Massively Parallel Dantzig-Wolfe Decomposition Applied to Traffic Flow Scheduling

Joseph Rios*

NASA Ames Research Center, Moffett Field, CA 94035

Kevin Ross†

University of California at Santa Cruz, Santa Cruz, CA 95064

Optimal scheduling of air traffic over the entire National Airspace System is a computationally difficult task. To speed computation, Dantzig-Wolfe decomposition is applied to a known linear integer programming approach for assigning delays to flights. The optimization model is proven to have the block-angular structure necessary for Dantzig-Wolfe decomposition. The subproblems for this decomposition are solved in parallel via independent computation threads. Experimental evidence suggests that as the number of subproblems/threads increases (and their respective sizes decrease), the solution quality, convergence, *and* runtime improve. A demonstration of this is provided by using one flight per subproblem, which is the finest possible decomposition. This results in thousands of subproblems and associated computation threads. This massively parallel approach is compared to one with few threads and to standard (non-decomposed) approaches in terms of solution quality and runtime. Since this method generally provides a non-integral (relaxed) solution to the original optimization problem, two heuristics are developed to generate an integral solution. Dantzig-Wolfe followed by these heuristics can provide a near-optimal (sometimes optimal) solution to the original problem hundreds of times faster than standard (non-decomposed) approaches. In addition, when massive decomposition is employed, the solution is shown to be more likely integral, which obviates the need for an integerization step. These results indicate that nationwide, real-time, high fidelity, optimal traffic flow scheduling is achievable for (at least) 3 hour planning horizons.

Nomenclature

$\mathbf{q}$	Dual variable vector
$\mathcal{F}$	Set of flights
$\mu(f,j)$	Minimum sector time of flight f in sector j
a_f	Air hold time for flight f
c_f^a	Air-holding cost for flight f
c_f^g	Ground-holding cost for flight f
D_i	Sub-matrix of connecting constraints relating to sub-block i
F_i	Sub-matrix used to generate subproblem i
g_f	Ground hold time for flight f
l	Number of subproblems
m_0	Number of connecting constraints
P_i	The set of feasible solutions to subproblem i
$w_{f,t}^k$	Binary variable over sector, flight, and time

*Aerospace Engineer, Automation Concepts Research Branch, Mail Stop 210-10, Joseph.L.Rios@nasa.gov. Member AIAA.

†Assistant Professor, School of Engineering, 1156 High Street, kross@soe.ucsc.edu.

I. Introduction

Traffic Flow Management (TFM) of the National Airspace System (NAS) endeavors to deliver flights from their origins to their destinations while minimizing delays and respecting all capacities. There are several models for solving this problem. Some models aggregate flights into flows[1–3] and others consider controls for individual flights.[4–7] Typically, the latter set of models are computationally difficult to solve for large-scale, high fidelity scenarios. One of the more heavily studied aircraft-level models presented by Bertsimas and Stock-Patterson[5] has runtime concerns that should not be overlooked.[8–11] Despite the computational issues, aircraft-level models are attractive due to their ability to encapsulate important information about the state of the NAS and their ability to provide detailed control actions targeting specific flights. This highlights the major tradeoff between aggregate and aircraft-level models: computation time versus solution detail. The Bertsimas and Stock-Patterson (BSP) particular model has been the focus or basis of several other research efforts. As of yet it has not been definitively demonstrated that the model is feasible for "real-world" implementation due to the computational barrier. Some research efforts deal with tractability by lowering the time resolution by up to an order of magnitude (15-minute time bins) and/or simplifying the airspace (generic sectors, limited number of airports, etc.).[4,5,12] It is an open research question as to what time resolution is appropriate for various planning horizons. Previous work on accelerating solution of the Bertsimas and Stock-Patterson (BSP) model focused on either decomposing the problem in time[8] or space,[10] or by reducing the number of flights under consideration.[9] All such methods are rough heuristics for arriving at "good" solutions in reasonable time.

In this paper a well-known, provably optimal approach to solving large-scale linear programs, Dantzig-Wolfe (DW) decomposition,[13] is applied to the BSP model. DW is implemented here with varying degrees of decomposition, from a small number of subproblems to a massively parallel version, wherein each subproblem represents constraints for only one or two flights. While many others in various application domains[14–16] have used DW, there is no clear example in the literature of such a high-level of parallelism as is accomplished here. There has been considerable work on applying DW to integer programs.[17–19] In this work the focus is solving the relaxation of the master problem using integer solutions to the subproblems. Two different integerization heuristics are then applied to obtain a final, integer solution. The present massively parallel version is shown to have positive traits in terms of solution quality, convergence, and overall runtime. By using DW, the computational barrier to implementing a large-scale, high fidelity, aircraft-level model may be lowered to the point that future decision support tools may make use of the model with commercial-off-the-shelf software and hardware. In general, there has been considerable interest in the computational qualities of DW for many decades. Ho, for example, reviews some of them from the 1980's[20] and more recently Tebboth completed a thesis on the topic.[21]

This paper is organized as follows. In Section II further details on the BSP model and DW decomposition are provided. This is followed by Section III where a description of the tools and implementation details for this study's experiments is provided including a discussion of the integerization heuristics. Next, in Section IV a description of the data used in the experiments along with how they were acquired is presented. Following the presentation of the data, Section V describes the experiments performed and their respective results. Finally, Section VI offers concluding remarks including potential future research directions.

II. Background

In this section, the BSP model and the DW algorithm are described after a brief discussion of the Traffic Flow Management problem. The BSP model is a binary integer program that neatly describes the issues associated with TFM (respecting capacities and schedules). The DW algorithm will allow parallel computation of smaller subproblems such that an optimal, relaxed solution can be found in reduced time.

II.A. Traffic Flow Management

The most complete description of the state-of-the-art in TFM is provided by Sridhar, Grabbe, and Mukherjee.[22] TFM is primarily concerned with capacities and delays in the NAS. In the most general sense, TFM endeavors to minimize delays in the system while respecting all capacities. Capacity values implicitly aid

in maintaining safety in the system. The key inputs to a TFM problem are typically demand and capacity forecasts. Estimating demand and capacity are active areas of research.

There are several modeling choices that define the particular TFM problem to be solved. For aircraft-level models, Bertsimas describes a taxonomy of models,[5] which are categorized according to parameters such as whether uncertainty is considered, whether air holding and/or rerouting of flights is allowed, and how many airports may be involved in the problem.

For this research, individual aircraft are considered along with every sector and airport in the NAS. Rerouting is not considered, but air holding is allowed. The input data are considered to be deterministic. A more precise description of the TFM problem addressed in this research is described in the next section.

II.B. Bertsimas-Stock Patterson Binary Integer Program

The BSP model[5] is used to perform scheduling that minimizes delay costs. The model as presented by Bertsimas and Stock-Patterson is given here:

$$\text{Minimize:} \quad \sum_f [c_f^g g_f + c_f^a a_f]$$

$$\text{Subject to:} \quad \sum_{f:P(f,1)=k} (w_{f,t}^k - w_{f,t-1}^k) \leq D_k(t), \qquad \forall\, k \in \text{Airports}, t \in \text{Time} \quad (1)$$

$$\sum_{f:P(f,\text{last})=k} (w_{f,t}^k - w_{f,t-1}^k) \leq A_k(t), \qquad \forall\, k \in \text{Airports}, t \in \text{Time} \quad (2)$$

$$\sum_{f:P(f,i)=j,P(f,i+1)=j'} (w_{f,t}^j - w_{f,t}^{j'}) \leq S_j(t), \qquad \forall\, j \in \text{Sectors}, t \in \text{Time} \quad (3)$$

$$w_{f,t+\mu(f,j)}^{j'} - w_{f,t}^j \leq 0 \qquad \forall\, f \in \mathcal{F}, j = P(f,i), j' = P(f,i+1) \quad (4)$$

$$w_{f,t}^j - w_{f,t-1}^j \geq 0 \qquad \forall\, f \in \mathcal{F}, j \in (f\text{'s flight path}) \quad (5)$$

$$w_{f,t}^j = \begin{cases} 1, & \text{if flight } f \text{ arrives at sector } j \text{ by time } t, \\ 0, & \text{otherwise.} \end{cases}$$

The air and ground delay (a_f and g_f, respectively) for each flight f are ultimately expressed in terms of the binary variables, w, through a substitution for a_f and g_f, which is fully described in the original paper.[5] For each flight, the model is able to decide if the flight needs to be held anywhere (and for how long) in order to satisfy the airport departure, airport arrival, and sector capacity constraints (constraints (1), (2), and (3), repsectively). Airports are denoted by k and sectors by j. Each of the capacities is a function of time, t. Each flight in the set of flights, $\mathcal{F}$, is described as an ordered list of distinct sectors, j, from a set of sectors, J, with earliest and latest feasible entry times for each of those sectors. For modeling purposes, airports are considered a subset of sectors with associated arrival and departure capacities. A sector in a flight path is denoted by $P(f,y)$, where f is the flight and y is the ordinal representing its place in the flight path. For ease of notation, $P(f,\text{last})$ is used to represent the last sector (usually an airport) in f's path. The parameters c_f^g and c_f^a are the costs of holding f on the ground or in the air, respectively, for one unit of time. The original model, as published, also contained a set of constraints to represent connecting flights. Those constraints are relaxed here.

The problem is also constrained by the physical and temporal limitations of the flights. Specifically, each flight spends, at least, the specified minimum sector time, $\mu(f,j)$, in each of its sectors, j as described by constraints (4). Coupled with the set of constraints enforcing the logic of the variables, i.e. all 0's before all 1's (constraints (5)), physical and temporal logic is satisfied.

A high-resolution (1-minute accuracy) is used in this paper. In addition, an accurate model of the airspace used is complete with all airports and en route sectors. Recall that the computational complexity of the model can be reduced by lowering the resolution (using, say, 15-minute time bins) or considering only a subset of airports.

II.C. Dantzig-Wolfe Decomposition

DW is a decomposition method for linear programs of block angular form (see, for example, figure 1). Borrowing some notation from Bertsimas and Tsitsiklis,[23] noting that bold variables represent vectors and capital letters represent matrices, begin with a linear program (LP) of the form:

$$\begin{aligned}
&\text{Minimize:} && \mathbf{c}_1'\mathbf{x}_1 + \mathbf{c}_2'\mathbf{x}_2 + \ldots + \mathbf{c}_l'\mathbf{x}_l && (6)\\
&\text{Subject to:} && D_1\mathbf{x}_1 + D_2\mathbf{x}_2 + \ldots + D_l\mathbf{x}_l = \mathbf{b}_0 && (7)\\
& && F_1\mathbf{x}_1 = \mathbf{b}_1 && (8)\\
& && F_2\mathbf{x}_2 = \mathbf{b}_2 && (9)\\
& && \vdots &&\\
& && F_l\mathbf{x}_l = \mathbf{b}_l && (10)
\end{aligned}$$

where $\{\mathbf{x}_i | i \in [1 \ldots l]\}$ are the decision variables.

Assuming each set $P_i = \{\mathbf{x}_i \geq 0 | F_i\mathbf{x}_i = \mathbf{b}_i\}$ is bounded[a], each P_i can be described as a convex combination of its extreme points, $\mathbf{x}_i^j, j \in J_i$. It follows that $\mathbf{x}_i = \sum_{j \in J_i} \lambda_i^j \mathbf{x}_i^j$ where $\sum_{j \in J_i} \lambda_i^j = 1$. Given this description of $\mathbf{x}_i$, a substitution can be made into the LP described in Eqns. (6) to (10) resulting in the following so-called *master program*:

$$\begin{aligned}
&\text{Minimize:} && \sum_{i=1}^{l} \sum_{j \in J_i} \lambda_i^j \mathbf{c}_i'\mathbf{x}_i^j\\
&\text{Subject to:} && \sum_{i=1}^{l} \sum_{j \in J_i} \lambda_i^j D_i\mathbf{x}_i^j = \mathbf{b}_0\\
& && \sum_{j \in J_1} \lambda_1^j = 1\\
& && \sum_{j \in J_2} \lambda_2^j = 1\\
& && \vdots\\
& && \sum_{j \in J_l} \lambda_l^j = 1
\end{aligned}$$

where $\{\lambda_i^j | \forall j \in J_i, \forall i \in [1 \ldots l]\}$ are the decision variables.

If the original formulation (Eqns. (6) to (10)) had m constraints (rows) and n variables (columns), then the master formulation has only $m_0 + l$ constraints (where m_0 is the number of coupling constraints), but an exponentially larger number of columns since there is a column for each extreme point generated by the sets of constraints illustrated in Eqns. (8) to (10). Notice that the decision variables in the master program are actually the weights on the extreme points of each P_i.

Since the vast majority of the λ variables are valued zero at any given iteration, most columns are irrelevant to the master. This leads to the heart of the decomposition algorithm: *column generation*. Only potentially useful columns are added to a so-called "reduced master" problem. For each P_i, an independent LP is created. The constraints of each of those subproblems is simply P_i and the objective function is generated by examining the master problem:

[a]This assumption is not necessary for DW decomposition in general, but it is sufficient for the BSP formulation and simplifies the mathematical discussion.

$$\begin{aligned} &\text{Minimize:} && (\mathbf{c}_i' + \mathbf{q}' D_i)\mathbf{x}_i \\ &\text{Subject to:} && F_i \mathbf{x}_i = \mathbf{b}_i \end{aligned} \tag{11}$$

where $\mathbf{q}$ is the dual variable vector associated with the connecting constraints, which is made available as usual through the simplex algorithm applied to the master problem. The master problem, in essence, *asks* the subproblem for a variable (column) that will improve the master's objective function. The subproblem provides a variable with the greatest reduced cost (described by objective (11)). This is accomplished by minimizing the reduced cost of all of the master's decision variables associated with that particular subproblem. The subproblem either provides such a variable or it indicates that it is unable to provide one that improves the master. The master iterates over all subproblems until it's objective can no longer be improved. When this occurs, it implies that there is no variable/column that can enter the master's basis that will improve its objective, thus the current solution is optimal and the algorithm terminates. Since there are a finite number of corner points for each subproblem and the subproblems are solved with some form of the simplex method, the DW algorithm is guaranteed to terminate.

The interested reader is directed to the original paper by Dantzig and Wolfe[13] and/or a modern textbook on linear optimization[23] for details on this decomposition method. For discussions of computational issues associated with DW, the work of Ho[20,24] is insightful and infomative.

III. Dantzig-Wolfe Implementation of the Bertsimas-Stock Patterson Model

In this section, the details of this study's implementation of DW are provided. This is followed by a discussion of the suitability of the BSP model for use with DW. Finally, the heursitics used to move from the relaxed DW solution to a purely integer solution are described in detail.

III.A. DW Implementation Details

Assuming there are n subproblems in a DW implementation, at any iteration of the algorithm there are up to n potential columns to add to the reduced master formulation. One must instate a policy governing which (if any) of the columns are to be included in the reduced master. Some options include choosing the column(s) with the greatest reduced cost(s), choosing the first available column(s), or choosing all available columns that will improve the objective. The implementation for this study accepts all columns with strong potential to improve the master's objective (i.e., all columns with negative reduced cost).

Since there will be up to n columns entering the reduced master at each iteration of the DW algorithm, a decision needs to be made as to what should be done with the columns that exit the basis of the reduced master. This question is generally one that needs to be answered based on computing resources. If memory usage is (or will become) an issue, then at some point the non-basic columns (i.e., those with value of zero) should be purged from the reduced master. If memory usage is not a concern, then there is little to no harm in allowing the non-basic columns to remain in memory and, therefore, remain part of the reduced master formulation.[25] A benefit to keeping the non-basic columns is that there is an opportunity to use them later in the important integerization step. How these non-basic columns may be used in obtaining an integer solution will be discussed in Section III.C. Since these columns will prove useful and there were not any issues with becoming memory-bound during these experiments, this study chose to retain all columns that became part of the reduced master.

The next major implementation issue deals with how the subproblems should be solved. This decision is based on several factors including coding complexity, computing resources and problem size. Some implementations of DW may only have a single subproblem and, thus, do not have to be concerned with the decision to solve subproblems in parallel or serially. To minimize wall clock runtime, solving all subproblems in parallel is more efficient in the presence of multicore or cluster computers. For this study, all subproblems are launched simultaneously and each solves completely at each iteration of the DW algorithm as long as the subproblem has an new objective function for that iteration. The entire implementation is written in 'C' and multithreading is achieved through the use of the pthread library.

Finally, a decision on the form of the variables supplied to the master needs to be made. More specifically, should the integrality constraints be enforced on the subproblem solutions? This question is intimately tied to the question of how integrality is going to be handled in the final solution. As will be discussed in Section III.C, the heuristics used to obtain an integral solution will rely on the fact that the subproblems are asked to supply an integer (binary) solution to the master at each iteration.

III.B. Block-Angular Form of the BSP Model

As discussed in Section II.C, a linear program needs to exhibit block-angular form in order to become a candidate for DW decomposition. The BSP model is, indeed, of this form. The capacity constraints are the so-called "connecting" constraints since they involve variables from many different flights. The groups of flights involved in those capacity constraints cannot be easily disaggregated. For example, the set of flights which may use sector ZOB43 at time 21 will not likely be exactly the same set of flights that use any other resource (sector or airport) in the system.

Constraints (4) and (5) are used to generate the independent subproblems. This is due to the fact that these constraints for each individual flight are independent of those for any other flight. It follows that any collection of constraints associated with a set of flights is completely independent of the collection of constraints associated with a mutually exclusive set of flights. Using this insight, any number of subproblems can be generated by simply grouping all of the physical/temporal constraints associated with distinct sets of flights. More concretely, if 10 subproblems are desired, a list of all the flights within the planning horizon can be divided into ten distinct lists. Now all physical/temporal constraints associated with the flights in each list is completely independent of each other set of physical/temporal constraints. By grouping the constraints in this manner, the constraint matrix of the original problem is partitioned into independent rows. With the understanding that the ordering of columns within a constraint matrix is arbitrary, a block angular structure can be extracted from the BSP by re-ordering columns to group flights based on any desired criteria. To formalize, consider the complete set of flights $\mathcal{F}$ partitioned into n sets. For every $i \in n$, the set of physical and temporal constraints ((4) and (5)) can now be grouped into a set of constraints, F_i, such that all of the variables, w, used within those constraints are mutually exclusive of those used within every other set of constraints, $F_j, i \neq j$ (see figure 1).

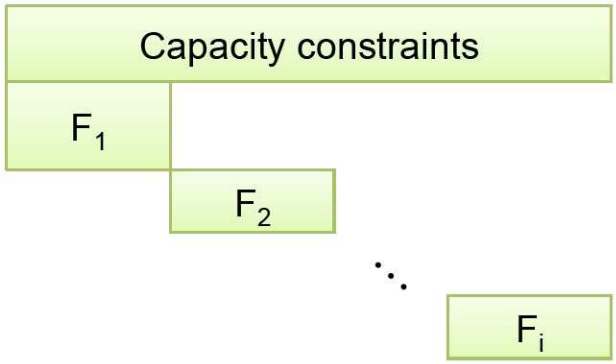

Figure 1. The block-angular form of the Bertsimas-Stock Patterson model. Each F_i represents the complete physical and temporal constraints of a distinct set of flights.

III.C. Integerization Heuristics

Since DW only provides a relaxed (i.e., not necessarily integral) solution to the original integer formulation, some method to generate an integer solution from this relaxed solution is required. A significant amount of research has been directed toward applying DW to integer programs. For example, Vanderbeck has developed methods for binary and general integer programs,[17,18] and mixed integer programs.[19] Concepts from Vanderbeck's work may be useful to future studies in the Traffic Flow Management domain, but the work presented here will rely on using the relaxed solution from DW and applying heuristics to obtain an integer solution rather than explicitly incorporating integrality throughout the solve process. The assumption in this study is that applying a heuristic at the end of the relaxation solve process will ultimately be more efficient than other integerization methods. The two integerization methods are now presented.

III.C.1. 'Choose Exactly One'

After obtaining a relaxed solution for any problem of at least nominal complexity, the reduced master problem will likely contain several columns generated by each subproblem. Specifically, if there were i iterations used to obtain the relaxed solution, each subproblem will have contributed between 1 and i columns to the reduced master. Since the choice was made to retain all columns that enter the reduced master problem (see Section III.A), the following question can be posed: From the generated columns can the reduced master obtain a feasible solution by selecting exactly one of the columns from each of the subproblems? This corresponds to enforcing a binary constraint on the weights (λ_i^j, see Section II.C). So instead of a solution consisting of a convex combination of the columns suggested by each subproblem, the reduced master must now choose one, and only one, column from each subproblem. This method is only applicable since integrality is strictly enforced in the subproblems (see Section III.A).

III.C.2. 'Rounding'

A crude, but highly efficient method for obtaining an integer assignment of variables is to simply round them to the nearest integer. Within this implementation of DW, integrality of the variables is enforced in the subproblems, but that integrality may be lost when solving the master problem. This is due to the fact that weights (λ_i^j, see Section II.C) are not necessarily integer. Rounding must be done with the understanding that the feasibility of the solution obtained by rounding the variables of a known feasible solution is not guaranteed. However, careful examination of what rounding implies in the context of the BSP model is interesting.

In the BSP model, the physical/temporal constraints should never be broken if the model is to have any basis in reality. If any of these constraints were to be violated, the implication is that a flight has, for example, flown across a sector faster than physically possible or perhaps bypassed a sector completely. It can be easily shown that, given a valid assignment of variables to satisfy these physical/temporal constraints, the rounded assignment to these variables maintains the satisfaction of these inequalities. To illustrate, consider the following:

$$
\begin{aligned}
w_1 - w_2 &\geq 0 \\
w_1 &\geq w_2 && (12)\\
w_1^r &\geq w_2^r && (13)\\
w_1, w_2 &\in [0,1], \\
w_i^r &\text{ is the rounded value of } w_i
\end{aligned}
$$

If Ineq. (12) is satisfied by some assignment of w_1 and w_2, then the fact that Ineq. (13) can be established via simple enumeration, as presented here. There are only three scenarios to check, given Ineq. (12):

$$
\begin{aligned}
w_1, w_2 &\geq 0.5 && (14)\\
w_1, w_2 &< 0.5 && (15)\\
w_1 &\geq 0.5 \text{ and } w_2 < 0.5 && (16)
\end{aligned}
$$

For the scenario presented by Ineqs. (14), both variables will be rounded to 1, thus satisfying Ineq. (13). Likewise, the scenario presented by Ineqs. (15), both variables will be rounded to 0 and will satisfy Ineq. (13). For the final scenario (Ineqs. (16)), w_1 will be rounded to 1 while w_2 is rounded to 0 which still satisfies Ineq. (13). This exposition implies that constraints (4) and (5) will not be violated if we are given a relaxed solution and decide to round the variables.

The same argument cannot be made for constraints (1) through (3). These constraints represent the maximum allowable capacity of the various resources. Violating these constraints, however, does not shatter any physical reality associated with aviation. In fact, in the NAS capacity values are often "violated" in the course of a typical day's operation. Decisions are made dynamically by air traffic managers as to whether

the violation of a sector's MAP value, for example, will put the system into an unsafe state. This assessment is made based on several factors including, but not limited to, the sector controller's experience, weather, and traffic patterns. In addition to this understanding of capacity constraints, it is clear that there is a large degree of uncertainty in the air traffic system. With these understandings and the computational costs of solving BSP, it may be reasonable to provide a solution that violates a small number of capacity constraints if it can be calculated relatively quickly. Thus, rounding is implemented as an heuristic to measure how quickly the integer solution can be obtained. The resulting rounded solution is examined to count the number of broken capacity constraints.

III.D. Tools

All experiments were performed on a dual quad-core Xeon with a clock speed of 2.66 GHz and 32 GB of available memory. The operating system was Red Hat Enterprise Linux 4. The optimization library used for implementing DW was the open source package GNU Linear Programming Kit (GLPK) version 4.34.[26] Vector and matrix math operations were performed with the Intel Math Kernel Library version 10.0.[27] The commercial optimization package CPLEX version 11.2[28] was used for the integerization described in Section III.C.1 and for obtaining solutions to the standard, monolithic (non-decomposed) versions of the BSP (see Section V). In general, CPLEX outperforms GLPK by a large margin, but the restrictive licensing of CPLEX prohibited this study from running in the multi-threaded nature it required. A comparison between GLPK and CPLEX is provided in table 1. The tool flow is illustrated in Section V with figure 2. It is worth noting at this point that use of a mathematical modeling language is omitted for optimal runtime performance. Languages such as AMPL or GAMS provide a clean method for describing any linear program, but the computational cost involved in translating that model into data structures used by optimization packages like GLPK or CPLEX is significant. Table 1 again provides measurements supporting this claim. If runtime is a concern, then a linear program should be described in a more "native" format like MPS or CPLEX's LP format. For this study, the latter was used.

IV. Data

A flight data set was chosen such that it represented a typical day with several resources operating at (or potentially over) capacity, but without any significant influence from poor weather. The ASDI (Aircraft Situation Display to Industry)[29] data for Thursday, August 24th, 2005 starting at 13:15 UTC (9:15 AM EDT) was used for all experiments. This represents the fairly dense mid-to-late morning traffic on the east coast, as well as the earlier morning rush on the west coast. This project was focused on only domestic flights so all international traffic were excluded, though their inclusion would not likely change the runtime or solution quality. Flights through Canadian airspace were retained, but the Canadian sector constraints were ignored. In addition, a small percentage of flights were omitted. These mostly included flights that re-entered the same sectors several times or were missing sectors in their flight plans. Over 90% of all flights were retained after the deletion of international and other flights. The data is considered high fidelity for this problem domain since the position of each flight is predicted for each minute of the simulation.

As far as capacity constraints are concerned, any sector or airport that was used by any flight in the system was included in the data set. This resulted in 974 sectors being included along with 905 airports. Many of the constraints generated by these resources (see constraints (1) and (2)) were not near their maximum capacity levels and were, thus, simplified out of the problem formulation. For example, if only a few flights are using Moffett Federal Airfield, then the capacity constraints generated for the BSP model could never be binding and, thus, could be removed from the formulation. There was no distinction made between low, high or superhigh en route sectors, but Terminal Radar Approach Control (TRACON) sectors were given special status by allowing them infinite capacity. The basis for this decision is that the capacity of these sectors is typically driven by the ability of the airports in the TRACON to accept or depart aircraft and those constraints (again, constraints (1) and (2)) are already included in the formulation.

To extract the data from the ASDI file, the Future ATM Concepts Evaluation Tool (FACET)[30] was used. FACET is capable of many functions, but for this study, its capabilities of playing back and simulating historical data and recording statistics were most relevant. Easing the burden of collecting data and con-

trolling simulations was the FACET Application Programming Interface (API) (based on a research tool called CARAT#[31]). The FACET API allows the user to interface with FACET through either Java or Matlab code in order to extract information about elements in the system. The default sector capacities known as Monitor Alert Parameters (MAP) and airport capacities as stored in FACET were used for the experiments. If an airport did not have explicit capacities in FACET, a default value of 10 was used for arrival and departure capacity per 15 minutes.

V. Experiments and Results

The primary measurement for this study is runtime. Secondarily, solution quality is closely observed. Several scenarios were generated from the base data set described in Section IV. These scenarios were generated by varying two key parameters, planning horizon and sector capacity. The planning horizon was varied from 60 minutes to 120 minutes to 180 minutes. The sector capacity was varied from a nominal value of 100% of MAP value down to a value of 70% throughout the entire NAS. This series of capacity values are not meant to replicate any particular "real-life" scenario, rather they were constructed to stress the model and solvers under more and more tightly constrained scenarios. Each experimental scenario will be referred to by the notation 'min/cap' where 'min' is the planning horizon and 'cap' is the percentage of nominal sector capacity available. For all experiments, the maximum allowable lateness for each flight into each of its sectors is set at 20 minutes. Any scenario that is run for an n-minute planning horizon is solved with a model with an $(n+20)$-minute planning horizon to allow all flights in the scenario to reach the final sector in their flight path for that planning horizon. The overall tool flow for each of the experiments is illustrated in figure 2.

Figure 2. The general tool flow used in the experiments. "Monolithic" refers to the original, non-decomposed model.

To solve these scenarios, the number of subproblems is varied. Due to an unwieldy experimental matrix, the selection of the number of subproblems was limited for most planning horizon/capacity pairs. For every planning horizon/capacity pair, the monolithic version of the BSP was run. In addition, a 256 subproblem instance was run for each planning horizon. Finally, a 5920, 4252, or 5370 subproblem instance was run for the 60, 120, or 180 minute scenarios, respectively. The number of these subproblems was chosen based on the number of flights in the scenario. For the 60 minute scenario, there are 5920 flights in the system. This number is a soft upper bound on the number of threads that could be run on the available hardware and operating system. For the 120 minute and 180 minute scenarios, a number of subproblems was chosen such that there were 2 flights per subproblem. There were 8505 flights in the 120 minute scenario and 10,741 flights in the 180 minute scenario resulting in 4252 and 5370 subproblems, respectively. For the 60/100 scenario, a larger set of subproblems were attempted. Specifically for that horizon/capacity pair, 16, 32, 64, 128, and 512 subproblems were also attempted. The chosen number of subproblems attempted for all other horizon/capacity pairs was determined based on the results from the full set of 60/100 experiments (see the results in figure 3).

To clarify, the reason for lower-quality solutions in the instances with fewer subproblems has to do with the smaller search space for the 'choose-one' heuristic. With 16 subproblems, for example, forcing a choice of one column per subproblem after solving the relaxation severely limits the search space for feasible solutions. As the number of subproblems increases, the complexity of each subproblem is diminished and the search space for the 'choose-one' heuristic is expanded resulting in solutions that are closer to the relaxed optimal. Specifically, note that with the maximal number of subproblems (5920 in the 60-minute scenario), there are a larger number of simpler columns from which the 'choose-one' heuristic can find a solution.

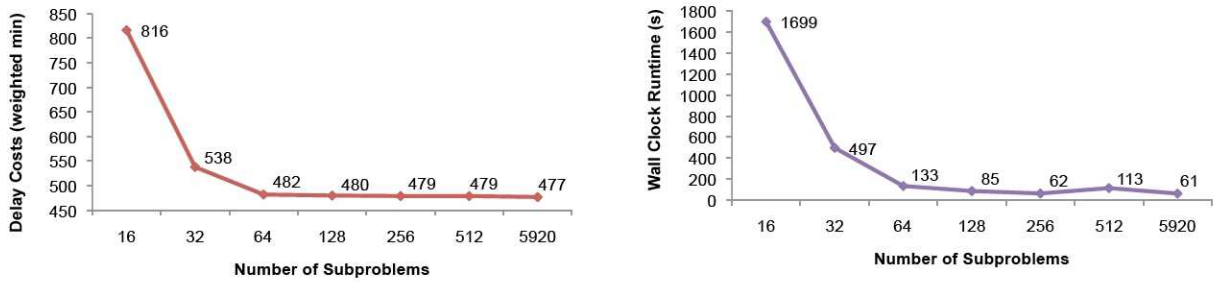

Figure 3. Results from the 60 minute, 100% capacity experiments using the 'choose one' integerization heuristic.

It is interesting to see the trends for runtime and solution quality so closely related. It is important to note that for the 60/100 scenario, the optimal relaxed solution was 477. Thus, the 5920 subproblem version of the 60/100 scenario was able to find an integer solution with the known optimal value while the other runs with fewer subproblems were not. To obtain the optimal solution using GLPK and a monolithic version of the BSP model will take over 2 days (229610 seconds), while using DW and multiple threads in parallel, the optimal solution is available in 61 seconds.

Scenario (min./capacity)	Metric	GLPK LP Format	AMPL/ CPLEX	CLPEX LP Format	GLPK DW with max. threads
60/100	Runtime (s)	229,610	2294	108	61
	Delay Cost	477	477	477	477
60/90	Runtime (s)	254,651	3360	1132	72
	Delay Cost	759	758	759	750
60/80	Runtime (s)	357,483	2576	216	88
	Delay Cost	1305	1302	1305	1304
60/70	Runtime (s)	6+ days	65,399	61,238	137
	Delay Cost	–	2742	2792	2587
120/100	Runtime (s)	–	–	460	101
	Delay Cost	–	–	1005	997
120/90	Runtime (s)	–	–	636	139
	Delay Cost	–	–	1626	1488
180/100	Runtime (s)	–	–	1372	189
	Delay Cost	–	–	1248	1233

Table 1. Scenario comparison over various optimization tools. Runtime efficiency increases left to right.

To understand the significance of the runtimes, table 1 illustrates the runtime and solution quality performance of several optimization approaches to solving the same BSP instances. Several important observations can be made from this table. The most important result is that DW implemented with GLPK consistently beats the non-decomposed version running on CPLEX. This bodes well for any future implementation that might make full use of multiple, parallel CPLEX solvers, or for the financial efficiency of using liberally licensed, open source software. It would be expected that a DW implementation using CPLEX would significantly outperform the GLPK DW implementation. Next, the cost of using a mathematical modeling language instead of a more native format is quite steep. The runtimes for AMPL/CPLEX should be com-

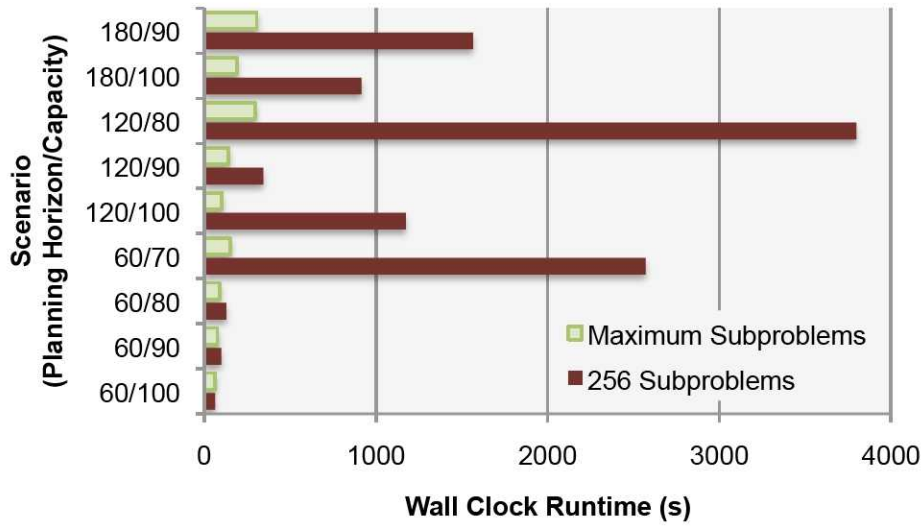

Figure 4. Runtime by scenario and number of subproblems. More subproblems lead to shorter runtimes even without adding CPUs/cores.

pared to the CPLEX LP Format results to see the runtime cost of mathematical model translation. Finally, the difference in solving power between a standard GLPK installation and CPLEX is starkly illustrated. Although there is much to be said for the ability to view and modify the source code of GLPK as well as launch as many instances of GLPK as desired without licensing worries.

Scenario	Solution Quality		Difference
	256 Subprobs.	Max Subprobs.	[%]
60/100	479	477	0.42
60/90	760	758	0.26
60/80	1551	1412	8.96
60/70	3565	3311	7.12
120/100	1008	997	1.09
120/90	1500	1488	0.80
120/80	2704	2638	2.44
180/100	1238	1233	0.40
180/90	2308	2039	11.66

Table 2. DW solution quality comparison. 'Max' indicates the maximum number of threads attempted for the scenario. Note the Instances with more threads always outperform those with fewer.

With this initial set of results for the 60-minute scenarios, each other, lengthier scenario was run with 256 and a "maximum" number of subproblems (a maximum of 2 flights per subproblem as discussed above). These results indicate that, indeed, the runtime improves along with the integer solution quality as the number of subproblems increases. Specifically, table 2 shows the integer solution quality difference between the 256 subproblem cases and the maximum subproblem cases, while figure 4 illustrates the runtimes of the 256 and maximum subproblem cases.

In the event that solving the integerization problem from the relaxed result takes too long, it is possible to use the rounding heuristic described in Section III.C.2. The longest wall clock time that this heuristic took to complete was 3 seconds more than the the time it takes to solve the relaxation. After rounding, data was readily available describing how many constraints were broken and by how much. The full set of results regarding broken constraints was generated for the 60 minute planning horizon scenarios and is presented in table 3.

Horizon/Capacity/Subproblems	Broken Constraints
60/100/16	7
60/100/32	6
60/100/64	4
60/100/128	2
60/100/256	3
60/100/512	2
60/100/5920	0
60/90/256	14
60/90/5920	4
60/80/256	27
60/80/5920	11
60/70/256	173
60/70/5920	163

Table 3. The number of broken constraints generated when applying the rounding integerization heuristic.

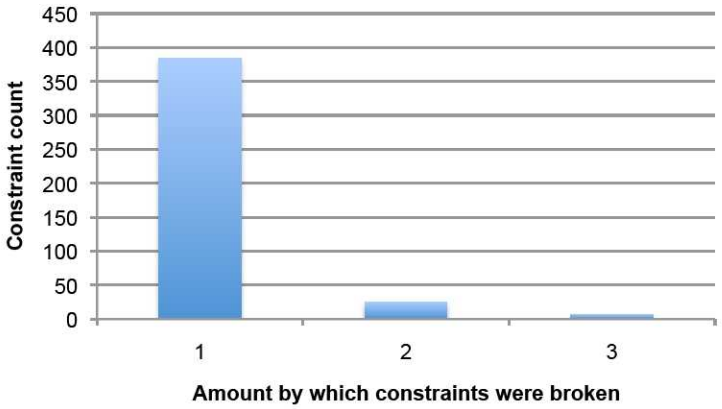

Figure 5. Histogram over all 60-minute scenarios of broken constraints generated by rounding heuristic.

It is important to put the concept of a "broken constraint" into practical perspective. In the NAS the occupancy of a sector is sampled every minute. For each 15-minute time bin for each sector, that sector/time pair is considered over-capacity if any of the corresponding occupancy samples is above MAP. By this definition of capacity violation, sectors are known to go over capacity on a daily basis. The constraint violations counted in table 3 are equivalent to the one-minute samples taken in practice. Also, the constraint violations typically are temporally adjacent sector-time pairs. Thus, often a set of constraint violations will translate to only a single capacity violation as defined in the NAS if that schedule were to be enacted. For example, in the 60/90/256 case there were 14 constraint violations in the integer program when rounding was used as a heuristic, but that translated to only 4 practical sector capacity violations. For the 180/80/5370 case, there were 195 constraint violations that mapped to, roughly, 34 practical sector capacity violations. Another point on the broken constraints using the rounding heuristic is that there are 900+ sectors in the NAS. Over a 180 minute run, there are twelve 15-minute time bins. There are, therefore, over 10,000 sector-time pairs that have the potential for going over capacity. If a solution can be provided such that a traffic flow manager need only worry about 34 of them being violated, it is likely that the solution can be useful in a practical setting. Hunter et al.[32] provide a deeper discussion on the relationship between capacity, occupancy, congestion, and delay.

Figure 5 demonstrates that when constraints were broken by the use of rounding, they were not broken by much. The vast majority of constraint violations occurred by going over the bound by a single flight. The

fact that the constraints are not "too" broken, lends further potential for this rounding method to produce solutions that might be feasible in a practical setting or at least solutions that may provide a basis from which a human traffic flow manager might make intelligent decisions.

VI. Conclusion

The work presented here demonstrates that for practical Traffic Flow Management problems, Dantzig-Wolfe decomposition is a very efficient method of obtaining a high-quality solution. In addition, the solution quality *and* the runtime improve as the number of subproblems increases. This bodes well for any future decision-support tools for Traffic Flow Management because it is likely, thanks to a rough translation of Moore's Law, that the number of CPUs/cores available for use on this problem will increase faster than traffic density, thus ensuring mostly tractable problems for the foreseeable future. While the runtime does not halve with each doubling of subproblems, it must be remembered that the machine on which these experiments were run was bound by its 8-core limit. As more CPUs/cores become available (either through implementation on a cluster machine or in the future with higher core-count machines), the wall clock runtime will likely decrease for implementations with more subproblems. This tractability also implies that additional complexity may be accommodated within the model. Rerouting and uncertainty may be included in future implementations of the Bertsimas-Stock Patterson model and solutions may still be achieved in reasonable time.

The results presented here also demonstrate that given an optimal, relaxed solution to a Bertsimas-Stock Patterson instance, a rounding heuristic provides a rapid, near feasible solution that may be of some benefit to a traffic flow manager. In addition, it appears that problem instances wherein a relaxed solution has the same value as an integer solution, rounding will be unnecessary as Dantzig-Wolfe provides an integer optimal solution when massive decomposition is employed. Joining these insights and potentially connecting a Dantzig-Wolfe system with other approaches (such as problem-size reduction,[9] spatial decomposition,[10] or temporal decomposition[8]) makes it reasonable to state that large-scale, high fidelity, Traffic Flow Management problems are likely solvable in real-time (i.e., guaranteed solutions in a prescribed amount of time). This insight opens the door to decision support tool developers and researchers to investigate the use of optimal, aircraft-level models (like Bertsimas-Stock Patterson) in future tools.

References

[1]Sridhar, B., Soni, T., Sheth, K., and Chatterji, G., "An Aggregate Flow Model for Air Traffic Management," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Providence, Rhode Island, August 2004.

[2]Menon, P., Sweridul, G., and Bilimoria, K., "New Approach for Modeling, Analysis, and Control of Air Traffic Flow," *Journal of Guidance, Control, and Dynamics*, Vol. 27, No. 5, 2004, pp. 737–744.

[3]Bayen, A. M., Raffard, R. L., and Tomlin, C. J., "Adjoint-based control of a new Eulerian network model of air traffic flow," *IEEE Transactions on Control Systems Technology*, Vol. 14, No. 5, September 2006, pp. 804–818.

[4]Lindsay, K. S., Boyd, E. A., and Burlingame, R., "Traffic Flow Management Modeling with the Time Assignment model," *Air Traffic Control Quarterly*, Vol. 1, No. 3, 1993, pp. 255–276.

[5]Bertsimas, D. and Patterson, S. S., "The Air Traffic Flow Management Problem with Enroute Capacities," *Operations Research*, Vol. 46, No. 3, May-June 1998, pp. 406–422.

[6]Bayen, A. M., Grieder, P., Meyer, G., and Tomlin, C. J., "Lagrangian Delay Predictive Model for Sector-Based Air Traffic Flow," *Journal of Guidance, Control, and Dynamics*, Vol. 28, No. 5, 2005, pp. 1015–1026.

[7]Lulli, G. and Odoni, A. R., "The European Air Traffic Flow Management Problem," *Transportation Science*, Vol. 41, No. 4, November 2007, pp. 431–443.

[8]Grabbe, S., Sridhar, B., and Mukherjee, A., "Central East Pacific Flight Scheduling," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Hilton Head, South Carolina, August 2007.

[9]Rios, J. and Ross, K., "Solving High-Fidelity, Large-Scale Traffic Flow Management Problems in Reduced Time," *AIAA Aviation Technology, Integration and Operations Conference*, Anchorage, Alaska, September 2008.

[10]Rios, J. and Ross, K., "Parallelization of the Traffic Flow Management Problem," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Honolulu, Hawaii, August 2008.

[11]Roy, K. and Tomlin, C. J., "Traffic Flow Management Using Supply Chain and FIR Filter Methods," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2008.

[12]Bertsimas, D. J., Lulli, G., and Odoni, A. R., "The Air Traffic Flow Management Problem: An Integer Optimization Approach," *Lecture Notes in Computer Science*, , No. 5035, 2008, pp. 34–46.

[13]Dantzig, G. B. and Wolfe, P., "Decomposition Principle for Linear Programs," *Operations Research*, Vol. 8, No. 1, January-February 1960, pp. 101–111.

[14]Desaulniers, G., Desrosiers, J., Dumas, Y., Marc, S., Rioux, B., Solomon, M., and Soumis, F., "Crew Pairing at Air France," *European Journal of Operational Research*, Vol. 97, 1997, pp. 245–259.

[15]Prescott, E. S., "Computing Solutions to Moral-Hazard Programs Using the Dantzig-Wolfe Decompostion Algorithm," *Journal of Economic Dynamics and Control*, Vol. 28, 2004, pp. 777–800.

[16]Fuller, J. D. and Chung, W., "Dantzig-Wolfe Decompostion of Variational Inequalities," *Compuational Economics*, Vol. 25, 2005, pp. 303–326.

[17]Vanderbeck, F. and Wolsey, L. A., "An Exact Algorithm for IP Column Generation," *Operations Research Letters*, Vol. 19, 1996, pp. 151–159.

[18]Vanderbeck, F., "On Dantzig-Wolfe Decomposition in Integer Programming and Ways to Perform Branching in a Branch-and-Price Algorithm," *Operations Research*, Vol. 48, No. 1, January-February 2000, pp. 111–128.

[19]Vanderbeck, F. and Savelsbergh, M. W., "A Generic View of Dantzig-Wolfe Decomposition in Mixed Integer Programming," *Operations Research Letters*, Vol. 34, 2006, pp. 296–306.

[20]Ho, J. K., "Recent Advances in the Decompostion Approach to Linear Programming," *Mathematical Programming Study*, Vol. 31, 1987, pp. 119–127.

[21]Tebboth, J. R., *A Computational Study of Dantzig-Wolfe Decomposition*, Ph.D. thesis, University of Buckingham, 2001.

[22]Sridhar, B., Grabbe, S., and Mukherjee, A., "Modeling and Optimization in Traffic Flow Management," *Proceedings of the IEEE*, Vol. 96, No. 12, December 2008.

[23]Bertsimas, D. and Tsitsiklis, J. N., *Introduction to Linear Optimization*, Athena Scientific, Belmont, Massachusetts, 1997.

[24]Ho, J. K. and Loute, E., "An Advanced Implementation of the Dantzig-Wolfe Decomposition Algorithm for Linear Programming," *Mathematical Programming*, Vol. 20, 1981, pp. 303–326.

[25]O'Neill, R. P., "Column Dropping in the Dantzig-Wolfe Convex Programming Algorithm," *Operations Research*, Vol. 25, No. 1, January-February 1977, pp. 148–155.

[26]Makhorin, A., "GNU Linear Programming Kit, Version 4.34," http://www.gnu.org/software/glpk/glpk.html.

[27]Intel Corporation, "The Intel Math Kernel Library," http://www.intel.com/cd/software/products/asmo-na/eng/307757.htm, January 2009.

[28]ILOG, Inc., "ILOG CPLEX," http://www.ilog.com/products/cplex/, April 2007.

[29]"Aircraft Situation Display To Industry: Functional Description and Interface Control Document," Tech. Rep. ASDI-FD-001, Volpe National Transportation Center, U.S. Department of Transportation, June 2005.

[30]Bilimoria, K., Sridhar, B., Chatterji, G., Sheth, K., and Grabbe, S., "FACET: Future ATM Concepts Evaluation Tool," *ATM2000*, Napoli, Italy, June 2000.

[31]Menon, P., Diaz, G., Vaddi, S., and Grabbe, S., "A Rapid-Prototyping Environment for En Route Air Traffic Management Research," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, San Francisco, CA, August 2005.

[32]Hunter, G., Boisvert, B., and Ramamoorthy, K., "Advanced National Airspace Traffic Flow Management Simulation Experiments and Validation," *Proceedings of the 2007 Winter Simulation Conference*, edited by S. Henderson, B. Miller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, 2007.