1. Check whether the jet should be allowed to take off without picking up any passengers. If (a, t_1) has a *final-stop* label, then add relocation arc $(G_j(a, t_1, b), S_j(b, t_2^+))$ to model that the jet may take off without having loaded any passengers, and will be ready at airport b as soon as there are passengers available there (and no earlier). If (a, t_1) has a *middle-stop*(b) label, then add indirect direct flight arc $(G_j(a, t_1, b), S_j(b, t_2))$ to model that the jet may take off without having loaded any passengers (to drop off connecting passengers) and be ready at b as early as possible (in case it needs to relocate). Mark (b, t_2) as a *final stop* if an arc was added.

2. Check to see if the jet can load any passengers that will fly directly to their destination. For every request $r \in \mathcal{R}[a, t_1, b]$ (if any), add loading arc $(G_j(a, t_1, b), DL_j(a, t_1, b, r))$. By this, we are allowing jet j to load any request r whose origin is a, whose destination is b, and that can be satisfied by the flight departing at t_1.

3. Check whether the jet can load any passengers that will fly indirectly to their destination. Let $\Gamma = \{t_2\} \cup \{t > t_2 :$ $\mathcal{R}[b, t] \backslash \mathcal{R}[b, t-1]$ is nonempty$\}$, i.e., the time instants at which new requests become available for pickup at airport b. For every $t_3 \in \Gamma$, every $c \in \mathcal{A}$, and every $r \in \mathcal{R}[a, t_1, b, t_3, c]$ (if any), add loading arc $(G_j(a, t_1, b), IL_j(a, t_1, b, t_3, c, r))$. Mark (b, t_3) as a *middle-stop*(c) if an arc was added.

4. If no arcs were added out of node $G_j(a, t_1, b)$, eliminate the node.

Processing Direct Loading Nodes

Consider a jet j that is at airport a. Assume it has been decided that: (i) j will fly to airport b at time t_1, and (ii) j will satisfy a request r whose origin is a and whose destination is b. After these decisions have been made, it is possible to decide that, (iii) j will satisfy another request r_2 whose origin is a and whose destination is b, (iv) j will satisfy a request r_2 whose origin is a and whose destination is not b, or (v) j will satisfy no more requests and fly directly to b.

Formally, this is done as follows. Consider a node $DL_j(a, t_1, b, r)$ to be processed. Let $t_2 = \lceil t_1 + \text{RELOCATE\_TIME}(a, b) \rceil$, i.e., the first time at which jet j will be available for takeoff from airport b after flying there from a at time t_1.

1. Model that we may wish to satisfy additional requests by direct flights. For every request $r' \in \mathcal{R}[a, t_1, b]$ such that $r' > r$, add loading arc $(DL_j(a, t_1, b, r), DL_j(a, t_1, b, r'))$.

2. Model that we may wish to satisfy additional requests by indirect flights. Let $\Gamma = \{t_2\} \cup \{t > t_2\colon \mathcal{R}[b, t] \backslash \mathcal{R}[b, t-1]$ is nonempty$\}$, i.e., the time instants at which new requests become available for pickup at airport b. For every $t_3 \in \Gamma$, every $c \in \mathcal{A}$ and every $r' \in \mathcal{R}[a, t_1, b, t_3, c]$ (if any), add loading arc $(DL_j(a, t_1, b, r), IL_j(a, t_1, b, t_3, c, r'))$.

3. Model that we may not wish to satisfy requests with this flight. Add direct flight arc $(DL_j(a, t_1, b, r), S_j(b, t_2))$. Mark (b, t_2) as a *final stop*.

Processing Indirect Loading Nodes

Consider a jet j that is at airport a. Assume it has been decided that (i) j will fly from airport a to airport b at time t_1, (ii) j will fly from airport b to airport c at time t_2, and (iii) j will satisfy a request r whose origin is a and whose destination is c. After these decisions have been made, it is possible to decide that (iv) j will satisfy another request r_2 whose origin is a and whose destination is c, or (v) j will satisfy no more requests and fly directly to b.

Formally, this is done as follows. Consider a node $IL_j(a, t_1, b, t_2, c, r)$ to be processed.

1. Model that we may wish to satisfy additional requests by indirect flights. For every request $r' \in \mathcal{R}[a, t_1, b, t_2, c]$ such that $r' > r$, add loading arc $(IL_j(a, t_1, b, t_2, c, r), IL_j(a, t_1, b, t_2, c, r'))$.

2. Model that we may not wish to satisfy more requests with this flight. Add indirect flight arc $(IL_j(a, t_1, b, t_2, c, r), G_j(b, t_2, c))$. Mark (b, t_2) as a *middle stop*(c).

References

Benoist, T., E. Bourreau, Y. Caseau, B. Rottembourg. 2001. Towards stochastic constraint programming: A study of online multi-choice knapsack with deadlines. *Proc. Internat. Conf. Constraint Programming* (*CP-2001*), *Lecture Notes in Computer Science*, Vol. 2239. Springer-Verlag, London, 61–76.

Bent, R., P. Van Hentenryck. 2003. A two-stage hybrid algorithm for pickup and delivery problems with time windows. *Lecture Notes in Computer Science 2833. Proc. Internat. Conf. Constraint Programming* (*CP-2003*), *Lecture Notes in Computer Science*, Vol. 2833. Springer-Verlag, Berlin, 123–137.

Bent, R., P. Van Hentenryck. 2004. Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Oper. Res.* **52**(6) 977–987.

Bent, R., P. Van Hentenryck. 2005. A two-stage hybrid algorithm for the pickup and delivery vehicle routing problem with time windows. *Comput. Oper. Res.* **33**(4) 875–893.

Borndorfer, R., M. Grotschel, F. Klostermeier, C. Kuttner. 1997. Telebus Berlin: Vehicle scheduling in a dial-a-ride system. Preprint SC 97-23, Konrad-Zuse-Zentrum fur Informationstechnik, Berlin.

Campbell, A., M. Savelsbergh. 2005. Decision support for consumer direct grocery initiatives. *Transportation Sci.* **39**(3) 313–327.

Cordeau, J.-F. 2006. A branch-and-cut algorithm for the dial-a-ride problem. *Oper. Res.* **54**(3) 573–586.

Cordeau, J.-F., G. Laporte. 2003. The dial-a-ride problem (DARP): Variants, modeling issues and algorithms. *4OR—Quart. J. Belgian, French, Italian Oper. Res. Soc.* **1**(2) 89–101.

Cordeau, J.-F., G. Laporte, J.-Y. Potvin, M. W. P. Savelsbergh. 2007. Transportation on demand. C. Barnhart, G. Laporte, eds. *Handbooks in Operations Research and Management Science*: *Transportation*. Elsevier, Amsterdam, 429–466.

DayJet Corporation. 2005. How to keep air transportation moving at the speed of business. White paper, http://www.dayjet.com/Resources/BusinessTravelers/WhitePapers.aspx.

Desaulniers, G., J. Desrosiers, A. Erdman, M. M. Solomon, F. Soumis. 2002. VRP with pickup and delivery. P. Toth, D. Vigo, eds. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 137–153.

Dumas, Y., J. Desrosiers, F. Soumis. 1991. The pickup and delivery problem with time windows. *Eur. J. Oper. Res.* **54** 7–22.

Espinoza, D., R. Garcia, M. Goycoolea, G. L. Nemhauser, M. W. P. Savelsbergh. 2008. Per-seat, on-demand air transportation part II: Parallel local search. *Transportation Sci.* **42**(3) 279–291.

Fallows, J. 2005. Fly me to the moon? No, but the next best thing. *New York Times* (July 10). http://www.nytimes.com/2005/06/10/business/yourmoney/10flies.html.

Fukasawa, R., H. Longo, J. Lysgaard, M. Poggi de Arago, M. Reis, E. Uchoa, R. F. Werneck. 2006. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Math. Programming* **106** 491–511.

Hicks, R., R. Madrid, C. Milligan, R. Pruneau, M. Kanaley, Y. Dumas, B. Lacroix, J. Desrosiers, F. Soumis. 2005. Bombardier flexjet significantly improves its fractional ownership operations. *Interfaces* **35**(1) 49–60.

Keskinocak, P., S. Tayur. 1998. Scheduling of time-shared jet aircraft. *Transportation Sci.* **32**(3) 277–294.

Lysgaard, J., A. N. Letchford, R. W. Eglese. 2004. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Math. Programming* **100** 423–445.

Martin, C., D. Jones, P. Keskinocak. 2003. Optimizing on-demand aircraft schedules for fractional aircraft operators. *Interfaces* **33**(1) 22–35.

Ropke, S., D. Pisinger. 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Sci.* **40**(4) 455–472.

Savelsbergh, M. W. P., M. Sol. 1995. The general pickup and delivery problem. *Transportation Sci.* **29**(1) 17–29.

Savelsbergh, M. W. P., M. Sol. 1998. DRIVE: Dynamic routing of independent vehicles. *Oper. Res.* **46**(4) 474–490.

United States Department of Transportation. 2003. America on the go: U.S. business travel, http://www.bts.gov.

Van Hentenryck, P., R. Bent, T. Vergados. 2006. Online stochastic reservation systems. *Second Internat. Conf. Integration of AI and OR Techniques in Constraint Programming for Combin. Optim. Problems* (*CP-AI-OR-06*), *Lecture Notes in Computer Science*, Vol. 3990. Springer, Berlin/Heidelberg, 212–227.

Xu, H., Z. L. Chen, S. Rajagopal, S. Arunapuram. 2001. Solving a practical pickup and delivery problem. *Transportation Sci.* **37**(3) 347–364.

Yao, Y., W. Zhao, O. Ergun, E. Johnson. 2005. Crew pairing and aircraft routing for on-demand aviation with time windows. SSRN Working Paper Series, http://ssrn.com/abstract=822265.