TRANSPORTATION SCIENCE
Vol. 00, No. 0, Xxxxx 0000, pp. 000–000
ISSN 0041-1655 | EISSN 1526-5447 | 00 | 0000 | 0001

INFORMS
DOI 10.1287/xxxx.0000.0000

The Airlift Planning Problem *

Dimitris Bertsimas
Sloan School of Management and Operations Research Center, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA, 02139; dbertsim@mit.edu

Allison Chang
Lincoln Laboratory, Massachusetts Institute of Technology; 244 Wood Street, Lexington, MA, 02420; aachang@ll.mit.edu

Velibor V. Mišić
Anderson School of Management, University of California, Los Angeles, 110 Westwood Plaza, Los Angeles, CA, 90095; velibor.misic@anderson.ucla.edu

Nishanth Mundru
Operations Research Center, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA, 02139; nmundru@mit.edu

The United States Transportation Command (USTRANSCOM) is responsible for planning and executing the transportation of United States military personnel and cargo by air, land and sea. The airlift planning problem faced by the air component of USTRANSCOM is to decide how requirements (passengers and cargo) will be assigned to the available aircraft fleet and the sequence of pickups and dropoffs that each aircraft will perform in order to ensure that the requirements are delivered with minimal delay and with maximum utilization of the available aircraft. This problem is of significant interest to USTRANSCOM due to the highly time-sensitive nature of the requirements that are typically designated for delivery by airlift, as well as the very high cost of airlift operations. At the same time, the airlift planning problem is extremely difficult to solve due to the combinatorial nature of the problem and the numerous constraints present in the problem (such as weight restrictions and crew rest requirements). In this paper, we propose an approach for solving the airlift planning problem faced by USTRANSCOM based on modern, large-scale optimization. Our approach relies on solving a large-scale mixed-integer programming model that disentangles the assignment decision (which aircraft will pickup and deliver which requirement) from the sequencing decision (in what order the aircraft will pickup and deliver its assigned requirements), using a combination of heuristics and column generation. Through computational experiments with both a simulated data set and a planning data set provided by USTRANSCOM, we show that our approach leads to high-quality solutions for realistic instances (e.g., 100 aircraft and 100 requirements) within operationally feasible time frames. Compared to a baseline approach that emulates current practice at USTRANSCOM, our approach leads to reductions in total delay and aircraft time of 8 to 12% in simulated data instances and 16 to 40% in USTRANSCOM's planning instances.

Key words: pickup and delivery with time windows; mixed-integer programming; column generation; local search; construction heuristics

1. Introduction

The United States Transportation Command (USTRANSCOM) is the main entity responsible for the transportation of personnel and cargo for the United States military across the globe. Of the three modes of transportation employed by USTRANSCOM – air, land and sea – transportation by air is often the most efficient means of delivering requirements (passengers and cargo) and thus is often the mode of choice for many high-priority and short-notice missions.

The planning of airlift missions is of critical importance to USTRANSCOM, for two reasons. First, due to the time-sensitive nature of requirements transported by air, a primary concern for

* This material is based upon work supported under Air Force Contract No. FA8721-05-C-0002. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the U.S. Air Force.

USTRANSCOM is that requirements be picked up and delivered within their predefined time windows with minimal or no delay. Second, transporting requirements by air is expensive: a recent RAND study estimates that airlift missions cost between \$9,000 and \$12,000 per flying hour for certain aircraft types (Robbert 2013). By carefully planning its airlift missions, USTRANSCOM could potentially ensure that all requirements are delivered on time, while doing so with a more efficient use of its aircraft – more precisely, using fewer aircraft and through shorter missions.

At the same time, planning efficient airlift missions is difficult. The primary reason for this is the combinatorial nature of the problem: planners must decide which requirements will be picked up and delivered by which aircraft, and in what sequence those aircraft will pick up and deliver those requirements. This naturally leads to an extremely large number of possible schedules. A secondary reason is that airlift missions are governed by a multitude of constraints: requirements must be picked up and delivered within their defined time windows; the total weight of the requirements being transported by each aircraft at any time cannot exceed the weight capacity of that aircraft; and the schedule must respect aircraft crew rest constraints. At present, planners schedule each requirement one at a time, and look for opportunities to combine some missions with others to reduce transportation costs. However, this process is largely manual and ad hoc, and there is a need for a decision support infrastructure to systematically design schedules that deliver requirements with minimal delay and with maximum utilization of the available aircraft.

In this paper, we present a methodology for designing airlift schedules for USTRANSCOM based on modern, large-scale optimization. This method is based on solving a large-scale mixed-integer programming (MIP) model of the problem using a combination of heuristics – an initialization heuristic and a local search heuristic – and column/constraint generation. This method is capable of efficiently producing high-quality schedules – that is, ones that deliver requirements on time or with minimal delay, with a minimal amount of aircraft hours.

We make the following contributions:

1. We propose an MIP formulation of the airlift planning problem. This formulation jointly decides the assignment of requirements to aircraft and the sequence of pickups and dropoffs so as to minimize a weighted combination of delay and uptime (the total aircraft-hours required by the schedule). Our formulation has a number of features not typically found in prior models for pickup and delivery problems, such as the ability of aircraft to rest, constraints on rest periods and active times, and time windows that are both hard and soft; we show how to model these various features using the language of MIP.

2. Motivated by the difficulty of the original MIP problem, we propose a reformulation of the problem as a large-scale MIP, where the decision variables correspond to assigning a set of requirements and the sequencing of each aircraft's mission is captured through the objective function coefficients. This reformulation allows us to decouple the decision of which requirements will be assigned to which aircraft from the decision of how to sequence the pickup and dropoff events of each aircraft. We propose a three-phase method for solving this reformulated problem: in the first phase, we run an initialization heuristic that constructs an initial feasible assignment of requirements to aircraft; in the second phase, we improve this assignment using a local search heuristic; and in the final phase, we use column generation to further improve the solution and to also obtain a lower bound on the quality of the ultimate solution. Unlike most prior approaches to pickup and delivery problems, our three-phase method makes use of a smaller version of our original MIP problem for sequencing each aircraft; this is advantageous, because each aircraft is sequenced optimally, but also necessary in order to respect the operational constraints required by USTRANSCOM.

3. We demonstrate the value of this method through computational experiments with both a simulated data set and a planning data set provided to us by USTRANSCOM. Using our two heuristics, we are able to obtain high quality solutions for large instances (100 aircraft and 100 requirements) within six hours; the column generation algorithm we propose further improves these solutions and provides an approximation of the suboptimality gap, which is below 1.5% on average for the simulated data instances, and 7% on average for the USTRANSCOM planning

instances. More importantly, the solutions produced by our method significantly outperform a baseline approach that emulates current practice at USTRANSCOM; in comparison, our overall method leads to average reductions in total delay and uptime of 8 to 12% in the simulated data instances and 16 to 40% in USTRANSCOM's internal planning instances.

The rest of the paper is organized as follows. In Section 2, we present a survey of related work. In Section 3, we define the problem and provide an initial MIP formulation of the problem. In Section 4, we present a large-scale optimization approach for efficiently solving the problem defined in Section 3. We present the results of our computational study in Section 5. Finally, in Section 6, we conclude.

2. Literature review

We divide our review of the literature in two parts. We first survey the large body of work on vehicle routing problems, which are closely related to the airlift planning problem. We then survey related work in other areas, specifically in air traffic management and dynamic resource allocation.

Vehicle routing. The airlift planning problem falls in the general category of pickup and delivery problems with time windows (PDPTW). The PDPTW is a generalization of the vehicle routing problem, and consequently is also an NP-hard problem (Garey and Johnson 1979). The vehicle routing problem involves designing a set of routes, originating and terminating at a single depot, for a fleet of vehicles that service a set of customers with known demands, with each customer being serviced exactly once. For a general survey of vehicle routing problems, we direct the reader to Laporte (2007) and Toth and Vigo (2014).

PDPTW involves designing a set of minimum-cost routes to satisfy transportation requests (requirements). Each requirement has a pickup and dropoff location, and a corresponding size. The decision maker has a fleet of vehicles, each with some capacity and predefined start and end locations. Each requirement has to be transported by one vehicle from its pickup location (origin) to its dropoff location (destination). The routes must satisfy the precedence relations of pickup and delivery points, along with the time windows imposed by them. For a comprehensive review of the PDPTW literature, we refer the reader to Desrochers et al. (1988), Savelsbergh and Sol (1995), Cordeau et al. (2004) and Toth and Vigo (2014, Chap. 6 and 7). In the PDPTW, all vehicles depart from and return to a central depot. The airlift planning problem is best described as a PDPTW with multiple depots, along with other real life constraints/features such as maximum active time (the aircraft cannot operate beyond a certain amount of time without initiating a rest period) and minimum rest time (when the aircraft rests, it must rest for at least some minimum number of hours). In addition, the time constraints in our problem have both a "hard" component (each requirement must be picked up/dropped off within some time window) and a "soft" component (if a requirement is picked up/dropped off past a certain time within that time window, it is considered late, and the objective penalizes this lateness).

Many heuristics have been proposed for the PDPTW problem, such as tabu search (Nanry and Barnes 2000), simulated annealing (Bent and Van Hentenryck 2006) and large neighborhood search (Ropke and Pisinger 2006). Within this body of work, there exist some approaches that are similar to our initialization and local search heuristics (Algorithms 3 and 4 in Section 4.4 respectively). For example, in Nanry and Barnes (2000), the initial feasible solution is produced by an insertion algorithm that attempts to insert a pickup-delivery pair onto the first vehicle and if more than one feasible insertion point exists, then the procedure picks the insertion that least increases the partial solution's travel time; if no feasible insertion point exists, then a new vehicle is added to the solution. This procedure resembles our initialization heuristic, which also builds a partial assignment of requirements to aircraft and seeks to assign a requirement in a way that least increases the total objective of the partial solution, while maintaining feasibility. Nanry and Barnes (2000) also discuss a type of neighborhood used in their tabu search which involves removing a pickup-delivery pair from its current vehicle route and inserting it into one of the other vehicle routes; this procedure resembles our local search heuristic, which also involves moving one requirement from its current aircraft and assigning it to a different aircraft. One major difference between the heuristics we consider from those considered before is that in most previously considered heuristics, the insertion of a pickup-delivery pair into a vehicle route leaves the order of the other pickup-delivery pairs intact; stated differently, one does not re-optimize the sequence of pickup-delivery pairs after each insertion. In contrast, our heuristics (Algorithm 3 and 4) involve exactly re-optimizing each aircraft's sequence whenever the set of requirements assigned to that aircraft is changed (when a requirement is added to the aircraft, or a requirement is removed from the aircraft). This is advantageous because it does not leave value on the table, so to speak, with regard to the sequence of each aircraft; each aircraft's sequence achieves the lowest possible objective value. From an implementation point of view, inserting a pickup and a dropoff into an existing event sequence in the airlift planning problem is also not straightforward to do due to the constraints on weight and active time, and also due to the fact that aircraft may choose to rest for a variable time period between events.

Outside of heuristic methods, a number of previous studies have considered the use of column generation methods and branch-and-price methods for solving the PDPTW exactly. Column generation was first applied to PDPTW by Dumas, Desrosiers, and Soumis (1991); they formulated the problem as a set partitioning problem, where each column maps to a feasible vehicle route and each constraint corresponds to a request that must be satisfied exactly once, and they solved the pricing subproblem using dynamic programming and label elimination methods. Savelsbergh and Sol (1998) later proposed a branch-and-price method for solving the problem, where the pricing subproblem is solved using a combination of construction and improvement heuristics. Ropke and Cordeau (2009) propose a branch-cut-price approach. In their approach, the pricing subproblem is formulated as a constrained shortest path problem; when it needs to be solved exactly, it is solved using labeling algorithms, and otherwise it is solved using a combination of large neighborhood search, label heuristics and heuristics based on construction and improvement. Other examples of column generation approaches for PDPTW include Xu et al. (2003) and Sigurd, Pisinger, and Sig (2004).

The column generation approach that we consider in this paper is similar to these previously proposed approaches; the master problem that we solve is effectively a set partitioning problem, where the variables correspond to whether an aircraft a is assigned to a set of requirements S and we must iteratively add (a, S) pairs using column generation. However, our approach differs in several ways. First, we use column generation for the purpose of improving the solution obtained by our heuristics and for obtaining a lower bound on the quality of that solution; we do not embed our column generation within a branch-and-price scheme in order to solve the problem exactly. Second, like some prior applications of column generation to PDPTW, we also use a heuristic approach for solving the pricing subproblem, but our approach differs in that it is a set-based local search. More precisely, we perform a local search over sets of requirements S that lead to the lowest reduced cost, where for each set S we solve a single-aircraft sequencing problem in order to compute the reduced cost (cf. our comparison of our heuristic algorithms); in other words, our subproblem heuristic exactly re-optimizes the actual sequence of pickups and dropoffs for each set of requirements (i.e., the heuristic does not additionally involve a local search over the sequence of pickups and dropoffs). We are not aware of a similar approach in the vehicle routing literature. Outside of the vehicle routing literature, a similar algorithm was proposed in the operations management literature, in the context of finding an assortment (set of products) that maximizes a black-box revenue function and was shown to perform well for a variety of revenue functions (Jagabathula 2014).

Some papers have also considered mixed-integer programming approaches not based on set partitioning and column generation. Ropke, Cordeau, and Laporte (2007) consider a branch-and-cut approach for solving a "two-index" formulation of the PDPTW, where the binary decision variables indicate whether a given pickup/delivery is performed immediately before a different pickup/delivery. Lu and Dessouky (2004) solve the multiple vehicle pickup and delivery problem using branch-and-cut. Mahmoudi and Zhou (2016) consider the PDPTW with many vehicles and propose a multi-commodity flow formulation of it using a state-space-time network representation; they consider a Lagrangian relaxation of the multi-commodity flow problem, which leads to a collection of single vehicle routing subproblems that can be solved using a forward dynamic programming algorithm.

Some work in PDPTW has focused on air transportation. One recent example is the paper of Carnes et al. (2013), which considers the assignment and sequencing of aircraft for performing non-emergency transportation of patients between different facilities. The problem has some similarities to the one that we consider. For example, each patient is analogous to a requirement in our context, and each aircraft is limited in how many requests (requirements/patients) that it can carry at any time. However, some constraints are different and do not have analogs in our problem – for example, a schedule in Carnes et al. (2013) cannot transport infectious patients with uninfected patients, and must spend additional time on the ground after delivering an infectious patient to be disinfected. Similarly, there is no constraint in Carnes et al. (2013) analogous to our rest and maximum active time constraints. In addition, the solution methods are different. In Carnes et al. (2013), the problem is solved through a set partitioning formulation, but because of the scale of their problem and constraints on how many requests can be assigned to each aircraft, the columns of the formulation can be completely enumerated and their costs precomputed, which allows them to solve the problem directly to full optimality. In contrast, this is not possible in our setting, because of the scale and the lack of a similar constraint on how many requirements can be assigned to a single aircraft; as a result, we must turn to both column generation and initialization/local search heuristics.

We summarize the key differences between our paper and the existing PDPTW literature as follows:

1. **Problem setting**. Our problem contains many elements not typically found in PDPTW problems, such as hard and soft time windows, variable wait times, constraints on maximum active time, and combined delay and uptime as an objective. To the best of our knowledge, no other paper has studied a PDPTW problem with this specific combination of features, at the scale required by USTRANSCOM.

2. **Heuristics**. Both our initialization and local search heuristics involve modifying an assignment of requirements to aircraft, where the value of an assignment is obtained by solving a small, single aircraft MIP problem for each aircraft to sequence the pickups and dropoffs; in this way, the single aircraft MIP problem is used as a black box. This contrasts with many other construction and local search heuristics for PDPTW that do not use MIP, but directly remove pickup/dropoff events from one vehicle's sequence and insert them into the sequence of a different vehicle. Our use of the single aircraft MIP is necessary for our problem setting because, as mentioned earlier, it is not straightforward to insert events into a sequence in a way that respects the various constraints of the problem.

3. **Column generation**. Our column generation approach differs in two ways from existing column generation approaches. First, our master problem is not a pure set partitioning problem, as is often the case in many PDPTW problems, but also involves an assignment decision; this difference is important because the aircraft are heterogeneous in their travel times, capacities and the ports at which they must start and end their missions. Second, the subproblem in our column generation algorithm is solved heuristically, again using the aforementioned single aircraft MIP problem as a black box. As with our initialization and local search heuristics, the use of the single aircraft MIP problem is a practical necessity in order to ensure feasibility with respect to all of the constraints of the problem; the features of our problem complicate the use of traditional methods such as dynamic programming. Third, we use column generation in a heuristic manner, to improve the solution obtained from our local search heuristic, as opposed to using it within a branch-and-price method. As such, our column generation method can be implemented with less effort and requires less time to run than branch-and-price. However, unlike branch-and-price,

the final integer solution produced by our column generation algorithm is not guaranteed to be optimal.

Other areas. Outside of vehicle routing, the present paper is related to some other work in air traffic management and dynamic resource allocation.

The air traffic flow management (ATFM) problem is to decide what flow management actions to take for a set of flights – such as ground holding, airborne holding and rerouting – so as to minimize the aggregate delay of those flights, while respecting constraints on the capacity of each sector of airspace. We focus on some specific examples of work in this area; for a more detailed review of this literature, the reader is referred to Vossen, Hoffman, and Mukherjee (2012). Bertsimas and Stock Patterson (1998) propose an integer programming formulation of the problem where the only allowed interventions are ground holding and speed control, and each aircraft's path is fixed. Bertsimas and Stock Patterson (2000) extended this approach to allow for re-routing decisions by considering a dynamic, multi-commodity flow problem, which they solve using a combination of Lagrangian relaxation, randomized rounding and solving an integer packing problem. Bertsimas, Lulli, and Odoni (2011) build on both of these previous papers by proposing an integer programming model that allows for the full range of flow management actions and that can be solved directly using commercial software for very large-scale instances (for example, the continental United States network).

There are similarities and differences between the airlift planning problem and the ATFM problem studied in the above papers. Our problem bears some resemblance to the ATFM problem in that routing decisions are somewhat similar to the scheduling decisions (what sequence should an aircraft perform its assigned pickups and dropoffs) in the airlift planning problem; unlike the ATFM problem, though, the airlift planning problem additionally involves an assignment decision, which the ATFM problem does not. Another point of difference has to do with whether the aircraft are coupled or not. Although one can consider constraints that couple aircraft together in our problem (such as so-called "maximum-on-ground" constraints, which limit how many aircraft can be present at an airbase at any moment in time), we do not consider such constraints in this paper, and so the aircraft in our model are decoupled. In contrast, the sector capacity constraints in the ATFM problem (i.e., there can be no more than five aircraft in a sector) effectively couple the aircraft together, and one cannot plan the flight path/flow management decisions for one aircraft independently of another. From a modeling perspective, our approach for the airlift planning problem involves an integer programming model where the time of each pickup/dropoff is represented through continuous variables, and the sequence of events is represented through slot-based variables (i.e., event e is the pth event in aircraft a's event sequence). In contrast, time is discretized in Bertsimas and Stock Patterson (1998, 2000) and Bertsimas, Lulli, and Odoni (2011), and the sequence is represented through "by" variables (i.e., flight f reaches sector j by time t). Furthermore, from a solution perspective, our approach decouples the assignment and scheduling decisions and solves the problem using heuristics and column generation; in Bertsimas and Stock Patterson (1998, 2000) and Bertsimas, Lulli, and Odoni (2011), one directly solves the full integer programming formulation of the problem.

Beyond the ATFM problem, our work is related to the more general problem of dynamic resource allocation, where one must allocate requests to resources so as to complete them within some predefined time windows with minimum cost. One salient example in this area is Bertsimas, Gupta, and Lulli (2014), who propose a general integer programming formulation for dynamic resource allocation and a specialized algorithm for solving it based on adjustable time windows, with the ATFM problem as an application of the framework. The formulation in Bertsimas, Gupta, and Lulli (2014) is similar to those in Bertsimas and Stock Patterson (1998, 2000) and Bertsimas, Lulli, and Odoni (2011), in that the main decision variables model whether a request has begun utilizing a resource by some discrete time; as mentioned in the discussion above, this formulation differs significantly from our slot-based formulation.

3. Problem definition

In this section, we define the airlift planning problem. We begin by describing the notation for the parameters and decision variables in Section 3.1. Then, in Section 3.2, we provide a mixed-integer programming (MIP) formulation of the problem, and describe in detail the objective function and the constraints of the formulation.

3.1. Notation

We assume that there are R requirements that must be transported, indexed from 1 to R, and we let $\mathcal{R} = \{1, \ldots, R\}$ denote the set of requirements. We assume that there are A aircraft that can be used to transport the requirements, indexed from 1 to A and let $\mathcal{A} = \{1, \ldots, A\}$ denote the set of aircraft. We will model the problem by defining four primitive events that may occur over the planning horizon:

1. Pickup(r): requirement r is picked up.
2. Dropoff(r): requirement r is dropped off.
3. Start(a): aircraft a begins its mission.
4. End(a): aircraft a ends its mission.

We use $\mathcal{E}$ to denote the set of all possible events:

$$\mathcal{E} = \{\text{Pickup}(r) \mid r \in \mathcal{R}\} \cup \{\text{Dropoff}(r) \mid r \in \mathcal{R}\} \cup \{\text{Start}(a) \mid a \in \mathcal{A}\} \cup \{\text{End}(a) \mid a \in \mathcal{A}\}.$$

Each event may only be performed by one aircraft. All Pickup(r) and Dropoff(r) events must be executed, and may be executed by any aircraft. We assume that the same aircraft used to execute Pickup(r) must also be used to execute Dropoff(r). The Start(a) and End(a) events must be executed by aircraft a if and only if aircraft a is used in the schedule.

Each event e must occur at a specific location or "port". We let $\tau_{e,e',a}$ denote the travel time of aircraft a from the port of event e to the port of event e'. We let ℓ_e denote the earliest time by which event e may be executed, u_e denote the latest time by which event e may be executed without penalty, and B_e denote the maximum slack of event e, that is, B_e is the largest amount of time by which event e may be executed past u_e. Stated differently, an event may be executed at any time t within $[\ell_e, u_e + B_e]$; if t is within $[\ell_e, u_e]$, there is no penalty, but if t is within $(u_e, u_e + B_e]$, the requirement is deemed late, where the delay/slack is given by $t - u_e$. The event e cannot be executed past $u_e + B_e$ or before ℓ_e.

We use T to denote the end of the time horizon, which is the maximum of all of the times by which any event may be executed; mathematically, it is defined as

$$T = \max_{e \in \mathcal{E}} (u_e + B_e).$$

Similarly, we assume that the problem starts at time 0; the interval $[0, T]$ thus defines the planning horizon of the problem.

We let w_e denote the change in an aircraft's weight associated with executing event e. For Start(a) and End(a) events, $w_e = 0$; for Pickup(r) events, $w_{\text{Pickup}(r)} > 0$ (performing a pickup leads to an increase in the carried weight of the aircraft), and for Dropoff(r) events, $w_{\text{Dropoff}(r)} = -w_{\text{Pickup}(r)} < 0$ (performing a dropoff leads to a decrease in the carried weight of the aircraft). For each aircraft $a \in \mathcal{A}$, we use W_a to denote the weight capacity of that aircraft; at any point in time, the total weight carried by aircraft a cannot exceed W_a. We assume that for each requirement r, the weight of that requirement fits within the capacity of at least one of the aircraft (i.e., for each r, there exists an $a \in \mathcal{A}$ such that $w_{\text{Pickup}(r)} \leq W_a$).

Each aircraft is subject to rest constraints. We let γ_a denote the maximum active time of aircraft a, where the active time is defined as the accumulated time that the aircraft has been in the air since its last rest period. We let δ_a denote the minimum rest time of aircraft a; this is the least amount of time that an aircraft must rest on the ground before it can return to flying and executing its remaining events.

The objective that we will optimize consists of a weighted combination of the total delay/slack of all events plus the total uptime of all aircraft used in the schedule, where the uptime of aircraft a is defined as the time from the Start(a) event to the End(a) event of aircraft a. We use π_e to denote the priority of event e; the priority of an event is the weight coefficient of that event in the objective. We use C to denote the objective weight coefficient of the total uptime of all of the aircraft; we assume that $C > 0$.

3.2. Formulation

In this section, we define the airlift planning problem by formulating it as a mixed-integer programming (MIP) problem.

We begin by modeling each aircraft as having a sequence of P slots. Each slot fits at most one event. Since any one aircraft can perform at most $2R+2$ events – the Pickup(r) and Dropoff(r) event for each requirement in $\mathcal{R}$, as well as the Start(a) and End(a) events of that aircraft – we set the number of slots P to be $2R+2$. In addition, for the purpose of defining the MIP formulation, let us also define $\mathcal{E}(\mathcal{R})$ to be the set of pickup and dropoff events for all requirements:

$$\mathcal{E}(\mathcal{R}) = \{\text{Pickup}(r) \mid r \in \mathcal{R}\} \cup \{\text{Dropoff}(r) \mid r \in \mathcal{R}\}.$$

Additionally, let us define $\mathcal{E}(a,\mathcal{R})$ as the set of pickup and dropoff events for all requirements in $\mathcal{R}$, together with the start and end events for aircraft a, formally,

$$\mathcal{E}(a,\mathcal{R}) = \{\text{Pickup}(r) \mid r \in \mathcal{R}\} \cup \{\text{Dropoff}(r) \mid r \in \mathcal{R}\} \cup \{\text{Start}(a), \text{End}(a)\}.$$

We define our decision variables as follows. We let $x_{e,a,p}$ be a binary variable that is 1 if aircraft a executes event e in its pth slot, and 0 otherwise. We let $y_{e,e',a,p}$ be a binary variable that is 1 if aircraft a executes event e in slot p of its sequence and event e' in slot $p+1$ of its sequence, and 0 otherwise. We let $z_{a,p}$ be a binary variable that is 1 if aircraft a rests immediately after executing the event in slot p. We let $t_{a,p}$ be a continuous variable that represents the time at which aircraft a executes the event in slot p. We let $s_{e,a,p}$ be a continuous variable that represents the delay/slack of event e if it is performed by aircraft a in slot p of that aircraft's sequence. Finally, we use $\omega_{a,p}$ to denote the accumulated active time from aircraft a's Start(a) event to immediately before the event in slot p of aircraft a's sequence.

$$\underset{\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{t},\mathbf{s},\boldsymbol{\omega}}{\text{minimize}} \quad \sum_{e\in\mathcal{E}} \pi_e \sum_{a\in\mathcal{A}} \sum_{p=1}^{P} s_{e,a,p} + C \sum_{a\in\mathcal{A}} (t_{a,P} - t_{a,1}) \tag{1a}$$

$$\text{subject to} \quad \sum_{a\in\mathcal{A}} \sum_{p=1}^{P} x_{e,a,p} = 1, \quad \forall\, e \in \mathcal{E}(\mathcal{R}), \tag{1b}$$

$$\sum_{p=1}^{P} x_{\text{Start}(a),a,p} \le 1, \quad \forall\, a \in \mathcal{A}, \tag{1c}$$

$$\sum_{p=1}^{P} x_{\text{End}(a),a,p} = x_{\text{Start}(a),a,1}, \quad \forall\, a \in \mathcal{A}, \tag{1d}$$

$$\sum_{e\in\mathcal{E}(a,\mathcal{R})} x_{e,a,p} \le x_{\text{Start}(a),a,1}, \quad \forall\, a \in \mathcal{A},\ p \in \{1,\ldots,P\}, \tag{1e}$$

$$\sum_{e\in\mathcal{E}(a,\mathcal{R})} x_{e,a,p'} \le 1 - x_{\text{End}(a),a,p}, \quad \forall\, a \in \mathcal{A},\ p \in \{2,\ldots,P-1\},\ p' \in \{p+1,\ldots,P\}, \tag{1f}$$

$$\sum_{e\in\mathcal{E}(a,\mathcal{R})} x_{e,a,p+1} \le \sum_{e\in\mathcal{E}(a,\mathcal{R})} x_{e,a,p}, \quad \forall\, a \in \mathcal{A},\ p \in \{1,\ldots,P-1\}, \tag{1g}$$

$$\sum_{p=1}^{P} x_{\text{Dropoff}(r),a,p} = \sum_{p=1}^{P} x_{\text{Pickup}(r),a,p}, \quad \forall\, r \in \mathcal{R},\ a \in \mathcal{A}, \tag{1h}$$

$$\sum_{p'=1}^{p} x_{\text{Pickup}(r),a,p'} \geq \sum_{p'=1}^{p} x_{\text{Dropoff}(r),a,p'}, \quad \forall\, r \in \mathcal{R},\ a \in \mathcal{A},\ p \in \{1,\dots,P\}, \tag{1i}$$

$$y_{e,e',a,p} \geq x_{e,a,p} + x_{e',a,p+1} - 1, \quad \forall\, a \in \mathcal{A},\ e,e' \in \mathcal{E}(a,\mathcal{R}),\ p \in \{1,\dots,P-1\}, \tag{1j}$$

$$y_{e,e',a,p} \leq x_{e,a,p}, \quad \forall\, a \in \mathcal{A},\ e,e' \in \mathcal{E}(a,\mathcal{R}),\ p \in \{1,\dots,P-1\}, \tag{1k}$$

$$y_{e,e',a,p} \leq x_{e',a,p+1}, \quad \forall\, a \in \mathcal{A},\ e,e' \in \mathcal{E}(a,\mathcal{R}),\ p \in \{1,\dots,P-1\}, \tag{1l}$$

$$\sum_{e \in \mathcal{E}(a,\mathcal{R})} \sum_{p'=1}^{p} w_e x_{e,a,p'} \leq W_a, \quad \forall\, a \in \mathcal{A},\ e \in \mathcal{E}(a,\mathcal{R}),\ p \in \{1,\dots,P\}, \tag{1m}$$

$$t_{a,p+1} \geq t_{a,p} + \sum_{e,e' \in \mathcal{E}(a,\mathcal{R})} \tau_{e,e',a} y_{e,e',a} + \delta_a z_{a,p}, \quad \forall\, a \in \mathcal{A},\ p \in \{1,\dots,P-1\}, \tag{1n}$$

$$t_{a,p} \geq \sum_{e \in \mathcal{E}(a,\mathcal{R})} \ell_e x_{e,a,p}, \quad \forall\, a \in \mathcal{A},\ p \in \{1,\dots,P\}, \tag{1o}$$

$$t_{a,p} \leq \sum_{e \in \mathcal{E}(a,\mathcal{R})} (u_e x_{e,a,p} + s_{e,a,p}) + T\left(1 - \sum_{e \in \mathcal{E}(a,\mathcal{R})} x_{e,a,p}\right), \quad \forall\, a \in \mathcal{A},\ p \in \{1,\dots,P\}, \tag{1p}$$

$$s_{e,a,p} \leq B_e \cdot x_{e,a,p}, \quad \forall\, a \in \mathcal{A},\ e \in \mathcal{E}(a,\mathcal{R}),\ p \in \{1,\dots,P\}, \tag{1q}$$

$$\omega_{a,1} = 0, \quad \forall\, a \in \mathcal{A}, \tag{1r}$$

$$\omega_{a,p} = \omega_{a,p-1} + \sum_{e,e' \in \mathcal{E}(a,\mathcal{R})} \tau_{e,e',a} \cdot y_{e,e',a,p-1}, \quad \forall\, a \in \mathcal{A},\ p \in \{2,\dots,P\}, \tag{1s}$$

$$\omega_{a,q'} - \omega_{a,q} \leq \gamma_a \cdot \left(1 + \sum_{p=q}^{q'-1} z_{a,p}\right), \quad \forall\, a \in \mathcal{A},\ q \in \{1,\dots,P-1\},\ q' \in \{q+1,\dots,P\}, \tag{1t}$$

$$x_{e,a,p} \in \{0,1\}, \quad \forall\, a \in \mathcal{A},\ e \in \mathcal{E}(a,\mathcal{R}),\ p \in \{1,\dots,P\}, \tag{1u}$$

$$z_{a,p} \in \{0,1\}, \quad \forall\, a \in \mathcal{A},\ p \in \{1,\dots,P\}, \tag{1v}$$

$$y_{e,e',a,p} \geq 0, \quad \forall\, e,e' \in \mathcal{E}(a,\mathcal{R}),\ a \in \mathcal{A},\ p \in \{1,\dots,P-1\}, \tag{1w}$$

$$t_{a,p}, \omega_{a,p} \geq 0, \quad \forall\, a \in \mathcal{A},\ p \in \{1,\dots,P\}, \tag{1x}$$

$$s_{e,a,p} \geq 0, \quad \forall\, e \in \mathcal{E}(a,\mathcal{R}),\ a \in \mathcal{A},\ p \in \{1,\dots,P\}. \tag{1y}$$

To understand the formulation, let us first describe each relevant group of constraints, followed by the objective function.

Mission constraints: Constraint (1b) requires that each pickup and dropoff event happens; each such event must be assigned to the slot of some aircraft. Constraint (1c) ensures that the Start(a) event of each aircraft a is executed at most once by aircraft a. Constraint (1d) ensures that aircraft a executes the End(a) event if and only if it executes the Start(a) event in slot 1. Constraint (1e) ensures that in each slot of each aircraft, there can be at most one event if the aircraft executes Start(a) in slot 1, and there is no event if that aircraft does not start in slot 1. Constraints (1e) and (1c) together ensure that an aircraft performs any events if and only if it executes Start(a) in slot 1; in other words, Start(a) is always the event of the first slot of aircraft a if aircraft a is to be used in the solution. Constraint (1f) ensures that there are no events in the slots of aircraft a after the End(a) event (if the aircraft ends its mission, it cannot continue performing pickups and dropoffs). Constraint (1g) requires that if there is an event in slot $p+1$, there must be an event in slot p; stated differently, there cannot be any empty slots between Start(a) and End(a).

Pickup/dropoff constraints: Constraint (1h) ensures that a requirement must be picked up and dropped off by the same aircraft. Constraint (1i) ensures that, in terms of slots, Pickup(r) always comes before Dropoff(r). (The left hand side of the constraint is 1 if Pickup(r) happens *by* slot p and 0 otherwise, and the right hand side is 1 if Dropoff(r) happens by slot p and 0 otherwise; the constraint requires that if a requirement is dropped off by slot p, it must have been picked up by slot p as well.)

Transition constraints: Constraints (1j) through (1l) are forcing constraints that ensure that the y variables take their correct values based on the x variables.

Capacity constraint: Constraint (1m) requires that the weight carried by aircraft a at any slot p cannot exceed the capacity W_a of that aircraft.

Travel time dynamics constraint: Constraint (1n) ensures that the $t_{a,p}$ variables respect travel times. More precisely, this constraint requires that the event in slot $p+1$ of aircraft a can be performed no earlier than the time of slot p of aircraft a plus the correct travel time (encoded by the weighted sum of $y_{e,e',a,p}$ variables) and the minimum rest time δ_a if the aircraft rests after performing the event of slot p.

Time window constraints: Constraint (1o) ensures that the time at which aircraft a performs whatever event is in slot p satisfies the earliest allowable time ℓ_e of that event. Constraint (1p) ensures that the time at which aircraft a performs the event in slot p satisfies the latest allowable time u_e, adjusted by the slack/delay $s_{e,a,p}$ of that event. In the case that there is no event assigned to slot p of aircraft a, these constraints become vacuous; the right-hand side of constraint (1o) becomes 0, while the right-hand side of constraint (1p) becomes T. Constraint (1q) requires that the slack/delay variable $s_{e,a,p}$ is bounded by B_e if event e happens in slot p of aircraft a and is forced to zero otherwise.

Active time constraints: Constraints (1r) and (1s) model the dynamics of the accumulated active time from the start of the aircraft's mission. Constraint (1t) ensures that the active time accumulated between any two periods does not exceed how much active time is afforded by the rests taken over those two periods.

Variable definitions: Finally, constraints (1u) through (1y) define the variables. Note that since the x variables are binary, the forcing constraints (1j) through (1l) ensure that the y variables are automatically forced to their correct binary values; thus, we can model the y variables as continuous variables.

Objective function: The objective (1a) consists of two parts: the first part represents the priority-weighted sum of slacks/delays, while the second part represents the weighted total uptime of all of the aircraft.

Note that since $C > 0$, $t_{a,P} - t_{a,1}$ will always correctly reflect the uptime of aircraft a. If aircraft a is not used, all y variables corresponding to aircraft a will be zero, and $t_{a,1}, \dots, t_{a,P}$ are free to take any values in $[0, T]$, so long as they form an increasing sequence of values; due to the positive weight C, we will have at optimality that $t_{a,1} = \cdots = t_{a,P}$, so that $t_{a,P} - t_{a,1}$ will thus be zero. In the case that aircraft a is used, let p' be the slot at which End(a) occurs; then, for every slot $p \geq p'$, all of the y variables corresponding to those slots are zero, and the variables $t_{a,p'+1}, \dots t_{a,P}$ are free to take any value between $t_{a,p'}$ and T. Since C is positive, we will have at optimality that $t_{a,p'} = t_{a,p'+1} = \cdots = t_{a,P}$; moreover, if aircraft a is used, then the event in slot 1 must be Start(a). Thus, $t_{a,P} - t_{a,1}$ again correctly captures the uptime.

We remark that problem (1) above is formulated in terms of slots. One may wonder if the problem can be formulated more compactly, by using a formulation that models whether event i precedes event j (see, for example, Desrochers et al. 1988, p. 69). In order to model event times correctly in such a framework, one must use big-M constraints, which generally leads to weaker MIP formulations. Furthermore, our experimentation with this type of modeling approach in both the full aircraft problem as well as the single aircraft problem (problem (1) restricted to a single aircraft; see problem (2) in Section 4.1) did not yield an appreciable change in solution times.

4. Method

As stated in Section 3, the formulation of the problem (problem (1)) in Section 3 is for the purpose of clearly defining the problem, as opposed to providing a path towards solving the problem. It turns out that problem (1) is too difficult to solve directly as a MIP; as we will see in Section 5.2, for small problems ($A = R = 10$), the problem can take as much as two hours to solve, and for slightly larger problems ($A = R = 20$), even the relaxation cannot be solved within a reasonable time frame.

Due to the difficulty of solving problem (1), we now focus our attention on an alternate approach for attacking the problem. This alternate approach is motivated by the fact that the problem consists of two coupled problems: we first must assign the requirements to the aircraft, and then we must sequence the events that each aircraft must perform. Solving both problems simultaneously within one formulation is very difficult, as acknowledged above. However, if one fixes an assignment of requirements to the aircraft, then the optimization problem simplifies considerably, for two reasons; each aircraft can be scheduled independently, leading to A independent optimization problems, and each optimization problem is a simpler one than the overall problem, since one major dimension of the problem – the assignment of requirements to aircraft – is eliminated.

In this section, we begin in Section 4.1 by describing a formulation for the *single* aircraft problem, where the set of requirements assigned to one aircraft is fixed, and one only has to sequence the events for that aircraft. Having defined the single aircraft problem, in Section 4.2 we describe an alternate, large-scale MIP reformulation of the full problem (1) from Section 3 that takes advantage of the coupled nature of the problem described above and the single aircraft problem defined in Section 4.1. We then develop in Section 4.3 a column generation approach for provably solving the LP relaxation of this problem. In Section 4.4, we additionally develop two heuristics – an initialization heuristic for finding an initial solution that is feasible and a local search heuristic for improving this initial solution – that can be used to warm start the column generation approach. Finally, in Section 4.5, we summarize the overall algorithmic approach.

4.1. Continuous-time MIP for single aircraft scheduling

We begin by describing our large-scale approach by defining the so-called single aircraft scheduling problem, which is the foundation for our overall algorithmic approach. In the single aircraft scheduling problem, a set of requirements is assigned to one aircraft, and we wish to schedule the pickup and dropoff events for that aircraft in a way that minimizes the objective from problem (1) (the weighted combination of delay and uptime) restricted to that one aircraft.

Let a be the aircraft of interest, and let $S \subseteq \mathcal{R}$ be the set of requirements that is to be scheduled. We define the notation for this problem below, which is similar to the notation of the full problem (1), except without the need for the aircraft indices in the variables and constraints since there is only a single aircraft involved.

Parameters. In addition to those parameters defined earlier in Section 3.1, we define some additional parameters. We let $\mathcal{E}(a,S)$ denote the set of events that must be performed by aircraft a when it is assigned the set of requirements S; formally, it is defined as:

$$\mathcal{E}(a,S) = \{\text{Start}(a), \text{End}(a)\} \cup \{\text{Pickup}(r) : r \in S\} \cup \{\text{Dropoff}(r) : r \in S\}.$$

Similarly to Section 3.2, we let P denote the number of slots in the formulation, which is the size of $\mathcal{E}(a,S)$; this is then simply $P = 2|S| + 2$.

Variables. The decision variables are defined as follows. For each event $e \in \mathcal{E}(a,S)$ and slot $p \in \{1,\dots,P\}$ we let $x_{e,p}$ be a binary decision variable that is 1 if event e is performed in slot p, and 0 otherwise; we also let $x^{by}_{e,p}$ be a binary decision variable that is 1 if event e is performed by slot p. For each e and $p \in \{1,\dots,P-1\}$, we let $y_{e,e',p}$ be a binary decision variable that is 1 if event e is performed in slot p, and event e' is performed in slot $p+1$. For each slot p, we let z_p be a binary decision variable that is 1 if the aircraft rests after performing the event of slot p, and 0 otherwise. We let t_p be the time at which the aircraft starts to execute the event in slot p. We let $s_{e,p}$ denote the slack of event e if it is assigned to slot p. Finally, we let ω_p denote the accumulated active time of the aircraft at the start of slot p from the beginning of the aircraft's mission.

With these definitions, we now define the single aircraft scheduling problem.

$$\underset{\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{t},\mathbf{s},\boldsymbol{\omega}}{\text{minimize}} \quad \sum_{e \in \mathcal{E}(a,S)} \sum_{p=1}^{P} \pi_e s_{e,p} + C(t_P - t_1) \tag{2a}$$

$$\text{subject to} \quad \sum_{p=1}^{P} x_{e,p} = 1, \quad \forall\, e \in \mathcal{E}(a,S), \tag{2b}$$

$$\sum_{e \in \mathcal{E}(a,S)} x_{e,p} = 1, \quad \forall\, p \in \{1,\dots,P\}, \tag{2c}$$

$$x_{\text{Start}(a),1} = 1, \tag{2d}$$

$$x_{\text{End}(a),P} = 1, \tag{2e}$$

$$x_{e,1} = x^{by}_{e,1}, \quad \forall\, e \in \mathcal{E}(a,S), \tag{2f}$$

$$x_{e,p} = x^{by}_{e,p} - x^{by}_{e,p-1}, \quad \forall\, e \in \mathcal{E}(a,S),\ p \in \{2,\dots,P\}, \tag{2g}$$

$$x^{by}_{e,p-1} \le x^{by}_{e,p}, \quad \forall\, p \in \{2,\dots,P\}, \tag{2h}$$

$$x^{by}_{\text{Dropoff}(r),p} \le x^{by}_{\text{Pickup}(r),p}, \quad \forall\, r \in S,\ p \in \{1,\dots,P\}, \tag{2i}$$

$$\sum_{e' \in \mathcal{E}(a,S)} y_{e,e',p} = x_{e,p}, \quad \forall\, e \in \mathcal{E}(a,S),\ p \in \{1,\dots,P-1\}, \tag{2j}$$

$$\sum_{e' \in \mathcal{E}(a,S)} y_{e',e,p-1} = x_{e,p}, \quad \forall\, e \in \mathcal{E}(a,S),\ p \in \{2,\dots,P\}, \tag{2k}$$

$$\sum_{e \in \mathcal{E}(a,S)} w_e x^{by}_{e,p} \le W_a, \quad \forall\, p \in \{1,\dots,P\}, \tag{2l}$$

$$t_p \ge t_{p-1} + \sum_{e,e' \in \mathcal{E}(a,S)} \tau_{e,e',a} \cdot y_{e,e',p-1} + \delta_a z_{p-1}, \quad \forall\, p \in \{2,\dots,P\}, \tag{2m}$$

$$t_p \ge \sum_{e \in \mathcal{E}(a,S)} \ell_e x_{e,p}, \quad \forall\, p \in \{1,\dots,P\}, \tag{2n}$$

$$t_p \le \sum_{e \in \mathcal{E}(a,S)} u_e x_{e,p} + \sum_{e \in \mathcal{E}(a,S)} s_{e,p}, \quad \forall\, p \in \{1,\dots,P\}, \tag{2o}$$

$$s_{e,p} \leq B_e \cdot x_{e,p}, \quad \forall\, e \in \mathcal{E}(a,S),\ p \in \{1,\ldots,P\}, \tag{2p}$$

$$\omega_1 = 0, \tag{2q}$$

$$\omega_p = \omega_{p-1} + \sum_{e,e' \in \mathcal{E}(a,S)} \tau_{e,e',a} \cdot y_{e',e,p-1}, \quad \forall\, p \in \{2,\ldots,P\}, \tag{2r}$$

$$\omega_{q'} - \omega_q \leq \gamma_a \cdot \left(1 + \sum_{p=q}^{q'-1} z_p\right), \quad \forall\, q \in \{1,\ldots,P-1\},\ q' \in \{q,\ldots,P\}, \tag{2s}$$

$$x^{by}_{e,p} \in \{0,1\}, \quad \forall\, e \in \mathcal{E}(a,S),\ p \in \{1,\ldots,P\}, \tag{2t}$$

$$x_{e,p} \geq 0, \quad \forall\, e \in \mathcal{E}(a,S),\ p \in \{1,\ldots,P\}, \tag{2u}$$

$$y_{e,e',p} \geq 0, \quad \forall\, e,e' \in \mathcal{E}(a,S),\ p \in \{1,\ldots,P\}, \tag{2v}$$

$$z_p \in \{0,1\}, \quad \forall\, p \in \{1,\ldots,P\}, \tag{2w}$$

$$s_{e,p} \geq 0, \quad \forall\, e \in \mathcal{E}(a,S),\ p \in \{1,\ldots,P\}, \tag{2x}$$

$$\omega_p \geq 0, \quad \forall\, p \in \{1,\ldots,P\}. \tag{2y}$$

Mission constraints: Constraint (2b) requires that each event in $\mathcal{E}(a,S)$ – all Pickup(r) and Dropoff(r) events as well as Start(a) and End(a) – are assigned to a slot. Constraint (2c) requires that each slot has exactly one event assigned to it. Constraints (2d) and (2e) ensure that Start(a) and End(a) are in the first and last slots, respectively.

"By"-"at" linking constraints: Constraints (2f) and (2g) link the $x^{by}_{e,p}$ variable ("event e happens *by* slot p) and the $x_{e,p}$ variables ("event e happens *at* slot p) together. Constraint (2h) links the x^{by} variables in consecutive periods; in words, if event e happens by slot $p-1$, then event e happens by slot p.

Pickup/dropoff constraint: Constraint (2i) requires that, in terms of slots, the Pickup(r) event comes before the Dropoff(r) event.

Transition constraints: Constraints (2j) and (2k) ensures that the y variables, which model transitions from one event in one slot to another event in the next slot, are consistent with the sequence of events as represented by the x variables.

Capacity constraint: Constraint (2l) ensures that the total weight carried in the aircraft at each slot is no more than the capacity of the aircraft.

Travel time dynamics constraint: Constraint (2m) models the one-step dynamics of the time of each event. In words, the time at which the event of slot p is started is at least the time of the event in slot $p-1$, plus the travel time from the event of slot $p-1$ to the event of slot p and the minimum rest period (if the aircraft rests after slot $p-1$).

Time window constraints: Constraints (2n) and (2o) require that the time at which the event in slot p is executed is within the time window of that event (no earlier than the earliest allowable time, ℓ_e, and no later than the latest allowable time u_e plus any slack captured in $s_{e,p}$). Constraint (2p) requires that the slack variable $s_{e,p}$ is at most B_e if event e happens in slot p, and is forced to zero otherwise.

Active time constraints: Constraint (2q) ensures that the accumulated active time before the first event (Start(a)) is exactly zero; constraint (2r) models the dynamics of how active time is accumulated from one slot to the next. Constraint (2s) ensures that the active time accumulated between any two periods does not exceed how much active time is afforded by the rests taken over those two periods.

Variable definitions: Constraints (2t) through (2y) specify that the x^{by} and z variables are binary, while the x, y, s, t and ω variables are continuous. Note that although the x ("at" variables) and y variables have a binary interpretation, they can be modeled as continuous variables since, by virtue of the constraints, the x and y variables will be automatically forced to their correct binary values whenever the x^{by} variables are binary.

Objective function: The objective (2a) is identical to that in the continuous-time formulation, only it is restricted to aircraft a; it is the priority-weighted sum of event slack times, plus the weighted total time from the start of the mission to the end of the mission for the given aircraft.

For a given aircraft a and a given set of requirements S, we use $f(a,S)$ to denote the optimal objective value of problem (2) when it is feasible. In the case that problem (2) is infeasible, we define $f(a,S) = +\infty$.

Problem (2) bears strong resemblance to problem (1); compare, for example, constraint (2m) to constraint (1n). Despite this resemblance, the two problems are different, with the key difference being that problem (2) pertains to a single aircraft, where the set of requirements assigned to it is known. As such, problem (2) is a considerably easier problem to solve than problem (1). Note also that since the assigned set of requirements is known in problem (2), a number of constraints of a more technical nature from problem (1) can be eliminated from problem (2) (for example, there is no analog of constraint (1f) because we know that End(a) will be in the last slot).

In addition to this conceptual difference between the goals of problems (2) and (1), there is also one other major difference between the two formulations, which has to do with the use of "by" variables (the $x^{by}_{e,p}$ variables) in addition to "at" variables (the $x_{e,p}$ variables). The reason for this choice is that it leads to more desirable branching behavior. More specifically, by branching on a fractional $x_{e,p}$ variable, the up branch ($x_{e,p} = 1$) contains a lot of information (event e happens in slot p), but the down branch ($x_{e,p} = 0$) does not (event e does not happen in slot p, so it could occur in any of the other slots). In contrast, by branching on an $x^{by}_{e,p}$ variable, the solution space is partitioned in a more balanced way; the up branch ($x^{by}_{e,p} = 1$) tells us that event e happens in a slot in $\{1,\ldots,p\}$, whereas the down branch ($x^{by}_{e,p} = 0$) tells us that event e happens in a slot in $\{p+1,\ldots,P\}$. The use of "by" variables has been considered in other scheduling applications (see, e.g., Bertsimas and Stock Patterson 1998, 2000, Bertsimas, Lulli, and Odoni 2011) and more generally in integer programming (Vielma 2015). Empirically, we have observed that the single aircraft problem can be solved faster when formulated using "by" variables as opposed to just the "at" variables. Unfortunately, our experimentation with using "by" variables within the full aircraft formulation (1) did not lead to an appreciable improvement in the solvability of that formulation, which is why problem (1) is formulated only in terms of the "at" variables.

4.2. Large-scale MIP formulation

In this section, we present a large-scale MIP formulation of the airlift planning problem. This formulation is termed "large-scale" because in general, it contains an extremely large number of variables, that scales exponentially with the number of requirements.

We begin by defining some additional notation. We let $\mathcal{P}(\mathcal{R})$ denote the power set of $\mathcal{R}$, that is, the collection of all subsets of $\mathcal{R}$. We let V denote the set of pairs $(a,S) \in \mathcal{A} \times \mathcal{P}(\mathcal{R})$ for which the single aircraft problem is feasible, i.e., $f(a,S) < +\infty$. We use $\mathcal{P}_a(\mathcal{R})$ to denote the collection of subsets of $\mathcal{R}$ for which the single aircraft problem with aircraft a is feasible, that is,

$$\mathcal{P}_a(\mathcal{R}) = \{S \in \mathcal{P}(\mathcal{R}) \mid (a,S) \in V\}.$$

The only decision variable in this new formulation is $x_{a,S}$, a binary decision variable that is 1 if the set of requirements S is assigned to aircraft a, and 0 otherwise.

The large-scale MIP formulation of the airlift planning problem can now be defined as follows.

$$\begin{aligned}
\underset{\mathbf{x}}{\text{minimize}} \quad & \sum_{(a,S)\in V} f(a,S)\cdot x_{a,S} && \text{(3a)}\\
\text{subject to} \quad & \sum_{(a,S)\in V} \mathbb{I}\{r\in S\}\cdot x_{a,S} = 1, \quad \forall\, r\in\mathcal{R}, && \text{(3b)}\\
& \sum_{(a,S)\in V} \mathbb{I}\{a=a'\}\cdot x_{a,S} = 1, \quad \forall\, a'\in\mathcal{A}, && \text{(3c)}\\
& x_{a,S}\in\{0,1\}, \quad \forall (a,S)\in V. && \text{(3d)}
\end{aligned}$$

To understand the formulation, let us first consider the constraints. Constraint (3b) requires that each requirement in the entire set of requirements $\mathcal{R}$ is assigned to one of the aircraft. Constraint (3c) requires that each aircraft is assigned to a set of requirements S; note that S could be the empty set $\emptyset$. Constraint (3d) requires that the $x_{a,S}$ variables be binary. Taken together, the constraints ensure that the $x_{a,S}$ variables represent a partitioning of the set of requirements $\mathcal{R}$ over the available set of aircraft $\mathcal{A}$. The objective (3a) represents the same objective as in problem (1), only expressed using the $f(a,S)$ parameters and the $x_{a,S}$ decision variables.

We note that the above problem is a set partitioning problem which, as mentioned in Section 2, are often used to model vehicle routing problems. We note that our problem involves an assignment decision (as in, e.g., Carnes et al. 2013), as opposed to being pure partitioning problem (as in, e.g., Dumas, Desrosiers, and Soumis 1991). This is necessary because the aircraft are not homogeneous. In particular, the aircraft may be required to start and end their missions at different ports and they may have different weight capacities and different nominal air speeds. Thus, assigning the same set of requirements to two different aircraft can lead to significant differences in the quality of the overall solution, which needs to be accounted for in the above formulation.

4.3. Column generation approach

Problem (3) is, like problem (1), still an extremely challenging problem to solve. There are three reasons for this. First, although the problem has a tractable number of constraints (it contains $R+A$ linear constraints), the number of variables will in general be extremely large, as the variables correspond to subsets of $\mathcal{R}$. Second, the $f(a,S)$ values that are used to define the optimization problem are not available to us a priori; each such value must be obtained by solving an integer programming problem (namely, problem (2)). Finally, the problem is still an integer programming problem. However, the advantage of considering problem (3) is that it is amenable to column generation methods.

In this section, we will consider a column generation approach for approximately solving problem (3). We begin by considering the linear programming (LP) relaxation of problem (3), which is given below:

$$\begin{aligned}
\underset{\mathbf{x}}{\text{minimize}} \quad & \sum_{(a,S)\in V} f(a,S)\cdot x_{a,S} && \text{(4a)}\\
\text{subject to} \quad & \sum_{(a,S)\in V} \mathbb{I}\{r\in S\}\cdot x_{a,S} = 1, \quad \forall\, r\in\mathcal{R}, && \text{(4b)}\\
& \sum_{(a,S)\in V} \mathbb{I}\{a=a'\}\cdot x_{a,S} = 1, \quad \forall\, a'\in\mathcal{A}, && \text{(4c)}\\
& x_{a,S}\geq 0, \quad \forall\, (a,S)\in V. && \text{(4d)}
\end{aligned}$$

We will develop our column generation approach from a dual perspective. The dual of this relaxation is given by

$$\underset{\boldsymbol{\alpha},\boldsymbol{\beta}}{\text{maximize}} \quad \sum_{r\in\mathcal{R}} \alpha_r + \sum_{a\in\mathcal{A}} \beta_a \tag{5a}$$

$$\text{subject to} \quad \sum_{r\in S} \alpha_r + \beta_a \le f(a,S), \quad \forall\ (a,S)\in V. \tag{5b}$$

Clearly, by solving problem (5), we also solve problem (4). To solve problem (5), let us assume that rather than starting with all of the constraints enumerated over the set V, which is the set of all possible (a,S) values, we instead assume that we start with a subset $\bar{V}$ of all possible (a,S) pairs. We define the dual restricted master problem to be problem (6) with the constraints restricted to the (a,S) pairs in $\bar{V}$; formally, it is:

$$\underset{\boldsymbol{\alpha},\boldsymbol{\beta}}{\text{maximize}} \quad \sum_{r\in\mathcal{R}} \alpha_r + \sum_{a\in\mathcal{A}} \beta_a \tag{6a}$$

$$\text{subject to} \quad \sum_{r\in S} \alpha_r + \beta_a \le f(a,S), \quad \forall\ (a,S)\in \bar{V}. \tag{6b}$$

Since $\bar{V}\subseteq V$, it is clear that the objective value of problem (6) is an upper bound on problem (5), and that an optimal solution to problem (6) is not necessarily feasible for problem (5).

To solve problem (5) using the dual restricted master problem (6), we will use constraint generation, which we now describe at a high level. Let $(\boldsymbol{\alpha},\boldsymbol{\beta})$ be an optimal solution to the dual restricted master problem (6). To check whether $(\boldsymbol{\alpha},\boldsymbol{\beta})$ is an optimal solution to the dual master problem (5), we must verify that constraint (6b) holds for all (a,S) pairs in the set V. If, for all $(a,S)\in V$, we have that $\sum_{r\in S}\alpha_r+\beta_a-f(a,S)\le 0$, then $(\boldsymbol{\alpha},\boldsymbol{\beta})$ is feasible for problem (5), and therefore an optimal solution of problem (5). Otherwise, if we find a (a,S) pair such that $\sum_{r\in S}\alpha_r+\beta_a-f(a,S)>0$, then $(\boldsymbol{\alpha},\boldsymbol{\beta})$ is not feasible for problem (5), because the corresponding (a,S) constraint in problem (5) is violated; having found such a (a,S) pair, we can then add it to the set $\bar{V}$ and solve the problem again, yielding a new solution $(\boldsymbol{\alpha},\boldsymbol{\beta})$. We then repeat the procedure until we find a feasible solution. We note that generating constraints in the dual problem is equivalent to generating columns in the primal problem (4).

In our implementation of this scheme, we equivalently replace constraint (6b), which ranges over all aircraft, with A constraints that correspond to each individual aircraft, as follows:

$$\begin{aligned}
&\sum_{r\in S}\alpha_r+\beta_1\le f(1,S), \quad \forall\ S\in\mathcal{P}_1(\mathcal{R}),\\
&\qquad\vdots\\
&\sum_{r\in S}\alpha_r+\beta_A\le f(A,S), \quad \forall\ S\in\mathcal{P}_A(\mathcal{R}).
\end{aligned}$$

Given a candidate solution $(\boldsymbol{\alpha},\boldsymbol{\beta})$, we then check each family of constraints to determine if there are any violated constraints, and add all such violated constraints. Algorithm 1 describes the full constraint generation scheme.

We now comment on two important aspects of this method. First, Algorithm 1 is an algorithm for solving the LP relaxation (4) and does not provide an integer feasible solution (a solution to problem (3)). To produce an integer solution, we can solve problem (3) with the $x_{a,S}$ variables restricted to those (a,S) pairs generated during the constraint generation algorithm, i.e., those in $\bar{V}$. The integer solution that is produced in this way is *not* guaranteed to be an optimal solution for problem (3). In practice, however, the objective value of this solution is often close to that

Algorithm 1 Constraint generation algorithm for solving dual problem (5).

```
Require: Initial set of $(a,S)$ pairs $\bar{V}$.
  Solve problem (6) with $\bar{V}$, to obtain dual solution $(\boldsymbol{\alpha},\boldsymbol{\beta})$.
  For each $a \in \mathcal{A}$, compute:
    $\bar{c}_a = \max_{S \in \mathcal{P}_a(\mathcal{R})} (\sum_{r \in S} \alpha_r + \beta_a - f(a,S))$,
    $S_a = \arg\max_{S \in \mathcal{P}_a(\mathcal{R})} (\sum_{r \in S} \alpha_r + \beta_a - f(a,S))$.
  while $\max_{a \in \mathcal{A}} \bar{c}_a > 0$ do
    Set $\bar{V} \leftarrow \bar{V} \cup \{(a,S_a) \mid a \in \mathcal{A}, \bar{c}_a > 0\}$.
    Solve problem (6) with $\bar{V}$, to obtain dual solution $(\boldsymbol{\alpha},\boldsymbol{\beta})$.
    For each $a \in \mathcal{A}$, compute:
      $\bar{c}_a = \max_{S \in \mathcal{P}_a(\mathcal{R})} (\sum_{r \in S} \alpha_r + \beta_a - f(a,S))$,
      $S_a = \arg\max_{S \in \mathcal{P}_a(\mathcal{R})} (\sum_{r \in S} \alpha_r + \beta_a - f(a,S))$.
  end while
  return Optimal value of problem (5) with final $\bar{V}$ set.
```

of problem (4); since the optimal value of problem (4) is a lower bound on the optimal value of problem (3), this indicates that the integer solution is near-optimal.

Second, the key step in Algorithm 1 is finding the violated constraint, that is, finding the set S maximizing $\sum_{r \in S} \alpha_r + \beta_a - f(a,S)$ for each $a \in \mathcal{A}$. The problem of finding such a set S is a difficult optimization problem, and solving it exactly is extremely computationally intensive. Due to its difficulty, we opt to solve this problem in an approximate manner, rather than in an exact manner. The approximate approach that we propose is a local search procedure that starts at a set of requirements S_0, and then iteratively searches through neighboring sets – obtained by either deleting a requirement in S or adding a requirement not in S to S – to find sets that lead to an improved value of the constraint violation $\sum_{r \in S} \alpha_r + \beta_a - f(a,S)$. Letting $\phi(a,S,\boldsymbol{\alpha},\boldsymbol{\beta})$ denote the constraint violation, that is,

$$\phi(a,S,\boldsymbol{\alpha},\boldsymbol{\beta}) = \sum_{r \in S} \alpha_r + \beta_a - f(a,S),$$

we provide the pseudocode for the local search procedure as Algorithm 2.

Note that Algorithm 2 does not provably solve the separation problem $\max_S \phi(a,S,\boldsymbol{\alpha},\boldsymbol{\beta})$; it only finds a locally optimal solution. More precisely, if it terminates with an S such that $\phi(a,S,\boldsymbol{\alpha},\boldsymbol{\beta}) > 0$, then we have successfully identified a violated constraint; however, if it terminates with an S such that $\phi(a,S,\boldsymbol{\alpha},\boldsymbol{\beta}) \leq 0$, the algorithm does not guarantee the non-existence of an S for which $\phi(a,S,\boldsymbol{\alpha},\boldsymbol{\beta}) > 0$. The danger, therefore, is that by using this approximate solution approach, we might declare the current dual solution $(\boldsymbol{\alpha},\boldsymbol{\beta})$ to be an optimal solution when in fact it is not. One way of addressing this issue is to run Algorithm 2 not from one, but from multiple randomly generated starting sets S_0. In this way, if $(\boldsymbol{\alpha},\boldsymbol{\beta})$ is not optimal, then we increase the likelihood of identifying an S for which $\phi(a,S,\boldsymbol{\alpha},\boldsymbol{\beta}) > 0$.

4.4. Heuristics

The constraint generation method presented as Algorithm 1 starts from a user-specified set of (a,S) pairs denoted by $\bar{V}$. The set $\bar{V}$ can be chosen based on a known solution to the problem (3). More precisely, suppose that we know a partitioning of the requirements $\mathcal{R} = S_1 \cup \cdots \cup S_A$, where S_a is the set of requirements assigned to aircraft a; we can then use this solution to warm start Algorithm 1 by setting $\bar{V}$ as

$$\bar{V} = \{(1,S_1)\} \cup \cdots \cup \{(A,S_A)\}.$$

By the definition of the dual restricted master problem (6), it is easy to see that the objective value corresponding to $\bar{V}$ will be exactly $\sum_{a \in \mathcal{A}} f(a,S_a)$. By inputting a solution $S_1,\ldots,S_A$ that achieves

Algorithm 2 Local search procedure for solving dual separation problem.

```
Require: Aircraft $a$, initial requirement set $S_0$; dual solution $(\boldsymbol{\alpha},\boldsymbol{\beta})$.
  Set $S \leftarrow S_0$.
  Compute $\phi_C = \phi(a, S, \boldsymbol{\alpha}, \boldsymbol{\beta})$.
  Compute:
    $V_A = \{S' \in \mathcal{P}(\mathcal{R}) \mid S' = S \cup \{r\}$ for some $r \notin S\}$
    $V_D = \{S' \in \mathcal{P}(\mathcal{R}) \mid S' = S \setminus \{r\}$ for some $r \in S\}$
  Compute:
    $\tilde{\phi} = \max_{S' \in V_A \cup V_D} \phi(a, S', \boldsymbol{\alpha}, \boldsymbol{\beta})$,
    $\tilde{S} = \arg\max_{S' \in V_A \cup V_D} \phi(a, S', \boldsymbol{\alpha}, \boldsymbol{\beta})$.
  while $\tilde{\phi} > \phi_C$ do
    Set $\phi_C \leftarrow \tilde{\phi}$.
    Set $S \leftarrow \tilde{S}$.
    Compute:
      $V_A = \{S' \in \mathcal{P}(\mathcal{R}) \mid S' = S \cup \{r\}$ for some $r \notin S\}$
      $V_D = \{S' \in \mathcal{P}(\mathcal{R}) \mid S' = S \setminus \{r\}$ for some $r \in S\}$
    Compute:
      $\tilde{\phi} = \max_{S' \in V_A \cup V_D} \phi(a, S', \boldsymbol{\alpha}, \boldsymbol{\beta})$,
      $\tilde{S} = \arg\max_{S' \in V_A \cup V_D} \phi(a, S', \boldsymbol{\alpha}, \boldsymbol{\beta})$.
  end while
  return Locally optimal solution $S$, objective value $\phi_C$.
```

a good objective value, we can potentially reduce the number of iterations required by the column generation algorithm.

In this section, we present two heuristics for constructing such a solution. The first heuristic that we consider is an initialization heuristic for constructing an initial feasible solution. The second heuristic is a local search heuristic that can be used to improve the solution generated by the initialization heuristic.

We begin by describing the initialization heuristic. We start by assuming that all of the requirements are unassigned and setting the requirement set S_a of each aircraft to be the empty set $\emptyset$. In each iteration, we select an unassigned requirement r, and attempt to add it to each of the aircraft. For each aircraft a, adding r to that aircraft will either result in infeasibility (i.e., the single aircraft problem for aircraft a with the requirements in $S_a \cup \{r\}$ is infeasible) or in a feasible schedule. Among those aircraft a for which r is feasible, we assign r to the aircraft a which results in the smallest change to the objective value of the whole solution. We then repeat the process for the remaining requirements, until all of the requirements have been assigned. Algorithm 3 provides a pseudocode description of the algorithm.

There are three aspects of Algorithm 3 that are worth noting. The first is that the behavior of Algorithm 3 is highly dependent on the order in which we proceed through the requirements. In our implementation of Algorithm 3 in our numerical experiments in Section 5, we randomly order the requirements. The second aspect, which relates to the first, is that Algorithm 3 is not guaranteed to result in a feasible solution. In the case that a requirement cannot be feasibly assigned to any of the aircraft, we re-start Algorithm 3 with a new random ordering; in our numerical experiments in Section 5, a feasible solution could be found in almost all cases on the first try. Finally, due to the randomization of the requirements, running Algorithm 3 multiple times can produce different solutions, some of which may be substantially better than others. In our implementation of this algorithm in Section 5, we run Algorithm 3 ten times to produce ten initial solutions, from which we retain the one with the best objective.

We now describe the second heuristic, which is a local search heuristic. We start with a partition $S_1, \dots, S_A$ of the requirements, where S_a represents the requirements assigned to aircraft a. In

Algorithm 3 Initialization heuristic for constructing initial feasible solution.

```
  Initialize $S_1 \leftarrow \emptyset, \dots, S_A \leftarrow \emptyset$.
  Initialize unassigned requirement set $\mathcal{R}_{\text{unassigned}} \leftarrow \mathcal{R}$.
  Compute objective value $Z \leftarrow \sum_{a \in \mathcal{A}} f(a, S_a)$.
  while $|\mathcal{R}_{\text{unassigned}}| > 0$ do
    Select $r$ from $\mathcal{R}_{\text{unassigned}}$.
    For each $a \in \mathcal{A}$, compute $Z_a \leftarrow f(a, S_a \cup \{r\}) + \sum_{a' \in \mathcal{A} \setminus \{a\}} f(a', S_{a'})$.
    Compute $k^* \leftarrow \arg\min_{a : Z_a < +\infty} Z_a$.
    Update $S_{a^*} \leftarrow S_{a^*} \cup \{r\}$.
    Update $Z \leftarrow Z_{a^*}$.
    Update $\mathcal{R}_{\text{unassigned}} \leftarrow \mathcal{R}_{\text{unassigned}} \setminus \{r\}$.
  end while
  return Partition $S_1, \dots, S_A$, objective value $Z$.
```

each iteration, we select one of the requirements, and consider the new partition that results from removing that requirement from its current aircraft, and assigning it to one of the other aircraft; there are $A - 1$ such possible new partitions. We accept the partition that most improves on the current partition, if at least one such partition exists; otherwise, we repeat this process for each of the other requirements we have not yet considered. We terminate if all such neighboring partitions of the current partition, obtained by moving a single requirement to a different aircraft, are unable to yield an improvement over the current partition. Algorithm 4 provides a pseudocode description of this local search heuristic.

Algorithm 4 Local search heuristic for finding an improved solution.

```
Require: Initial partition $S_1, \dots, S_A$.
  Compute $Z \leftarrow \sum_{a \in \mathcal{A}} f(a, S_a)$.
  Initialize $\mathcal{R}_{\text{untested}} \leftarrow \mathcal{R}$.
  while $|\mathcal{R}_{\text{untested}}| > 0$ do
    Select $r$ from $\mathcal{R}_{\text{untested}}$.
    Set $a_r$ to be aircraft of $r$ (i.e., $a_r = a$ such that $r \in S_a$).
    For each $a \in \mathcal{A} \setminus \{a_r\}$, compute
      $Z_a \leftarrow f(a, S_a \cup \{r\}) + f(a_r, S_{a_r} \setminus \{r\}) + \sum_{a' \in \mathcal{A} \setminus \{a, a_r\}} f(a', S_{a'})$.
    if $\min_{a \in \mathcal{A}} Z_a < Z$ then
      Set $a^* \leftarrow \arg\min_{a \in \mathcal{A}} Z_a$.
      Update $S_{a^*} \leftarrow S_{a^*} \cup \{r\}$.
      Update $S_{a_r} \leftarrow S_{a_r} \setminus \{r\}$.
      Update $Z \leftarrow \min_{a \in \mathcal{A}} Z_a$.
      Update $\mathcal{R}_{\text{untested}} \leftarrow \mathcal{R} \setminus \{r\}$.
    else
      Set $\mathcal{R}_{\text{untested}} \leftarrow \mathcal{R}_{\text{untested}} \setminus \{r\}$.
    end if
  end while
  return Locally optimal partition $S_1, \dots, S_A$, objective value $Z$.
```

4.5. Overall algorithmic approach

Our overall algorithmic approach for solving problem (3) is as follows:

1. **Initialization**: Execute Algorithm 3 (the initialization heuristic) to obtain an initial feasible partition $S_1, \dots, S_A$.

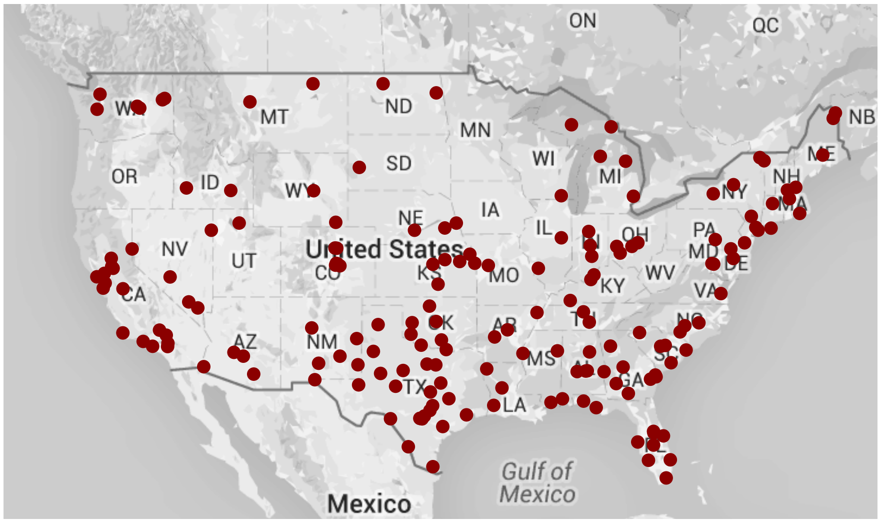

Figure 1 **Plot of geographic locations of ports in the continental United States.**

2. **Local search**: Starting from the initial partition $S_1, \dots, S_A$, execute Algorithm 4 (the local search heuristic) to obtain an improved partition $S'_1, \dots, S'_A$.
3. **Column/constraint generation**: Set $\bar{V}$ as

$$\bar{V} = \{(1, S'_1)\} \cup \cdots \cup \{(A, S'_A)\}$$

and execute Algorithm 1 (the constraint generation procedure) with this initial set $\bar{V}$ and using the approximate subproblem heuristic given by Algorithm 2.

4. **Final integer solution**: Using the final set $\bar{V}$ of (a, S) pairs generated in Step 3, solve the integer restricted master problem (i.e., problem (3) restricted to $\bar{V}$ instead of V) to obtain an integer solution $\mathbf{x}$. The final partition is given by $S''_1, \dots, S''_A$, where S''_a is the (unique) set for which $x_{a,S''_a} = 1$ in the solution $\mathbf{x}$.

5. Results

5.1. Background

In this section, we begin by describing the problem instances that we use in our numerical experiments, the hyperparameters and implementation details of our algorithmic approach and the baseline/status quo method that we will use to benchmark our algorithmic approach.

Data preprocessing. We first constructed a "master" set of ports. We assumed that this set consists of all possible United States Air Force (USAF) bases in the continental United States, leading to a set of 174 different airbases. For the aircraft, we assume that aircraft can be one of eight different cargo aircraft types used by the USAF. Figure 1 shows the distribution of ports in the continental United States, while Table 1 displays the eight different aircraft types, along with nominal values of their capacities and speeds. In the instances that we will shortly describe, aircraft travel times between pairs of events (the $\tau_{e,e',a}$ values) were calculated by computing the distance between the latitude-longitude pairs using the haversine formula and converting this distance into a time via the nominal speed of the aircraft.

Problem instances. We consider two different sets of instances, described below:

1. **T instances**. In these instances, the data is generated as follows:

- *Ports*: Ten unique ports are selected from the master list of 174 ports.

Table 1 **Cargo aircraft types considered in numerical experiments.**

Aircraft Type	Aircraft Name	Speed (miles/hour)	Capacity (short tons)
1	C-5 Galaxy	541	145
2	C-12 Huron	334	7.5
3	C-17 Globemaster	500	85
4	C-23 Metroliner	218	2.5
5	C-27 Spartan	362	12.5
6	C-40	615	20
7	C-130 Hercules	374	22.5
8	C-130 Super Hercules	400	22

- *Requirements*: Each requirement is assumed to have a weight of 12 short tons. The port of each pickup and dropoff event is uniformly randomly selected from the ten ports. For each requirement, the pickup and dropoff time windows are assumed to be equal (i.e., $\ell_{\text{Pickup}(r)} = \ell_{\text{Dropoff}(r)}$, $u_{\text{Pickup}(r)} = u_{\text{Dropoff}(r)}$, $B_{\text{Pickup}(r)} = B_{\text{Dropoff}(r)}$). For each requirement r, we set $\ell_{\text{Pickup}(r)} = 24 \times q$, where q is chosen uniformly randomly from $\{2, \ldots, 18\}$; we set $u_{\text{Pickup}(r)} = \ell_{\text{Pickup}(r)} + 12$, and we set $B_{\text{Pickup}(r)} = 12$.
- *Aircraft*: Each of the A aircraft is uniformly randomly selected to be one of aircraft types 7 and 8 (C-130 Hercules and C-130 Super Hercules, respectively). We set $\ell_{\text{Start}(a)} = \ell_{\text{End}(a)} = 0$, $u_{\text{Start}(a)} = u_{\text{End}(a)} = 24 * 20$ and $B_{\text{Start}(a)} = B_{\text{End}(a)} = 0$; in words, the aircraft can be used over a time horizon of twenty days, and must complete their missions by the end of the 20th day. The port of each Start(a) and End(a) event is uniformly randomly selected from the ten unique ports.

We consider values of $A \in \{10, 20, 50, 100\}$. We fix $R = A$. We consider twenty randomly generated **T** instances for $A \in \{10, 20, 50\}$; for $A = 100$, we only consider ten instances, due to the computationally demanding nature of such instances.

2. **R instances**. In addition to the T instances, we consider a set of instances derived from a set of requirements provided to us by USTRANSCOM. This set of requirements is used by USTRANSCOM analysts to internally benchmark their scheduling policies and is representative of a typical planning problem faced by USTRANSCOM in practice. The data set contains 111 requirements and includes partial time window information ($\ell_{\text{Pickup}(r)}$, $\ell_{\text{Dropoff}(r)}$, $u_{\text{Pickup}(r)}$ and $u_{\text{Dropoff}(r)}$ values) and weight information ($w_{\text{Pickup}(r)}$ values) for each requirement. With regard to the remaining (missing) data, we proceed as follows:

- *Ports*: The pickup and dropoff ports are indicated by a port ID number; due to the sensitive nature of the data, the precise port that each port ID value corresponds to was not provided to us. We observed twenty unique port ID values; to fill in the port information, we randomly select twenty unique ports from the master list of 174 ports, and assigned each port ID to one of the twenty chosen ports.
- *Requirements*: The maximum slack value B_e of each pickup and dropoff event was not provided to us. We thus assumed a maximum slack of 1 day for each pickup and dropoff event, i.e., $B_{\text{Pickup}(r)} = B_{\text{Dropoff}(r)} = 24$. With the slacks defined in this way, these instances involve a time horizon of 44 days.
- *Aircraft*: The data set did not provide the set of aircraft that were available at the time to plan these instances. Thus, we considered A aircraft, where the type of each aircraft is uniformly randomly selected from all eight aircraft types in Table 1, and the start and end port of each aircraft was uniformly randomly selected from the twenty unique ports.

We consider instances with $A = 100$ aircraft and all of the requirements ($R = 111$). We randomly generate ten instances using the procedure outlined above.

We also consider a set of smaller scale instances derived from this data. To obtain these instances, we first randomly select R requirements without replacement from the complete set of 111 requirements, and then fill in all missing data as described above. For these instances, we set $A = R$, and let R range in $\{10, 20, 50\}$. For each value of R, we randomly generate twenty instances.

In all of the above instances, the priorities π_e of all events (used to form the objective function in problems (1) and (2)) are set to 1, and the weight C of the uptime is set to 1.

Software/Hardware. All linear and mixed-integer programming problems are solved using Gurobi 6.5 (Gurobi Optimization, Inc. 2016) and all algorithms are implemented in Julia (Bezanson et al. 2017) using the Julia for Mathematical Programming (JuMP) library (Lubin and Dunning 2015, Dunning, Huchette, and Lubin 2017). All experiments were conducted on a server with a 12-core 3.33GHz Intel Xeon X5680 processor and 96GB RAM.

Baseline method. To understand the relative merits of our algorithmic approach, we will compare our algorithmic approach against a baseline method. This baseline method is intended to roughly represent the current approach that USTRANSCOM uses to plan airlift operations. As discussed in Section 1, USTRANSCOM does not have a globally optimized approach for scheduling airlift operations. However, it has been observed that planners typically schedule a set of requirements one by one, in the order in which the requirements are entered into the system; this order is typically correlated with the earliest pickup time of each requirement.

For our baseline heuristic, we will therefore consider a modified version of Algorithm 3 where the order in which the requirements are assigned to the aircraft corresponds to the order of their earliest pickup times, that is, their $\ell_{\text{Pickup}(r)}$ values. Although this baseline heuristic is more sophisticated than current USTRANSCOM practice due to the use of MIP in the single aircraft scheduling component of Algorithm 3, it is a reasonable representation of how USTRANSCOM would currently schedule a set of requirements.

Our approach (Section 4.5). Unless stated otherwise, our overall approach is implemented as follows. We run the initialization phase (Step 1 of our approach in Section 4.5; this is Algorithm 3) ten times. From each of the ten initial solutions produced by Algorithm 3, we execute the local search phase (Step 2 of our approach in Section 4.5; this is Algorithm 4). Out of the ten locally optimized points, we use the best one to run the column/constraint generation algorithm (Step 3 of our approach of Section 4.5; this is Algorithm 1). For ease of exposition, we will refer to the initialization procedure (Algorithm 3) as H1; we will refer to the local search procedure (Algorithm 4) as H2; and we will refer to the column/constraint generation procedure (Algorithm 1) as CG.

In the execution of all three phases – H1, H2 and CG – we solve the single aircraft problem (2) to a time limit of two seconds; when reporting the objective value of the final solution of each phase, we compute it by solving the single aircraft problem (2) without any time limit. We apply the same time limits for the baseline method.

With regard to H1, we randomize the order in which we progress through the requirements. Also, as noted in Section 4.4, it is possible that H1 may not find a feasible solution on its first run. To address this possibility, we implemented H1 to be re-executed (with a new random ordering of the requirements) up to four times, for up to five runs in total.

With regard to CG, we use the approximate subproblem heuristic described as Algorithm 2. At each iteration of CG, we repeat Algorithm 2 up to ten times to find a violated constraint; if a violated constraint is not found after ten iterations, the algorithm is terminated. The initial starting set S_0 of Algorithm 2 is randomly generated, with each requirement r being included in the set S_0 independently with probability $2/R$. This choice was made based on preliminary experimentation with the T instances showing that the number of requirements assigned to each aircraft is small; by applying our specific form of random generation, we bias the initial set of requirements to a set that is small (the expected number of requirements in the set is $R \cdot (2/R) = 2$). In addition, when $|S_0|$ is very large, the single aircraft problem (2) requires more time to solve, which hinders the rate at which Algorithm 2 can locally optimize the set of requirements S. With regard to time limits, we terminate each run of Algorithm 2 after 30 seconds of execution, and we terminate the overall CG procedure after twelve hours of execution.

Table 2 Results for full MIP problem (1)**, for** $A = R = 10$ **and** $A = R = 20$.

A	R	Avg. Time (s)	Max. Time (s)	Gap (%)
10	10	1590.2	8917.6	0.00
20	20	21605.2	21607.3	–

The gap is omitted for the $A = R = 20$ instances, as the relaxation could not be solved for any of those instances within the three hour time limit.

5.2. Necessity of large-scale approach

The purpose of our first experiment is to establish the necessity of the overall approach that we summarize in Section 4.5. In this experiment, we attempt to solve the T instances described in Section 5.1 by directly solving the full MIP problem (1), for $A = R = 10$ and $A = R = 20$. We terminate the MIP after three hours of computation.

The results are shown in Table 2 for each of the T instances with $A = R = 10$ and $A = R = 20$. From this table, we can see that although the $A = R = 10$ instances are solved relatively quickly to full optimality, the $A = R = 20$ instances are not; in fact, in all of the $A = R = 20$ instances, the relaxation could not be solved within the three hour time limit. Moreover, even for the $A = R = 10$ instances, while the average time is on the order of 20 minutes, some instances can take significantly longer, with one instance taking almost 2.5 hours. The main takeaway from this table is that solving the full MIP formulation of the airlift planning problem is not a feasible option for realistic, large-scale problem instances.

5.3. Simulated data experiment

In this section, we compare the performance of our algorithmic approach in Section 4.5 against the baseline approach (see Section 5.1). We consider the T instances, which consist of simulated data. We consider the twenty instances for $A \in \{10, 20, 50\}$ and the ten instances for $A = 100$.

Table 3 shows results pertaining to the average objective value achieved by the different methods. As mentioned in Section 5.1, the value of C in the objective function of the single aircraft problem is set to 1, so that the objective represents the sum of the total delays of all events and the sum of the uptimes of all of the aircraft. The column "Avg. Improv. Baseline → CG" indicates the average relative reduction (over the instances for each (A, R) combination) in the objective value when comparing the CG solution relative to the baseline solution; the column "Max. Improv. Baseline → CG" indicates the best relative reduction over the instances for each (A, R) combination. The column "CG Gap" indicates the approximate optimality gap of the CG solution, which is defined as $100\% \times (Z^*_{IP,CG} - Z^*_{LP,CG})/Z^*_{LP,CG}$, where $Z^*_{IP,CG}$ is the objective value of the final integer solution produced by CG and $Z^*_{LP,CG}$ is the objective value of the LP relaxation (the value of problem (6) when CG terminates with the approximate separation problem heuristic, Algorithm 2). This value is averaged over the instances for each (A, R) combination.

From this table, we can see that the average objective of H1 is lower than that of the baseline method, that H2 improves upon the H1 solutions, and that CG further improves upon the best H2 solution. We can also see that the improvement generated by our overall algorithmic approach over the baseline method is substantial. By running H1, H2 and CG together, the objective value can be reduced by about 8 to 12% on average relative to the baseline method; in the best case, the reduction in the objective value can be as large as 20%. In addition, we can consider the approximate gap of the ultimate solution produced by CG. Although the bound Z_{LP} is an approximate bound and is not the provable objective value of problem (4), we can see that the approximate gap is fairly low – about 1.5% or lower.

Table 3 Objective values (hours of total delay and uptime) and related metrics of baseline, H1, H2 and CG methods for T instances.

A	R	Baseline Obj. (hrs)	H1 Obj. (hrs)	H2 Obj. (hrs)	CG Obj. (hrs)	CG Bound (hrs)	Avg. Improv. Baseline → CG (%)	Max. Improv. Baseline → CG (%)	Avg. CG Gap (%)
10	10	42.6	39.7	39.7	38.9	38.9	8.4	19.7	0.00
20	20	96.6	89.7	88.4	85.0	84.9	11.8	18.8	0.10
50	50	196.8	188.4	182.8	173.4	172.5	11.8	17.6	0.54
100	100	277.4	270.1	258.0	247.9	244.6	10.6	15.1	1.32

Table 4 Computation times of baseline, H1, H2 and CG methods for T instances.

A	R	Baseline Time (s)	H1 Time (s)	H2 Time (s)	CG Time (s)	H1 + H2 Time (s)	H1+H2+CG Time (s)
10	10	0.4	4.0	5.7	23.7	9.7	33.4
20	20	2.9	22.9	48.3	333.3	71.2	404.5
50	50	29.7	338.7	1053.0	10671.4	1391.7	12063.1
100	100	382.7	4239.9	16684.9	44536.4	20924.8	65461.2

Aside from the objective value, it is also important to compare the methods in their computation time. Table 4 shows the average computation time required by the different methods over the T instances, as well as the total time of our algorithmic approach (Section 4.5). We can see that the baseline method is the fastest method, followed by H1, H2 and finally CG. Note that our implementation of H1 involves running H1 ten different times, and selecting the best solution; therefore, we should expect that the total time for the H1 phase of our overall approach should be about ten times larger than that of baseline. The H2 phase in general requires more computation time than the H1 phase. However, even in the largest set of instances ($A = R = 100$), the H1 phase (with ten repetitions) requires just over one hour on average; similarly, the H2 phase (with the ten starting points produced by H1) requires just under five hours. Note from Table 3 that the solutions produced by H1 and H2 already provide a substantial improvement over the baseline method; thus, with only six hours of combined time, H1 and H2 can be used to produce significantly better solutions than the baseline method. With regard to the CG method, we can see that CG in general requires the most amount of time. For $A = 10$ and $A = 20$, the method completes within five minutes; for $A = 50$, it requires about three hours; and for $A = 100$, the twelve hour time limit is exhausted in all of the corresponding instances. These results indicate that if we are satisfied with the heuristic solution generated by H1 and H2, we can stop there; otherwise, with a modest amount of additional computation time, CG can further improve these solutions and provide us an approximate sense of how optimal they are.

With regard to the column generation phase of the method, it is also interesting to examine the quality of the solution with each iteration of column generation. Figure 2 shows the evolution of the master problem objective with each iteration of CG for one representative T instances with $A = R = 50$.. The line labeled "LP" shows the optimal objective of the relaxed master problem (problem (4)) restricted to the set of columns that have been cumulatively generated at each iteration. The line labeled "MIP" shows the optimal objective of the *integer* master problem (problem (3)) restricted to the same set of columns at each iteration. Note that the "LP" line is always below the "MIP" line as the former corresponds to the relaxed problem (4). We can see two behaviors from this plot. First, the LP objective is generally steadily decreasing with each iteration. Second, the MIP objective generally follows the LP objective closely, but does not change as regularly as the LP objective changes; this is to be expected, because not every column generated for the LP relaxation can be used to form a bona fide integer solution.

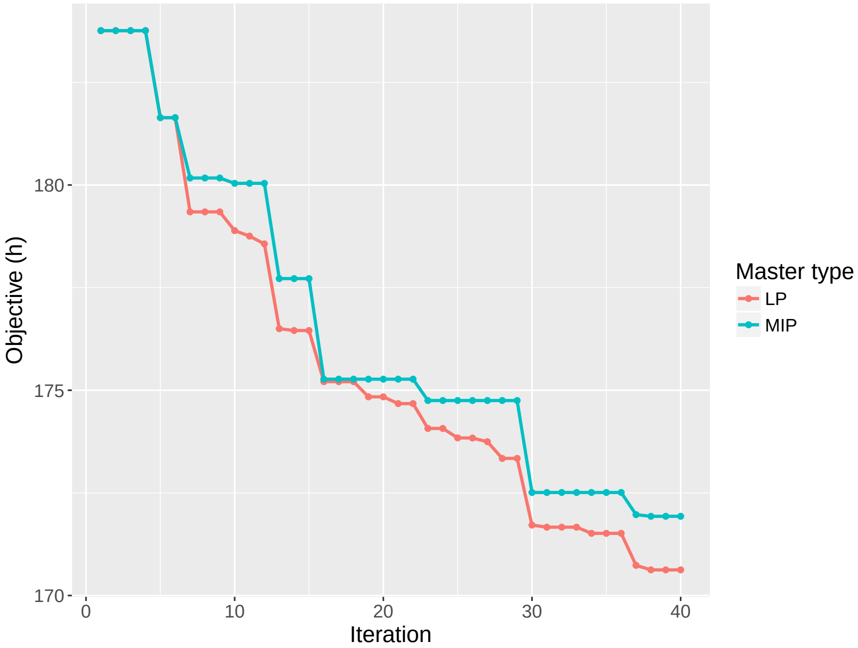

Figure 2 **Example of the evolution of the objective value of the relaxed master problem (line labeled "LP") and the integer master problem (line labeled "MIP") with each CG iteration, shown for a single T instance with $A = R = 50$.**

Finally, it is interesting to compare the solutions in terms of their qualitative behavior – namely, what is the delay, uptime and total number of aircraft used by the solutions. Tables 5, 6 and 7 show the average delay, uptime and total number of aircraft, respectively, averaged over the instances. From these tables, we can see that for our overall algorithmic approach, the objective values shown in Table 3 are mostly made up of uptime; the average delay for H1, H2 and CG is extremely small, indicating that our method is able to find solutions that ensure that all or nearly all requirements are picked up and delivered without delay. Moreover, we can see that as we progress from the baseline solution to the CG solution, the total delay generally decreases (for example, with $A = R = 100$, the total delay drops from 2.3 hours to 0.5 hours) and the total uptime also decreases. With regard to the number of aircraft, we can see that the solutions produced by our approach (the H1, H2 and CG solutions) are relatively efficient in terms of the number of aircraft used, and that the number of required aircraft decreases with higher numbers of requirements (for example, the CG solution uses over half of the aircraft for $A = R = 20$, but just under 30% of the aircraft for $A = R = 100$). More importantly, compared to the baseline method, our approach is able to achieve a significant reduction in the number of aircraft it uses. For example, with $A = R = 50$, the baseline method uses on average 26.4 out of the 50 aircraft, whereas the CG solution uses on average 20.8 aircraft; a reduction of 5.6 aircraft on average. We also emphasize that this reduction in aircraft is achieved with a simultaneous reduction in delay and uptime. These results underscore the potential of our overall algorithmic approach to deliver superior schedules compared to existing practice at USTRANSCOM.

5.4. Optimality experiment

One question we may have, when considering the results in Section 5.3, is regarding the lower bound produced by CG. In particular, due to the heuristic nature of Algorithm 2 that is used to solve the separation problem in Algorithm 1, the lower bound produced by Algorithm 1 is only an approximate bound, and the optimality gap that we compute for the end solution with respect to this bound (see Table 3) is therefore only an approximate gap. The question, then, is how far

Table 5 Delay (in hours) of baseline, H1, H2 and CG solutions for T instances.

A	R	Baseline Delay (hrs)	H1 Delay (hrs)	H2 Delay (hrs)	CG Delay (hrs)
10	10	0.0	0.0	0.0	0.0
20	20	0.1	0.1	0.0	0.0
50	50	1.9	1.5	0.7	0.9
100	100	2.3	2.0	0.8	0.5

Table 6 Uptime (in hours) of baseline, H1, H2 and CG solutions for T instances.

A	R	Baseline Uptime (hrs)	H1 Uptime (hrs)	H2 Uptime (hrs)	CG Uptime (hrs)
10	10	42.6	39.7	39.7	38.9
20	20	96.5	89.6	88.4	84.9
50	50	194.9	186.9	182.1	172.6
100	100	275.0	268.1	257.2	247.3

Table 7 Number of aircraft used in baseline, H1, H2 and CG solutions for T instances.

A	R	Baseline # Aircraft	H1 # Aircraft	H2 # Aircraft	CG # Aircraft
10	10	8.3	7.9	7.9	7.9
20	20	14.2	12.9	12.8	12.7
50	50	26.4	23.5	21.6	20.8
100	100	38.8	35.2	30.2	29.5

away is the approximate lower bound from the true lower bound and therefore, how close is the approximate gap to the true optimality gap?

To answer this question, we consider the T instances with $A = R = 10$ and $A = R = 20$, which were studied in Section 5.3. We use the final integer solution produced by CG and the final lower bound produced by CG with the heuristic separation procedure (Algorithm 2). We compare this integer solution and this lower bound to the exact lower bound, i.e., the exact optimal value of problem (4). This exact lower bound is obtained by running the same CG procedure as before, but solving the exact MIP formulation of the subproblem when Algorithm 2 fails to find a violated constraint. We refer to this exact CG procedure as “ExactCG”.

To analyze these results, we let $Z^*_{LP,ExactCG}$ denote the lower bound obtained from ExactCG. We also use (as in Section 5.3) $Z^*_{LP,CG}$ to denote the (approximate) lower bound obtained from CG and $Z^*_{IP,CG}$ to denote the objective value of the integer solution obtained from CG. We use “BoundGap” to denote the how far away the approximate lower bound is from the exact lower bound:

$$\text{BoundGap} = 100\% \times \frac{Z^*_{LP,CG} - Z^*_{LP,ExactCG}}{Z^*_{LP,ExactCG}}.$$

We use “SolGap” to denote how far away the integer solution produced by CG is from the exact lower bound:

$$\text{SolGap} = 100\% \times \frac{Z^*_{IP,CG} - Z^*_{LP,ExactCG}}{Z^*_{LP,ExactCG}}.$$

Table 8 Comparison of final integer solution and lower bound from CG to the exact lower bound from ExactCG.

A	R	$Z^*_{IP,CG}$	$Z^*_{LP,CG}$	$Z^*_{LP,ExactCG}$	BoundGap (%)	SolGap (%)
10	10	38.9	38.9	38.9	0.00	0.00
20	20	85.0	84.9	84.7	0.17	0.28

Table 9 Comparison of computation time required by CG and ExactCG.

A	R	Avg. CG Time (s)	Max. CG Time (s)	Avg. ExactCG Time (s)	Max. ExactCG Time (s)
10	10	23.7	36.8	53.2	95.3
20	20	333.3	828.8	37455.9	154414.0

Table 8 shows $Z^*_{LP,ExactCG}$, $Z^*_{IP,CG}$ and $Z^*_{LP,ExactCG}$, as well as BoundGap and SolGap, averaged over the twenty instances for $A = R = 10$ and $A = R = 20$. (We do not consider the larger instance sets because the exact subproblem MIP formulation required a prohibitively large amount of time to solve for those instances.) We can see that despite the heuristic nature of Algorithm 2, using this heuristic to solve the separation problem within the CG procedure (Algorithm 1) leads to the same lower bound as solving the separation problem exactly for $A = R = 10$ (the average BoundGap value is exactly 0%) and a very close lower bound for $A = R = 20$ (the average BoundGap value is 0.17%). In addition, we can also see that the integer solution produced by CG is optimal for $A = R = 20$, and on average is extremely close to optimal for $A = R = 20$. The main message of this experiment is that while the lower bound produced by CG is not theoretically guaranteed to be a true lower bound, empirically we have some assurance that it should be close (if not exactly equal) to the true lower bound.

In addition, it is also worthwhile to compare the timing performance of CG and ExactCG. Table 9 shows the average and maximum time, taken over the twenty instances with $A = R = 10$ and $A = R = 20$, for both CG and ExactCG. From this table, we immediately see that the computation time required for the ExactCG approach is extremely large (for $A = R = 20$, it requires over 10 hours on average, and in one instance almost two days of computation). In contrast, the time required for CG, which uses the heuristic subproblem procedure (Algorithm 2), is a fraction of what it is for ExactCG. For example, for $A = R = 20$, it requires about five minutes on average, and in one case almost 15 minutes. This table, together with the previous table (Table 8) establishes the value of CG over ExactCG; by using CG instead of ExactCG, we are able to obtain extremely close (if not identical) lower bounds, but in a fraction of the time needed for ExactCG.

5.5. Realistic data experiment

Finally, we compare our algorithmic approach against the baseline method using the planning instances derived from USTRANSCOM's internal requirement data set (the R instances; see Section 5.1). Our implementation of our algorithmic approach is the same as in Section 5.3, with one small modification. We still execute H1 ten times, but for the local search phase, instead of executing H2 for all ten initial solutions constructed by H1, we execute it from the best initial solution (the one with the lowest objective value) found by H1. This modification was necessary due to the large number of instances, and the instances being more challenging in that H2 requires more time to run than it did for the T instances (see Section 5.3). We do note that one could potentially run H2 in parallel on the instances produced by H1, as the multiple runs of H2 would not be dependent on each other.

We begin by examining the objective values. Table 10 shows the average objective value achieved by the different methods; as in Section 5.3, the objective represents the sum of the total delay and the total uptime. From this table, we can see that, as for the T instances in Section 5.3, there

Table 10 Objective values and related metrics of baseline, H1, H2 and CG methods for R instances.

A	R	Baseline Obj. (hrs)	H1 Obj. (hrs)	H2 Obj. (hrs)	CG Obj. (hrs)	CG Bound (hrs)	Avg. Improv. Baseline → CG (%)	Max. Improv. Baseline → CG (%)	CG Gap (%)
10	10	169.5	105.9	105.4	104.4	104.1	24.4	91.2	0.53
20	20	275.7	130.9	128.6	125.7	125.0	38.0	92.7	0.94
50	50	187.9	162.9	153.4	147.8	138.8	15.9	66.1	6.44
100	111	409.5	334.3	312.8	312.4	298.3	17.8	59.8	4.79

are improvements in the objective value from using our overall approach. These improvements, however, are more dramatic than for the T instances. In particular, the average improvement ranges from 15 to 40%, while in the best case, the improvement can range anywhere from about 60% to over 90%. In addition, for the $A = R = 20$ instances, there were two instances where the baseline method failed to produce a feasible solution. (These two solutions were not used to compute the average baseline objective value.) In contrast to the baseline method, the H1 method was able to find feasible solutions for all of the instances tested. With regard to the CG method, we can see that in general the improvement generated by CG is smaller on this set of instances than it is on the T instances; however, the CG method is still useful to consider here because it is able to give us an approximate bound of the optimal value. With regard to the CG bound and the approximate gap from CG, we can see that the gaps are of a reasonable magnitude (below 1% for $A \in \{10, 20\}$ and below 7% for $A \in \{50, 100\}$).

Table 11 compares the methods in terms of computation time. The column "H1 Time" reports the time required to execute H1 ten times, averaged over the instances for each value of A, as in Section 5.3. However, as alluded to earlier, H2 was only executed once in these instances, from the best solution produced by H1; therefore, the column "H2 Time" reports the time to execute H2 once, averaged over all of the instances. As with the T instances, the baseline method requires the least amount of time to run. The average time to run both H1 (with ten executions) and H2 (with one execution) is reasonable; even in the largest set of instances ($A = R = 100$), this time is no more than approximately six hours.

Comparing Table 11 to Table 4, the average time required to execute H2 once for the R instances is comparable or slightly higher than the average time required to execute H2 once for the T instances. The CG method also requires more time to execute for these instances than it does for the T instances. The main reason for this increase in the time for H2 and CG is that the R instances are less constrained; the time windows for the pickup and dropoff events found in the data are much wider than those in the simulated T instances. This has two consequences. The first consequence is that to achieve low delay and low uptime, one can load some aircraft with a large number of requirements. As a result, the requirement sets that are checked in the H2 phase tend to be large ones, so the time required to solve the single aircraft problem (2) is large. The same is true for the separation problem procedure (Algorithm 2) of the CG phase. The second consequence is that due to the large time windows, the single aircraft problem (2) is less constrained and admits more solutions, which makes it harder to solve than if the time windows were smaller (as in the simulated instances tested in Section 5.3).

Finally, we can compare the solutions at a finer level. Tables 12, 13 and 14 show the average delay, uptime and total number of aircraft, respectively, averaged over the instances. As for the simulated instances, we can see that for our overall algorithmic approach, the objective values shown in Table 10 are mostly made up of uptime; the average delay for H1, H2 and CG is extremely small, indicating that our method is able to find solutions that ensure that all or nearly all requirements are picked up and delivered without delay. We can also see that the solutions produced by our approach are very efficient in terms of the number of aircraft they use, even more so than they are for the simulated instances in Section 5.3. For example, with $A = 100$ and $R = 111$, our overall

Table 11 Computation times (in seconds) of baseline, H1, H2 and CG methods for R instances.

		Baseline	H1	H2	CG	H1+H2	H1+H2+CG
A	R	Time (s)	Time (s)	Time (s)	Time (s)	Time (s)	Time (s)
10	10	4.6	69.0	16.0	305.0	85.0	390.1
20	20	22.7	406.9	83.6	3532.2	490.5	4022.7
50	50	220.7	3001.5	1075.3	43750.5	4076.8	47827.3
100	111	1213.4	13632.3	5995.0	43644.7	19627.3	63272.0

Table 12 Delay (in hours) of baseline, H1, H2 and CG solutions for R instances.

		Baseline	H1	H2	CG
		Delay	Delay	Delay	Delay
A	R	(hrs)	(hrs)	(hrs)	(hrs)
10	10	0.1	0.4	0.2	0.2
20	20	0.2	0.4	0.4	0.1
50	50	0.1	0.1	0.0	0.0
100	111	3.6	0.0	0.0	0.0

Table 13 Uptime (in hours) of baseline, H1, H2 and CG solutions for R instances.

		Baseline	H1	H2	CG
		Uptime	Uptime	Uptime	Uptime
A	R	(hrs)	(hrs)	(hrs)	(hrs)
10	10	169.3	105.5	105.2	104.2
20	20	275.5	130.5	128.3	125.6
50	50	187.8	162.8	153.4	147.7
100	111	405.9	334.3	312.8	312.4

Table 14 Number of aircraft used in baseline, H1, H2 and CG solutions for R instances.

		Baseline	H1	H2	CG
A	R	# Aircraft	# Aircraft	# Aircraft	# Aircraft
10	10	3.6	3.5	3.4	3.4
20	20	6.6	6.4	5.9	5.7
50	50	12.9	12.5	11.4	12.4
100	111	23.0	23.8	22.0	22.1

approach is able to find a solution that delivers the requirements using only roughly 20% of the available aircraft.

Comparing these results to the baseline solution, the baseline achieves smaller delays for the smaller instances ($A \leq 20$) but equal or higher delays for the larger instances (for example, with $A = 100$, $R = 111$, the average delay is 3.6 hours, compared to 0.0 hours for the H1, H2 and CG solutions). With regard to uptime, H1, H2 and CG lead to a significant reduction in uptime over the baseline method. Finally, when comparing the number of aircraft, the baseline method is also fairly efficient in its use of aircraft, but our approach is able to find and exploit opportunities to reduce the number of aircraft; for example, the average number of aircraft decreases by 0.9 – almost one whole aircraft – for $A = R = 20$ and for $A = 100$, $R = 111$.

6. Conclusions

In this paper, we developed an approach based on modern optimization for solving the airlift planning problem faced by USTRANSCOM. The approach involves decoupling the planning problem into two problems – the problem of assigning requirements to aircraft and the problem of scheduling the corresponding pickup and dropoff events for each aircraft – and solving the problem using an initialization heuristic, a local search heuristic and a column/constraint generation procedure. We show using synthetic data and planning data used internally at USTRANSCOM that our approach is able to design missions that ensure the requirements are delivered with almost no delay, while reducing the total aircraft time used by over 10% on average relative to existing practice at USTRANSCOM; moreover, it is capable of doing so within an operationally feasible time frame. Given the very high cost associated with transporting requirements through airlift, our results suggest that our method can lead to significant cost savings for USTRANSCOM.

There are a number of interesting directions in which our approach can be extended. One major direction is to consider uncertainties in the problem. In the formulation of the problem in this paper, we assume that the data is known to the decision maker: we know the set of requirements, their weights and the travel times of the aircraft a priori. In reality, the set of requirements may not be known a priori; as time progresses, new requirements may materialize that need to be incorporated into the existing schedule. Similarly, the weights of the requirements and the time required by an aircraft to travel from one port to another may be subject to uncertainty. To address these and similar aspects of the problem, one may consider an adaptive and robust optimization approach (see Bertsimas, Brown, and Caramanis 2011).

Acknowledgments

The authors thank the associate editor and two anonymous referees for their thoughtful suggestions that have helped to improve the paper. The authors also thank Patrick McLeod and Theresa Baynes at USTRANSCOM's Joint Distribution Process Analysis Center for their support of this research.

References

Bent R, Van Hentenryck P, 2006 *A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. Computers & Operations Research* 33(4):875–893.

Bertsimas D, Brown DB, Caramanis C, 2011 *Theory and applications of robust optimization. SIAM Review* 53(3):464–501.

Bertsimas D, Gupta S, Lulli G, 2014 *Dynamic resource allocation: A flexible and tractable modeling framework. European Journal of Operational Research* 236(1):14–26.

Bertsimas D, Lulli G, Odoni A, 2011 *An integer optimization approach to large-scale air traffic flow management. Operations Research* 59(1):211–227.

Bertsimas D, Stock Patterson S, 1998 *The air traffic flow management problem with enroute capacities. Operations Research* 46(3):406–422.

Bertsimas D, Stock Patterson S, 2000 *The traffic flow management rerouting problem in air traffic control: A dynamic network flow approach. Transportation Science* 34(3):239–255.

Bezanson J, Edelman A, Karpinski S, Shah VB, 2017 *Julia: A fresh approach to numerical computing. SIAM Review* 59(1):65–98.

Carnes TA, Henderson SG, Shmoys DB, Ahghari M, MacDonald RD, 2013 *Mathematical Programming Guides Air-Ambulance Routing at Ornge. Interfaces* 43(3):232–239.

Cordeau JF, Laporte G, Potvin JY, Savelsbergh MWP, 2004 *Transportation on demand. Transportation, handbooks in operations research and management science* 14:429–466.

Desrochers M, Lenstra JK, Savelsbergh MWP, Soumis F, 1988 *Vehicle routing with time windows: optimization and approximation. Vehicle routing: Methods and studies* 16:65–84.

Dumas Y, Desrosiers J, Soumis F, 1991 *The pickup and delivery problem with time windows. European Journal of Operational Research* 54(1):7–22.

Dunning I, Huchette J, Lubin M, 2017 *Jump: A modeling language for mathematical optimization. SIAM Review* 59(2):295–320.

Garey MR, Johnson DS, 1979 *A Guide to the Theory of NP-Completeness. WH Freemann, New York* .

Gurobi Optimization, Inc, 2016 *Gurobi Optimizer Reference Manual.* URL `http://www.gurobi.com`.

Jagabathula S, 2014 *Assortment optimization under general choice*, (October 21, 2014). Available at SSRN: http://ssrn.com/abstract=2512831.

Laporte G, 2007 *What you should know about the vehicle routing problem. Naval Research Logistics* 54(8):811–819.

Lu Q, Dessouky M, 2004 *An exact algorithm for the multiple vehicle pickup and delivery problem. Transportation Science* 38(4):503–514.

Lubin M, Dunning I, 2015 *Computing in Operations Research using Julia. INFORMS Journal on Computing* 27:238–248.

Mahmoudi M, Zhou X, 2016 *Finding optimal solutions for vehicle routing problem with pickup and delivery services with time windows: A dynamic programming approach based on state-space-time network representations. Transportation Research Part B: Methodological* 89:19–42.

Nanry WP, Barnes JW, 2000 *Solving the pickup and delivery problem with time windows using reactive tabu search. Transportation Research Part B: Methodological* 34(2):107–121.

Robbert AA, 2013 *Costs of flying units in Air Force active and reserve components.* Technical report, RAND Corporation.

Ropke S, Cordeau JF, 2009 *Branch and cut and price for the pickup and delivery problem with time windows. Transportation Science* 43(3):267–286.

Ropke S, Cordeau JF, Laporte G, 2007 *Models and branch-and-cut algorithms for pickup and delivery problems with time windows. Networks* 49(4):258–272.

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

Savelsbergh M, Sol M, 1998 *Drive: Dynamic routing of independent vehicles. Operations Research* 46(4):474–490.

Savelsbergh MWP, Sol M, 1995 *The general pickup and delivery problem. Transportation Science* 29(1):17–29.

Sigurd M, Pisinger D, Sig M, 2004 *Scheduling transportation of live animals to avoid the spread of diseases. Transportation Science* 38(2):197–209.

Toth P, Vigo D, 2014 *Vehicle routing: Problems, methods, and applications*, volume 18 (SIAM).

Vielma JP, 2015 *Mixed integer linear programming formulation techniques. SIAM Review* 57(1):3–57.

Vossen TWM, Hoffman R, Mukherjee A, 2012 *Air traffic flow management.* Barnhart C, Smith B, eds., *Quantitative problem solving methods in the airline industry*, 385–453 (Springer).

Xu H, Chen ZL, Rajagopal S, Arunapuram S, 2003 *Solving a practical pickup and delivery problem. Transportation Science* 37(3):347–364.