Scheduling Aircraft Landings under Constrained Position Shifting

Hamsa Balakrishnan [*]

Stanford University, Stanford, CA 94305, USA

Bala Chandran[†]

University of California at Berkeley, Berkeley, CA 94720, USA

Optimal scheduling of airport runway operations can play an important role in improving the safety and efficiency of the National Airspace System (NAS). Methods that compute the optimal landing sequence and landing times of aircraft must accommodate practical issues that affect the implementation of the schedule. One such practical consideration, known as Constrained Position Shifting (CPS), is the restriction that each aircraft must land within a pre-specified number of positions of its place in the First-Come-First-Served (FCFS) sequence.

We consider the problem of scheduling landings of aircraft in a CPS environment in order to maximize runway throughput (minimize the completion time of the landing sequence), subject to operational constraints such as FAA-specified minimum inter-arrival spacing restrictions, precedence relationships among aircraft that arise either from airline preferences or air traffic control procedures that prevent overtaking, and time windows (representing possible control actions) during which each aircraft landing can occur. We present a Dynamic Programming-based approach that scales linearly in the number of aircraft, and describe our computational experience with a prototype implementation on realistic data for Denver International Airport.

Nomenclature

n	Number of aircraft
k	Maximum number of CPS shifts
Abbreviations	
ARTCC	Air Route Traffic Control Center
ASP	Arrival Sequencing Program
ATC	Air Traffic Control
ATCT	Air Traffic Control Tower
COMPAS	Computer Oriented Metering Planning and Advisory System
CPS	Constrained Position Shifting
CTAS	Center TRACON Automation System
ETA	Estimated Time of Arrival
FAST	Final Approach Spacing Tool
FCFS	First-Come First-Served order
MPS	Maximum Position Shift
NAS	National Airspace System
TMA	Traffic Management Advisor
TRACON	Terminal Radar Approach Control

[*]Ph.D. Candidate, Department of Aeronautics and Astronautics, Stanford University, Stanford, CA 94305-4035. `hamsa@stanford.edu`. AIAA Student Member.

[†]Ph.D. Candidate, Department of Industrial Engineering and Operations Research, University of California, Berkeley, CA 94720. `chandran@ieor.berkeley.edu`.

I. Introduction

The Air Traffic Control (ATC) system has the responsibility of maintaining a safe and orderly flow of aircraft in the National Airspace System (NAS) of the United States. An important part of this responsibility is the planning of airport operations, such as the arrival and departure of aircraft. In this paper, we present a procedure for computing safe and efficient arrival schedules while taking into account several key operational constraints.

Recent work on the statistical analysis of aircraft arrivals at several major airports in the United States has shown that the distribution of times between estimated arrival times of successive aircraft (estimated when the aircraft are 100 miles from their final destinations) is nearly-exponential in character.[1] This observation is of interest to researchers in air traffic management, because an arrival process with exponentially distributed inter-arrival times is purely random, in the sense that the probability distribution of the time until the next arrival is independent of the time at which the last arrival occurred. This suggests that the combination of various en route air traffic control actions, delays in the NAS, and the different airline schedules result in an arrival stream that is very close to random. It also implies that significant work needs to be done to convert this random arrival stream into an orderly arrival flow into the airport; a large part of this burden lies with the TRACON and en route traffic management controllers who plan the flow of aircraft into the airport. These air traffic controllers typically have a short time horizon (of about 45 minutes) in which to first assign, and then perform the necessary control actions to deliver, an aircraft to a particular position in the landing sequence.[2]

A chief focus of the many efforts to alleviate air traffic controller workload has been the development of automated decision-support tools for controllers.[3] The Federal Aviation Administration (FAA) has certified and deployed several software systems, such as the Traffic Management Advisor (TMA), the Descent Advisor (DA) and the Final Approach Spacing Tool (FAST), as part of the the Center TRACON Automation System (CTAS)[4] to help air traffic controllers sequence and space arriving aircraft.

The primary constraint that air traffic controllers need to ensure in an arrival sequence is that the inter-arrival spacings are more than the FAA-specified minimums. For reasons of safety, it is necessary that an arriving aircraft does not face interference from the wake-vortex of the aircraft landing in front of it. The intensity and risk of the wake vortex depends on the sizes of both the leading and trailing aircraft. Therefore, the required time interval between two landings also depends on the sizes of the two aircraft. For example, a small aircraft following a large aircraft needs greater separation than that required by a large aircraft following another large aircraft.

The most common approach to sequencing aircraft has been to maintain the First-Come-First-Served (FCFS) order (Odoni et al.[5]). In an FCFS schedule, aircraft land in order of their scheduled arrival times at the runway, and air traffic controllers only enforce the minimum separation requirements. There are two key advantages to the FCFS sequence and landing times: (i) the FCFS schedule is relatively easy to implement and promotes safety by reducing controller workload, and (ii) the FCFS order maintains a sense of fairness, since aircraft simply land in the order in which they arrive at the runway; the FCFS order also minimizes the standard deviation of delays of the aircraft.[2]

However, a drawback of the FCFS sequence of landings is that it may lead to reduced runway throughput due to large spacing requirements.[5] For example, a sequence of 10 alternating large and small aircraft will require greater spacings (and will therefore take more time to land overall) than one where 5 small aircraft are followed by 5 large aircraft. Air traffic controllers would like to complete landing a sequence of aircraft as quickly as possible, since the continued presence of the aircraft in the sky contributes to congestion and controller workload, and increases the associated risks. Low runway throughput leads to congestion around an airport and subsequent delays, compromising both safety and efficiency. This provides an incentive to deviate from the FCFS sequence to achieve sequences that lead to maximum runway throughput. However, the terminal area is an extremely dynamic environment, and re-sequencing aircraft increases the workload of controllers.[6] Due to limited flexibility, it might not be possible for air traffic controllers to implement an optimal sequence that deviates significantly from the FCFS order. However, the air traffic controllers do have a certain degree of flexibility, and can quite easily shift an aircraft in the sequence by a small number of positions. This is the basic motivation for Constrained Position Shifting (CPS) methods.

CPS, first proposed by Dear,[7] stipulates that an aircraft may be moved up to a specified maximum number of positions from its FCFS order. We denote the maximum number of position shifts allowed (also referred to as MPS) by k, and the resulting environment as a k-CPS scenario. For example, in 2-CPS, an aircraft that is in the 8th position in the FCFS order can be placed at the 6th, 7th, 8th,9th, or 10th position in the new order. An additional advantage of using CPS is that it maintains some sense of equity among the aircraft by not deviating too much from the FCFS order. This restricted deviation from the FCFS order therefore helps maintain a sense of fairness in the perception of the airlines that operate the aircraft.

Most previous work on optimizing runway schedules deal with the "static" case, i.e., the set of aircraft for which the optimal sequence is desired is fixed (as opposed to the dynamic case where the set of aircraft and optimal sequences change over time as the situation evolves). The static case is applicable when arrivals to an airport are batched over time, and the optimal sequence for each batch is computed separately. Since the set of aircraft is known a-priori in the static case, maximizing runway throughput is equivalent to minimizing the *makespan* (landing time of the last aircraft in the sequence). In most practical implementations, aircraft are scheduled in batches,[2] and it is hence useful to be able to solve the static problem.

Beginning with the early work of Dear,[7] there have been several attempts to develop efficient algorithms and implementations that deliver optimal k-CPS sequences. Dear[7] solved the problem by enumerating all possible sequences to find an optimal sequences, which is impractical for even a modest number of aircraft. Dear and Sherif[8] proposed a heuristic based on CPS for scheduling a single runway. Neumann and Erzberger[2] conducted an extensive investigation into various techniques for sequencing and spacing arrivals. Their study included an enumerative technique for computing the sequence which minimized the makespan, subject to a single position shift (1-CPS) constraint. They proposed a heuristic method based on their 1-CPS algorithm for taking into account operational constraints such as the earliest possible arrival times of the aircraft, and restrictions on overtaking.

Psaraftis[9] proposed a dynamic programming algorithm for single runway scheduling of which CPS was a special case. The complexity of his algorithm is $O(n(n+1)^m)$, where n is the number of aircraft and m is the number of aircraft types. However, his approach assumes that there are no restrictions on the possible landing times of aircraft, *i.e.*, that an aircraft can land at any time. Venkatakrishnan et al.[6] proposed a heuristic that incorporated possible arrival windows into Psaraftis' formulation.

Trivizas[10] proposed a dynamic programming algorithm requiring a sophisticated implementation, which computes an optimal k-CPS landing sequence with complexity $O(n2^k)$, where n is the number of aircraft. However, this approach cannot handle precedence relations between aircraft, for example, constraints of the form "If aircraft i lands before aircraft j in the FCFS order, then i must land before j in the final sequence". Such constraints are important for two reasons:

1. In almost all current ATC automation systems (for example, COMPAS in Frankfurt,[11] MAESTRO in Paris[12] and CTAS in Dallas Fort Worth[4]), very limited overtaking is allowed.
2. Airlines themselves may have preferences in precedence relations, arising from their banking strategies.

The automation systems mentioned above use versions of CPS, but resort to heuristics to handle operational constraints like time windows and precedence relations. Our approach is designed to solve these problems.

More recently, there have been several attempts at using optimization techniques such as integer programming to the problem of arrival sequencing. The problem of finding the optimal landing sequence when the spacing between arrivals depends on the aircraft type is $\mathcal{NP}$-hard, making it unlikely that efficient algorithms exist for solving this problem.[13] Beasley et al.[14] attempt to tackle this problem by using optimization tools for solving mixed-integer linear programs (MILPs). Integer programming methods are flexible enough to account for most operational constraints such as time windows, precedence and CPS, but the solution times of integer programs for such problems can be large and can vary significantly from one instance to another, making it unattractive for deployment as a real-time decision support tool.

Several authors, such as Bayen et al.,[15] have approached the problem by assuming that the required spacing between landings is the same between any two aircraft, independent of their types. We do not make this assumption because we would like to take advantage of the differences in spacing requirements to reduce the makespan of the sequence. Also, if we were to assume a single spacing for a mix of aircraft, safety would mandate that we choose the largest among the possible spacings for all types of aircraft – this could be overly conservative, and quite inefficient.

In this paper, we develop a new procedure for computing an optimal k-CPS sequence to minimize the makespan in the static case. The key contributions of this paper are:

1. We bridge the gap between the mostly heuristic or enumeration-based CPS methods and the optimization-based methods to come up with optimal solutions that take advantage of the structure of CPS sequences.
2. Our procedure can simultaneously handle the operational constraints of CPS, precedence relationships, and restrictions on landing times of aircraft, which have not been addressed in past work.
3. The procedure is easy to implement and requires no special-purpose optimization software. The running

time of our algorithm is fast and predictable (it scales linearly in the number of aircraft) and can hence be embedded within a real-time decision support tool such as CTAS.

We present our algorithmic framework in Section III, and in Section IV, we describe an implementation of our approach to realistic scenarios drawn from data of the arrival traffic flow into Denver. Using these scenarios, we demonstrate that the procedure we develop is flexible, can compute optimal solutions very quickly, and scales slowly and reliably with increase in number of aircraft. Finally, in Section V, we discuss the extensions of our methodology to more general airport operation scheduling problems, such as, multiple parallel runway scheduling, the simultaneous scheduling of arrival and departure sequences, incorporating airline preferences into scheduling, and the scheduling of surface operations at an airport.

II. Problem definition

Given a set of aircraft, we wish to determine the sequence that minimizes makespan (landing time of the last aircraft) subject to the operational constraints of CPS, precedence, minimum separation requirement, and possible arrival time windows for aircraft.

The FCFS order and MPS parameter

Given times of arrival of the aircraft at the boundary of the ARTCC, Trajectory Synthesizers[2] can compute the average time an aircraft would take to reach the runway, if there was no interference from other aircraft. This is known as the Estimated Time of Arrival (ETA) of the aircraft. The order of the ETAs gives us the FCFS sequence of landings. The MPS parameter, denoted by k, is typically small (1, 2, or 3).[17]

Arrival time windows

Given the time at which the aircraft crosses the Center boundary, there is an earliest time at which the aircraft can possibly land (here, we can account for any possible speed-ups by the aircraft), as well as a latest possible time of landing (determined by possible fuel constraints or the maximum delays that may be acceptable for an aircraft). The earliest and latest arrival times of aircraft i are denoted by $E(i)$ and $L(i)$ respectively. In general, however, it is not necessary that the possible landing times for an aircraft belong to a connected set; our approach is capable of handling situations in which an aircraft's landing time could lie in any one of a number of time intervals. We consider the single-interval case only to simplify the technical discussion of the algorithm.

Minimum aircraft separation

The FAA establishes minimum spacing requirements between landing aircraft to prevent the danger of wake turbulence. An aircraft faces the risk of instability if it interacts with the wake-vortex of the aircraft landing in front of it. To prevent this, there must be a minimum spacing between the aircraft, which depends on the size of the leading and trailing aircraft. We define the minimum time-separation matrix by S, where the element s_{ab} is the minimum required time between arrivals, if the first aircraft to land (leading aircraft) belongs to class a, and the second aircraft to land (trailing aircraft) belongs to class b.

The FAA[16] divides aircraft into three weight classes, based on the maximum take-off weight capacity. These classes are

1. Heavy: Aircraft capable of takeoff weights of more than 255,000 lbs.
2. Large: Aircraft of more than 41,000 lbs, maximum takeoff weight, up to 255,000 lbs.
3. Small: Aircraft of 41,000 lbs or less maximum takeoff weight.

Heavy aircraft include the B747, B767, A300, DC10 and DC8; large aircraft include the B757, DC9, ATR42; and small aircraft include single piston-engine aircraft and small turboprops like the Beach 99.

Using this classification of aircraft, the FAA specifies separation requirements during IFR approaches. These separation requirements can be used to determine the minimum separation required between landings, assuming a 5 nmi final approach path (de Neufville and Odoni[17]). The matrix of minimum time separations is given in Table 1.

An important assumption that we make in this paper is that the separation requirements satisfy the *triangle inequality*, that is

$$s_{ik} \leq s_{ij} + s_{jk}, \text{ for all aircraft types } i, j, k \tag{1}$$

The triangle inequality ensures that enforcing the minimum spacing between only successive aircraft in a sequence ensures that the minimum spacing requirement is met for all pairs of aircraft. It is easy to see that

	Trailing Aircraft		
Leading Aircraft	Heavy	Large	Small
Heavy	96	157	196
Large	60	69	131
Small	60	69	82

Table 1. Minimum time separations (in seconds) between landings.[17]

the FAA-specified minimum separation standards for inter-arrival times, as shown in Table 1, satisfy the triangle inequality. It is possible to extend the techniques presented in this paper to situations in which the triangle inequality is not satisfied, but these cases are beyond the scope of this paper.

Precedence constraints

Finally, we consider precedence constraints on the landing sequence. One source of these constraints is the different jet routes in which the aircraft arrive at the airport. Air traffic controllers cannot typically allow aircraft within a jet route to overtake each other (Neuman and Erzberger[2]). This restriction results in precedence constraints between the aircraft along the same jet route. Another source of such constraints would be from the airlines themselves, as we will briefly discuss in Section V-C. We represent precedence relations by an $n \times n$ matrix $\{p_{ij}\}$, such that element $p_{ij} = 1$ if aircraft i *must* land before aircraft j, and $p_{ij} = 0$ otherwise.

Problem statement

Consolidating our objective and constraints, we can pose the following problem:

Given n aircraft indexed $1, \cdots, n$, earliest and latest arrival times $E(i)$ and $L(i)$ for each aircraft i, separation matrix S, precedence matrix $\{p_{ij}\}$, and the maximum number of position shifts k, compute the k-CPS sequence and corresponding landing times that minimize the makespan of the sequence.

For simplicity, we assume that the aircraft are labeled $(1, 2, \cdots, n)$, according to their position in the FCFS sequence.

III. Algorithm

Our solution approach is to pose the problem as a modified shortest path problem on a network, which is then solved by dynamic programming. The procedure for generating the network is described in the following section.

A. Network generation

The network consists of n *stages* $\{1, \cdots, n\}$, where each stage corresponds to an aircraft position in the final sequence. A node in stage p of the network represents a subsequence of aircraft of length $\min\{2k+1, n-p+1\}$, where k is the maximum position shift. For example, for $n = 6$ and $k = 1$, the nodes in stages $1, \cdots, 4$ represent all possible sequences of length $2k + 1 = 3$ starting at that stage. Stage 5 contains a node for every possible aircraft sequence of length 2 starting at position 5, while stage 6 contains a node for every possible sequence of length 1 starting at position 6. This network, shown in Fig. 2, is obtained by finding all sequence combinations of possible aircraft assignments to each position in the sequence given in Fig. 1. One property of the nodes that we will use is that each aircraft in a node's subsequence is unique. For convenience, we refer to the first aircraft in a node's sequence as the *initial aircraft* of that node.

Position	1	2	3	4	5	6
Possible aircraft assignments	1	1	2	3	4	5
	2	2	3	4	5	6
		3	4	5	6	

Figure 1. Possible aircraft assignments for $n = 6$, $k = 1$.

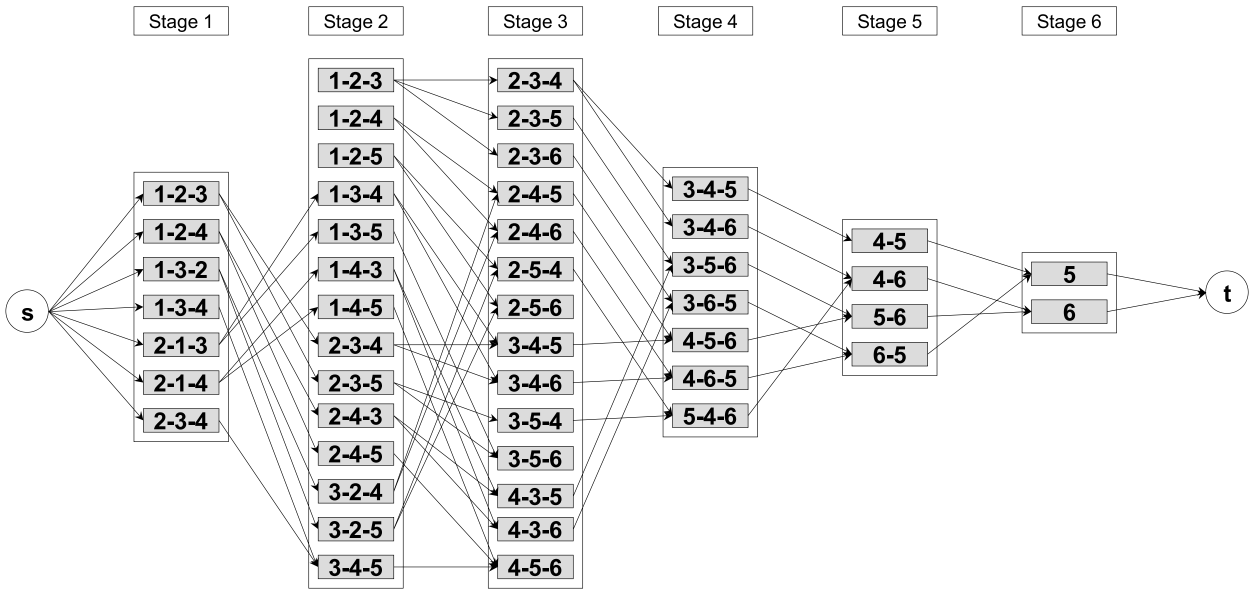

Figure 2. Network for $n = 6$, $k = 1$.

We then introduce two nodes, a source node s and a terminal node t that represent the beginning and end of the sequencing process respectively. We add arcs from s to each node in stage 1, and from each node in stage n to t. Finally, for each node in stage p, we draw directed arcs to all the nodes in stage $p + 1$ that can follow it. For example, a sequence (1–2–3) starting in stage 1 can be followed by the sequences (2–3–4) or (2–3–5) starting in stage 2. This results in a network where every directed path from s to t represents a possible sequence. For example, the path $s \to$ (2–1–3) $\to$ (1–3–4) $\to$ (3–4–6) $\to$ (4–6–5) $\to$ (6–5) $\to$ (5) $\to t$ represents the sequence 2–1–3–4–6–5.

Lemma 1 *Every possible k-CPS subsequence of length $2k + 1$ or less is contained in some node of the network.*

Proof: By construction, every subsequence of length $2k + 1$ or less starting in position p is explicitly enumerated in stage p. ■

Corollary 1 *Every k-CPS sequence can be represented by an s-t path in the network.*

Lemma 2 *Every s-t path in the network represents a feasible k-CPS sequence.*

Proof: First, we observe that every s-t path will consist of exactly n nodes (other than s and t) since each arc in the path moves forward by one stage. Given a path in the network, the corresponding aircraft sequence is obtained taking the initial aircraft of a node belonging to stage p and assigning it to position p in the sequence. Thus, position 1 in the sequence is the initial aircraft of the first node in the path, position 2 in the sequence is the initial aircraft of the second node in the path, and so on. Since there are n aircraft and the path is of length n, this procedure will yield a feasible sequence as long as we assign a unique aircraft to each position, i.e., as long as the initial aircraft of each node in the path is different.

We now prove this result by contradiction. Suppose there exists an s-t path in the network containing two nodes such that the initial aircraft of the two nodes is the same. Let one of these nodes be in stage p_1 and the other be in stage p_2 where $p_1 < p_2$. We have already established that $p_1 \neq p_2$ since each node in the path belongs to a different stage. Therefore, the aircraft appears in position p_1 and p_2 in the sequence represented by the path. The network is constructed using the fact that any aircraft can appear in at most $2k + 1$ positions, so p_2 is at most $2k + 1$ positions from p_1. But every subsequence of length $2k + 1$ or less is captured in some node along the path, so there exists a node in the path in whose subsequence the same aircraft appears in more than one position, which is not allowed to occur while generating the network. This contradiction implies that there cannot exist an s-t path containing two nodes with the same initial aircraft.

Therefore, any s-t path represents a sequence of n distinct aircraft, where each aircraft appears in a position that belongs to one of at most $2k + 1$ possible position assignments for that aircraft. ■

We now observe that there are some nodes in the network that cannot be part of any path from s to t. For example, node (1–2–4) belonging to stage 2 in Fig. 2 has no arcs leading into it, and so cannot belong to any path. Such nodes can be eliminated to simplify the network and reduce its size. For a node to belong to some s-t path, there should be a path from s to that node, and a path from that node to t. Finding all nodes in the network that can be reached from s can be done by a simple graph search algorithm (starting at s, traverse the network from left to right along the arcs and mark all nodes that are visited). Similarly, finding all nodes in the network from which t is reachable can be done by another application of a reverse graph search (starting at t, traverse the network from right to left along the reversed arc direction and mark all nodes that are visited). The set of nodes that are visited in both directions are retained in the network, while all other nodes (and the arcs leading into or out of them) are discarded. We refer to this process as *pruning* the network. The pruned network for the $n = 6$, $k = 1$ scenario is shown in Fig. 3. Note that the pruned network is significantly smaller than the original network.

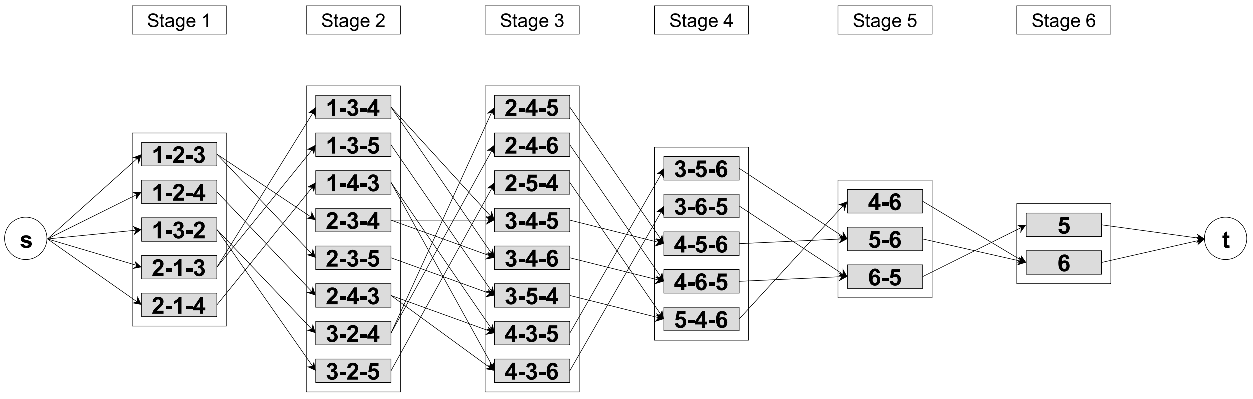

Figure 3. Pruned network for $n = 6$, $k = 1$.

B. Dynamic programming recursion

The main idea behind the approach is that we wish to find an s-t path of shortest "length" in the network, which will translate to a sequence with the smallest makespan. We now show that this can be done using dynamic programming.

Given two nodes i and j, the arc connecting them (if it exists) is denoted by (i, j). The set of predecessor nodes of j is denoted by $P(j)$. In other words, $P(j)$ is the set of all nodes i such that arc (i, j) exists. Let $e(i)$ denote the earliest time that the sequence of node i can begin, which is the earliest arrival time $E(.)$ of the initial aircraft of node i. Similarly, let $\ell(i)$ denote the latest arrival time $L(.)$ of the initial aircraft of node i. Each arc (i, j) in the network is associated with a "distance" d_{ij} which is the minimum separation between the initial aircraft of node i and that of node j, if they were to land in that order. This separation is determined by the weight classes of the two initial aircraft. Arcs that lead into the sink and out of the source have zero distance associated with them.

Let $T(i)$ be the earliest time that the sequence corresponding to node i can possibly begin, in a sequence starting at node s and ending in node i. We wish to find $T(t)$, the earliest time that the entire sequence can be completed, which is equal to the makespan. The values of $T(.)$ can be computed by the following dynamic programming recursion.

$$T(j) = \max\left\{e(j), \min_{i\in P(j):\ T(i)\leq \ell(i)} (T(i) + d_{ij})\right\} \tag{2}$$

Note that if the set $i \in P(j) :\ T(i) \leq \ell(i)$ is empty, $T(j)$ is defined to be ∞.

Lemma 3 *The recursion in Equation 2 correctly computes the values of $T(j)$ for all nodes j.*

Proof: $T(j) \geq e(j)$ since a sequence cannot start before the earliest arrival time of the initial aircraft in the sequence.

If $T(i) > \ell(i)$ for some predecessor node i, it implies that the sequence of node i cannot possibly start within the allowable time window $[e(i), \ell(i)]$; therefore if this node is used, the initial aircraft of node i cannot

```
procedure FindMakespan($V, A, d$):
    begin
        $T(s) \leftarrow 0$, $\ell(s) \leftarrow 0$;
        for each $p = 1, \cdots, n$ do
            for each node $j$ in stage $p$ do
                $T(j) = \max\left\{e(j), \min_{i \in P(j):\ T(i) \leq \ell(i)} (T(i) + d_{ij})\right\}$;
                $\text{pred}(j) = \arg\min_{i \in P(j):\ T(i) \leq \ell(i)} (T(i) + d_{ij})$;
        $T(t) = \min_{i \in P(t):\ T(i) \leq \ell(i)} T(i)$;
        $\text{pred}(t) = \arg\min_{i \in P(t):\ T(i) \leq \ell(i)} T(i)$;
    end
```

Figure 4. Algorithm for computing the minimum makespan.

be landed between its earliest and latest arrival times, and hence this node cannot be part of any feasible s-t path sequence. Therefore, all predecessors with $T(i) > \ell(i)$ can be ignored while finding the predecessor of node i.

It is not possible for $T(j)$ to be strictly less than $T(i) + d_{ij}$ for all predecessors i since that would violate the separation requirement between the initial aircraft of i and j. Therefore, $T(j) \geq T(h) + d_{hj}$ for at least one of the predecessors $h \in P(j)$ such that $P(j) \leq \ell(j)$. Since $T(j) \geq T(h) + d_{hj}$ for at least one feasible predecessor h, it is certainly greater than the minimum of $T(i) + d_{ij}$ over all feasible predecessors i.

We have shown so far that $T(j) \geq e(j)$ and $T(j) \geq \min_{i \in P(j):\ T(i) \leq \ell(i)} (T(i) + d_{ij})$. To complete the proof, we have to show that at least one of the above inequalities holds as an equality so that $T(j) = \max\left\{e(j), \min_{i \in P(j):\ T(i) \leq \ell(i)} (T(i) + d_{ij})\right\}$. We now prove the rest by contradiction. Suppose $T(j) > e(j)$ *and* $T(j) > \min_{i \in P(j):\ T(i) \leq \ell(i)} (T(i) + d_{ij})$. Then, it is possible to reduce the value of $T(j)$ by some sufficiently small quantity while still maintaining a feasible solution, which contradicts the minimality of $T(j)$.

Therefore, $T(j) \geq e(j)$ *and* $T(j) \geq \min_{i \in P(j):\ T(i) \leq \ell(i)} (T(i) + d_{ij})$ *and* at least one of the two inequalities holds as an equality. ■

Using the boundary condition $T(s)$ = current time, we can apply the dynamic programming recursion to nodes in stages $1, \cdots, n$ and finally to node t to generate the values of $T(.)$.

Corollary 2 *The k-CPS scheduling problem is infeasible if and only if* $T(t) = \infty$.

The predecessor (i.e., value of i) that minimizes the quantity $\min_{i \in P(j):\ T(i) \leq \ell(i)} (T(i) + d_{ij})$ is denoted by prev(j), and is used to construct construct the optimal sequence. Starting from node t, the recursive sequence of prev(.) nodes recovers the sequence corresponding to the optimal value $T(t)$. The landing time of an aircraft in the sequence defined by the path is the value of $T(.)$ of the node in the path for which that aircraft is the initial aircraft. The pseudocode for the algorithm is given in Fig. 4.

Initialization

In the case where aircraft are being scheduled in batches (as is done in practice), the landing times of the current batch of aircraft being scheduled must take reflect the required separation between the first aircraft of the current batch and the last aircraft of the previous batch. To enforce this constraint, we update the earliest arrival times of aircraft in the current batch to take into account the required separation. If the last aircraft i of the previous batch landed at time t, the earliest arrival time of aircraft j in the current batch is updated to $\max\{E(j), t + d_{ij}\}$, where d_{ij} is the minimum separation between aircraft i and j. The procedure then proceeds as before; $T(s)$ is set to the current time, and the dynamic programming recursion is performed with the new values of earliest arrival times.

C. Incorporating precedence constraints

Precedence constraints are conditions that specify the relative landing order of two aircraft. There are two possibilities, depending on whether or not the precedence constraints require that the FCFS order be reversed.

Case I: $a < b$ and $p_{ab} = 1$, i.e., aircraft a lands before b in the FCFS sequence and we require a to land before b in the optimal makespan sequence.

Let $f(b)$ be the position of aircraft b in the final (optimal makespan) sequence, and $f(a)$ be the position of aircraft a in the final sequence. Our precedence constraint implies that we need to preclude paths that result in $f(a) > f(b)$.

Lemma 4 *Suppose there exists a path in the network such that $f(a) > f(b)$. Then, the path contains at least one node such that b appears before a in that node's subsequence.*

Proof: Since each aircraft can shift at most k positions, $f(a) \leq a + k$ and $f(b) \geq b - k$. So $f(a) - f(b) \leq a - b + 2k \leq 2k - 1$ (because $a - b \leq -1$ given that $a < b$). Since $f(a)$ and $f(b)$ are within $2k+1$ of each other and every sequence of length at most $2k + 1$ is contained in some node along the path, the path contains some node whose sequence has $f(a) > f(b)$ and violates the precedence constraint. ■

Clearly, nodes that contain subsequences that violate the precedence constraints should be removed from the network since we do not want these nodes to belong to any s-t path. The above lemma shows that removing nodes that violate precedence constraints is not only necessary, but also sufficient for eliminating all s-t paths that violate precedence constraints. This yields the following procedure: in the presence of precedence constraints where $a < b$ and we require $f(a) < f(b)$, remove all nodes where the precedence is violated, and solve the problem on the resulting network.

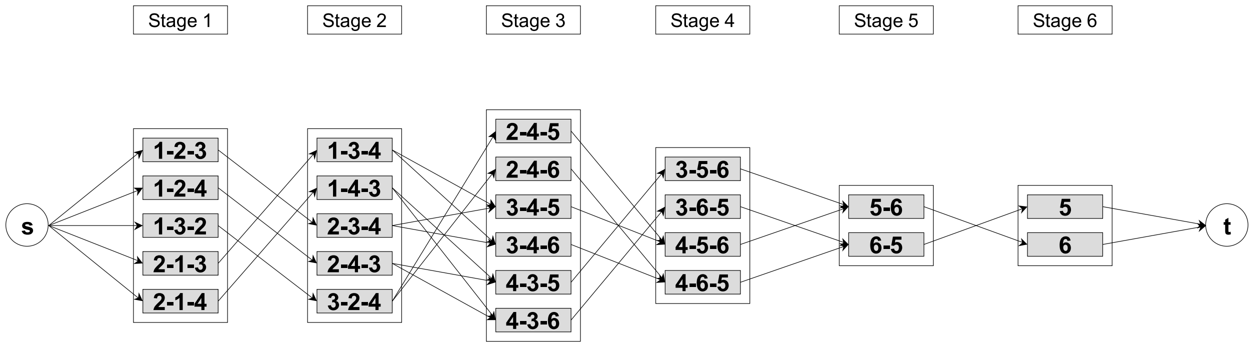

Figure 5. Network with precedence constraint that aircraft 4 lands before aircraft 5, for $n = 6$, $k = 1$.

We illustrate this procedure on the example network we have seen first in Fig. 2, and then in Fig. 3. Suppose our precedence constraint specifies that aircraft 4 necessarily land after aircraft 5. Then, we remove all nodes in which aircraft 5 precedes aircraft 4 – this eliminates the nodes (2–5–4), (3–5–4), and (5–4–6), and the arcs leading into and out of them. We then remove all the nodes that are not reachable from s or t in this network. The resultant network with the precedence constraints incorporated is shown in Fig. 5.

Case II: $a < b$ and $p_{ba} = 1$, i.e., a lands before b in the FCFS sequence but we require b to land before a in the optimal makespan sequence.

In this case, we need to eliminate paths that result in $f(a) < f(b)$. Since the earliest position that b can land is $b - k$, the earliest time that a can land given that it lands after b is $b - k + 1$. Similarly, the latest position that a can land is $a + k$, meaning that the latest time that b can land is $a + k - 1$. Any node that contains a sequence that violates these two constraints, i.e., has a in a position that $\leq b - k$ or has b in a position $\geq a + k$ should be removed from the network since they cannot belong to a feasible path. We refer to the network obtained after removing such nodes as the *position-constrained* network.

Lemma 5 *Suppose there exists a path in the position-constrained network such that $f(a) < f(b)$. Then, the path contains at least one node such that a appears before b in that node's subsequence.*

Proof: In the position-constrained network, $b - k + 1 \leq f(a) \leq a + k$ and $b - k \leq f(b) \leq a + k - 1$. So $f(b) - f(a) \leq a - b + 2k - 2 \leq 2k - 3$ (because $a - b \leq -1$ given that $a < b$). Since $f(a)$ and $f(b)$ are within $2k + 1$ of each other and every sequence of length at most $2k + 1$ is contained in some node along the path, the path contains some node whose sequence has $f(a) < f(b)$ and violates the precedence constraint. ■

This yields the following procedure: in the presence of precedence constraints where $a < b$ and we require $f(a) > f(b)$, we first remove all nodes where the position constraint is violated, giving the position-constrained network. Then, we remove all nodes that violate the precedence constraints, and solve the problem on the resulting network.

D. Complexity

Proposition 1 *The complexity of the algorithm for finding the minimum makespan for n aircraft and maximum position shift of k is $O(n(2k+1)^{(2k+2)})$.*

Proof: The nodes in each stage of the network are generated by all combinations of length $2k+1$, where each position in the sequence has at most $2k+1$ possible aircraft. The number of nodes in each stage is therefore at most $(2k+1)^{(2k+1)}$; since there are n stages, the total number of nodes in the network is bounded by $n(2k+1)^{(2k+1)}$. Each node can have at most $2k+1$ predecessors since the sequence of a node differs from the sequence of its predecessor only in the first and last position, therefore the number of arcs is $O(n(2k+1)^{(2k+2)})$. Pruning the network requires looking at each arc at most twice—once during the forward pass and once during the backward pass. The dynamic programming recursion examines each predecessor of a node at most once, so total complexity is equal to the number of arcs in the network, which is $O(n(2k+1)^{(2k+2)})$. ■

We note here that the complexity expression is a conservative upper bound; in practice, the number of arcs in the pruned network is significantly smaller than that predicted by the complexity expression. Table 2 shows the upper bound and the actual number of arcs in the pruned network for 50 aircraft.

k	Number of Arcs	
	$50 \times (2k+1)^{(2k+2)}$	Pruned network
1	4,050	617
2	781,250	33,943
3	288,240,050	4,104,950

Table 2. Actual number of arcs in pruned network and upper bound for 50 aircraft.

In the presence of precedence constraints, pre-processing the network to eliminate nodes that violate precedence constraints takes $O((k+1)^2)$ work per node since we would need to examine every pair of aircraft in the subsequence in the worst case. The complexity for preprocessing all the nodes would therefore be $O(n(2k+1)^{(2k+3)})$ which dominates the running time of pruning the network and performing the dynamic programming recursion.

While the is exponential in k, it is of little consequence, since k is typically small (at most 3 in practice).[17] The linear growth in n is useful since increasing the number of aircraft does not pose much of a computational burden.

E. Implementation

So far, we did not include the work done in generating the network into the complexity analysis. Since the basic network remains the same for given values of n and k, we generate, prune and store the network offline, and recall the network when required. Therefore, this needs to be done only once. In practice, n is typically no more than 50 aircraft (since we are dealing with a short scheduling time horizon of no more than 30 minutes) and k is no more than 3, the number of networks that need to be generated and stored is small (no more than 150) which requires very little disk space.

Given an instance of the problem for n and k, our implementation reads the appropriate network from a file, does the appropriate preprocessing to account for precedence constraints, and then runs the dynamic programming algorithm on the resulting the network. Our computational experience was that reading the file containing the nodes and arcs dominates the computation time.

IV. A real-world scenario: Denver International Airport

We now present an implementation of our approach on sample data based on arrival traffic at Denver International Airport. We chose Denver for several reasons. Denver airport is sufficiently separated from other major airports to discern traffic patterns easily (Fig. 6). The airport is also close to the middle of Denver Center airspace (ZDV). Comprehensive statistics on scheduled arrivals at Denver are readily available from the Bureau of Transportation Statistics[18] (Fig. 7), and the average times taken by aircraft along different jet routes in ZDV have also been published by Neuman and Erzberger.[2]

The Denver ARTCC airspace is shown in Fig. 9, with the jet routes that carry arrival traffic. The traffic at Denver is routed through one of 8 arrival gates: RAMMS, TOMSN (NW arrivals), LANDR, SAYGE (NE

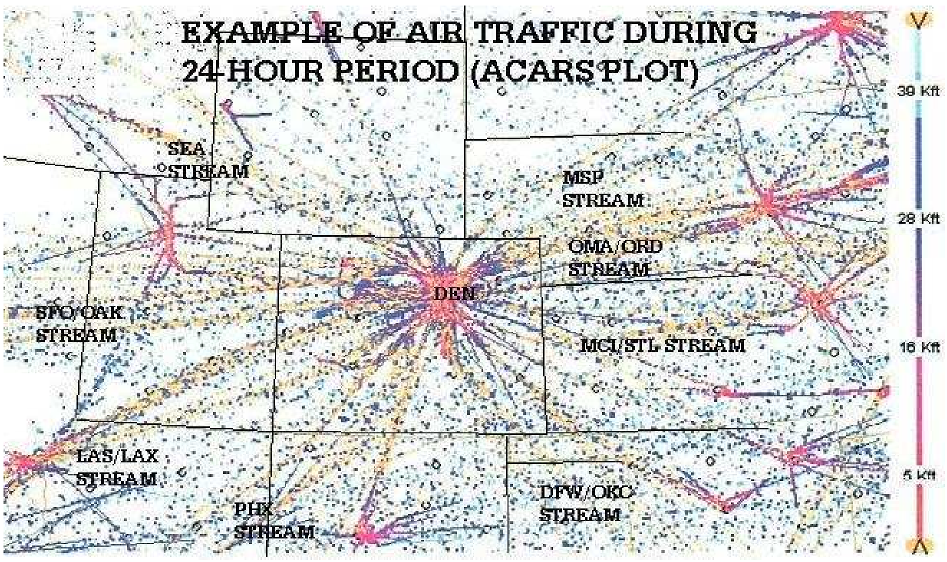

Figure 6. Traffic patterns around Denver International Airport (right).[19]

arrivals), DANDD, QUAIL (SE arrivals), LARKS and POWDER (SW arrivals). Once they have passed through the arrival gates, aircraft enter the Denver TRACON airspace. The TRACON airspace is defined by a 42 nmi. radius around the airport, and is shown in Fig. 8. It is possible to obtain real time data on the demand at any of the arrival fixes (Fig. 10[20]). At the arrival gates, aircraft are handed off to one of the *arrival control* positions in the TRACON. We consider the case in which all arrivals land either on Runway 25 or Runway 26 ("Land West all"). In such scenarios, traffic from the SE and SW gates land on Runway 25, and traffic from the NE and NW gates land on Runway 26. For simplicity, since we are currently interested in the single runway case, we only considered traffic in the Northern half-plane, and assume that all the traffic considered lands on Runway 26. It is the arrival control position's responsibility to form a desirable sequence of aircraft, which are then handed off to a *final control* position in the TRACON. The final control positions have the responsibilities of spacing aircraft for final approach, and of merging streams of aircraft from different fixes, if necessary. Near the beginning of the final approach, control over an aircraft is handed over to the air traffic control tower.

The Arrival Sequencing Program (Neuman and Erzberger[2]) attempts to find the most efficient landing sequence and the optimal landing times, subject to spacing constraints; the input is the times when the aircraft enter the Denver ARTCC. Given these times, one can compute the FCFS order at the airport, using the average time spent by aircraft on the different jet routes. This data, derived by Neuman and Erzberger,[2] is shown for the Northern arrivals in Table 3. These times were used to compute the FCFS order of arrivals at the runway, and the Estimated Times of Arrival (ETAs).

	Jet Route No.	**Time within ZDV**	**Gates**	
1	J163	42.30	TOMSN	Through NW Gates
2	J156	45.45	TOMSN	
3	J170	45.00	RAMMS	
4	J24	47.78	TOMSN	
5	J136	45.00	RAMMS	
6	J114	41.43	LANDR	Through NE Gates
7	J10	45.00	SAYGE	
8	J157	45.00	LANDR	
9	J60	45.00	SAYGE	

Table 3. Jet route traversal times.[2]

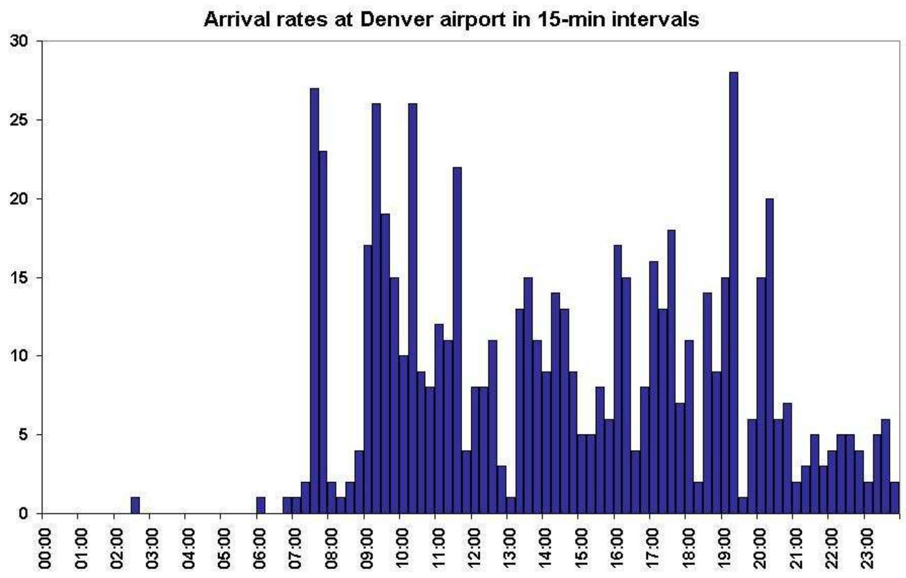

Figure 7. Histogram of arrival rates at Denver International Airport on July 6, 2005.

A. Data

In order to determine the number of aircraft arriving through each of the gates, we used the statistical distribution of flights through the different gates.[18] The distribution that we derived from this data is shown in Fig. 11. For each of the directions of arrival, we divided the traffic equally among all the corresponding jet routes. According to Neuman and Erzberger,[2] it is necessary to maintain FCFS order among aircraft in the same jet route. Therefore, we used the jet routes to determine the landing precedence relations. Neuman and Erzberger also investigated the possibility of speeding up aircraft, and the consequent fuel expense, and arrived at the conclusion that it is not economically worthwhile to move the landing time forward by more than a minute. Therefore, we set the earliest arrival time at 1 minute less than the ETA. We set the latest possible arrival time at 60 minutes after the ETA, implying that we would not put an aircraft on hold for more than an hour.

It has been noted that reasonable values of k for CPS might be 1, 2, or 3. For these values, it is possible for air traffic controllers to implement the reordering of aircraft to achieve the optimal sequence that is desired (de Neufville and Odoni[17]). Another benefit of such small k is the *fairness* that is maintained across aircraft. It is known that the FCFS order minimizes the standard deviation of delays, and provides a fair solution, so a CPS that maintains a restricted deviation from the FCFS order minimizes the makespan while still staying close to the a fair solution.

We assumed a Poisson arrival process in generating the sequence of times at which aircraft enter the Denver Center. This is based on the work of Willemain et al.[1] in which they show, based on studies at 9 major U.S. airports, that the inter-arrivals times at airports before the final control actions are executed can be well-modeled by exponential distributions.

B. Computing environment

We conducted all our experiments on a personal computer with 3.2GHz Intel Pentium 4 CPU on a Linux platform and 512 MB RAM. The CPS code was written in C++, while the optimal (no CPS constraint) problem was coded in AMPL[21] and solved using CPLEX[22] (version 9.1).

C. Results

First, we present the result of Monte Carlo simulations for varying mixes of aircraft. In particular, we consider two mixes of aircraft, one a 40% Heavy, 40% Large, and 20% Small mix, and the other a 45% Heavy, 45% Large, and 10% Small mix of aircraft. These are aircraft mixes that one could expect to see

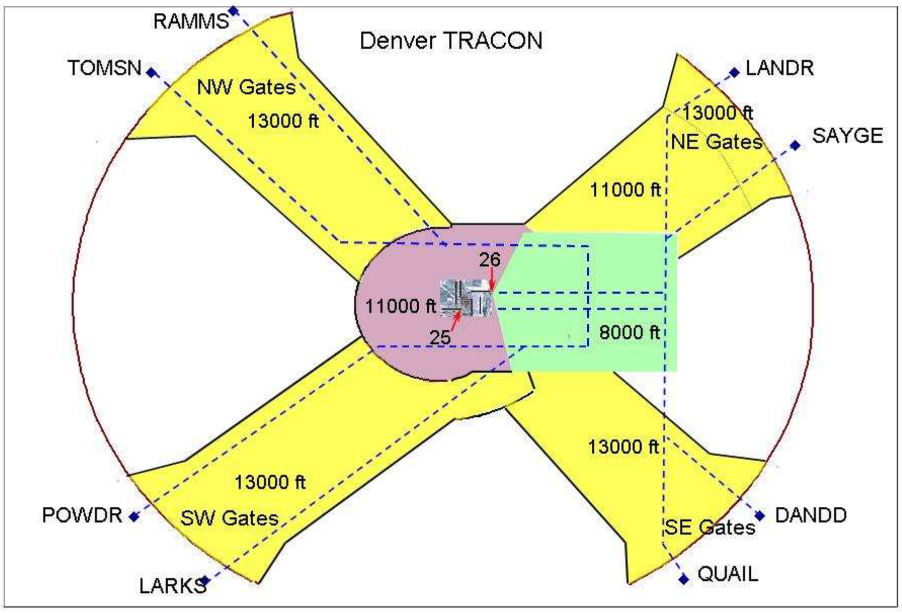

Figure 8. Denver airport approach airspace.

in practice, since most major airports are likely to have more Heavy and Large aircraft operations than Small ones. Intuition would suggest that a homogeneous mix of aircraft (where all aircraft belong to the same weight class) would not benefit from CPS, since all the inter-arrival spacings would be the same. We measure the improvement gained by performing CPS in terms of the difference between the FCFS makespan and the CPS makespan. Fig. 12 shows the gains made by CPS relative to the FCFS makespan for up to 50 aircraft. Each data point in the figure is the average over results from 100 different randomly generated arrival scenarios. The results support our intuition that the gains from CPS are greater when the mix is less homogeneous.

We also present an instance generated from the time-varying histogram of arrival rates at Denver International Airport (Fig. 7). The time-period we consider is between 9:00 AM and 10:30 AM, with a scheduling horizon of 30 min. This implies that we schedule aircraft in batches, half an hour of arrivals at a time. We chose this horizon because aircraft reach the arrival fixes after about 30 mins of flight within the center, and normally the sequencing decisions have to be made prior to the aircraft reaching the arrival fix. In order to demonstrate the behavior of the algoritm for a variety of congestion levels, we artificially reduce the arrival rate in the first half hour by 50%. Therefore, the assumed arrival rates for the three aircraft batches are 22 aircraft/hr between 9:00 AM and 9:30 AM, 34 aircraft/hr between 9:30 AM and 10:00 AM, and 36 aircraft/hr between 10:00 AM and 10:30 AM. A random instance of arrival times was generated using a Poisson process with the appropriate rates, and the jet routes were assigned to them as described in Section A. The resultant random scenario generated 11 aircraft in the first half-hour, 19 aircraft in the second, and 23 aircraft in the third.

The minimum makespan for each of these time periods, for $k = 1, 2, 3$, as well as the optimal solution are shown in Table 4.

Time period	**9:00–9:30 AM**	**9:30–10:00 AM**	**10:00–10:30 AM**
Number of aircraft scheduled	11	19	23
Procedure	**Landing time of last aircraft** (in mins past 9 AM)		
FCFS	74.49	101.84	144.18
1-CPS	74.16	101.51	139.92
2-CPS	73.84	101.34	139.06
3-CPS	73.84	101.22	139.06
Optimal (using CPLEX)	73.84	101.22	139.06

Table 4. Makespan for the different time periods, as well as different values of k, for the scenario shown in Figs. 13–15.

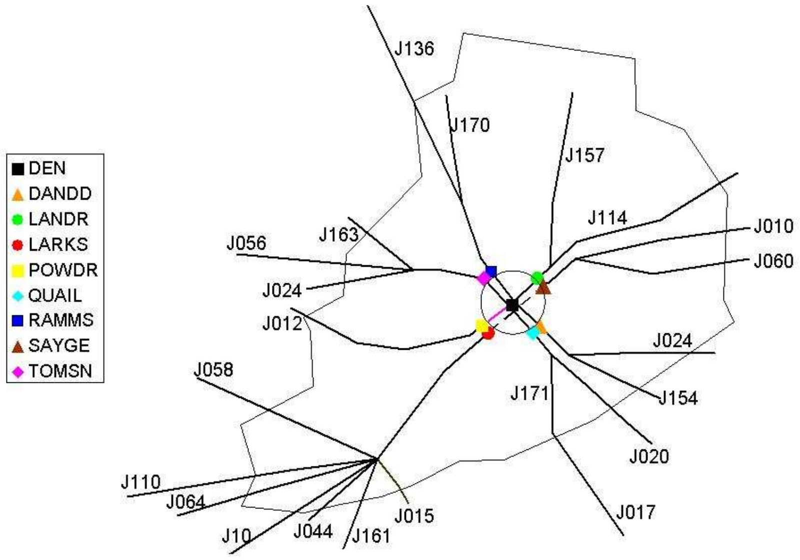

Figure 9. Denver airspace, showing jet routes and arrival gates.

The resultant sequences of this scenario are presented in Figs. 13, 14, and 15 as follows: The horizontal lines at the top of the figures correspond to the different jet routes. The first set of lines represents the jet routes leading to the NE gates (J114, J10, J157 and J60), and the second set represents jet routes leading to the NW gates (J163, J156, J170, J24 and J136). The 'x's on these lines represent the arrival time of an aircraft on that jet route at the boundary of the Denver Center. The sequence along a jet route also depicts our precedence constraints, since we do not allow overtaking along a jet route.

We project the arrivals on all the jet routes down onto a single horizontal line to show the sequence of arrivals at the Center boundary. Using these arrival times, and the travel time along a jet route (shown in Table 3), we can compute the order and times of arrival of aircraft at the runway. This time is shown in the figures with a shift of 40 min. In other words, the times shown below the horizontal dotted line are on a different scale from those above the line, one that is shifted forward by 40 min. While this arrival order gives us the FCFS order of landings, the time between arrivals does not necessarily satisfy the minimum spacing requirements. Therefore, we enforce the minimum spacing requirements, and compute the FCFS sequence of arrivals shown in the figures. The solid black lines correspond to Heavy aircraft, the dashed red lines to Large aircraft, and the dot-dashed green lines to Small aircraft. All the figures are identical in the characteristics we have described so far. Finally, we use the arrivals to compute the 1-CPS, 2-CPS, and 3-CPS solutions, which are shown in Figs. 13, 14, and 15, respectively. This shows the resultant deviations from the FCFS order, depending on the maximum position shift allowed.

The average delays for the given scenario, and the completion time (landing time of the last aircraft) are shown in Table 5. As expected, since we are minimizing the makespan, we can complete landing all the aircraft in less time, if more position shifts are allowed. Even though the makespan does not change between 2-CPS and 3-CPS, the average delay does decrease, and only three aircraft are involved in the sequence change that results in this decrease. When compared to the FCFS sequence, 1-CPS decreases the makespan by 1 min 15.6 sec, 2-CPS and 3-CPS decrease the makespan by 5 min 7.5 sec; 1-CPS decreases average delay by 59.36 sec per aircraft, 2-CPS decreases it by 63.32 sec per aircraft, and 3-CPS improves the average delay by 71.59 sec per aircraft. Therefore, although we explicitly only minimize the makespan, we also achieve substantial benefits in terms of the average delay incurred per aircraft, during congested times. We also point out that in this case, the minimum makespan (without CPS constraints, as computed using CPLEX) is the same as the makespan achieved using 2-CPS or 3-CPS. The optimal sequence of landings without CPS is shown in Fig. 16.

The algorithm we have proposed also runs fast enough in practice for a real-time implementation. The computational times for the scenario we have described above are shown in Table 6.

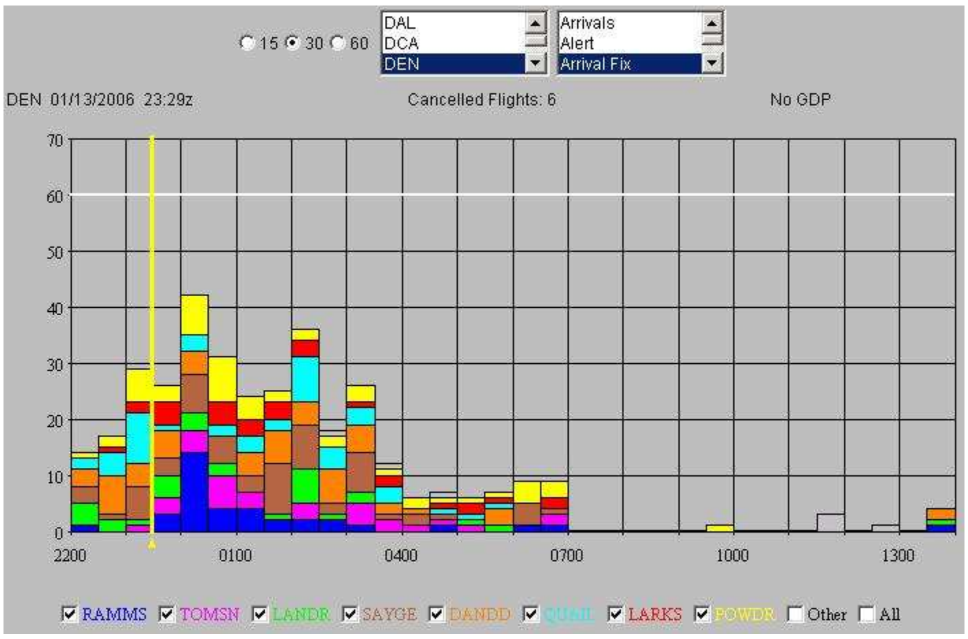

Figure 10. Real time data of demand at the different arrival fixes.

V. Extensions

The techniques we have proposed can be applied to solve several other problems. We briefly outline some of the extensions that we can solve, the details of which are beyond the scope of this paper.

A. Multiple runways

We consider the problem of scheduling and sequencing landings on multiple parallel runways. Even when there are multiple parallel runways being used simultaneously for landings at an airport, operations on the runways are not independent of each other, when the separation between their centerlines is less than 4300 ft. If the separation between the runway centerlines is less than 2500 ft, the separation requirements are the same as the inter-arrival separation for the single runway case (Table 1). One such example is Boston Logan airport, where runway 4R and runway 4L are 1600 ft apart.[6] If the separation between the runway centerlines is between 2500 ft and 4300 ft, the arrivals must be at least 1.5 nmi apart during the final approach. The multiple runway sequencing problem scenarios in which the runway separation is more than

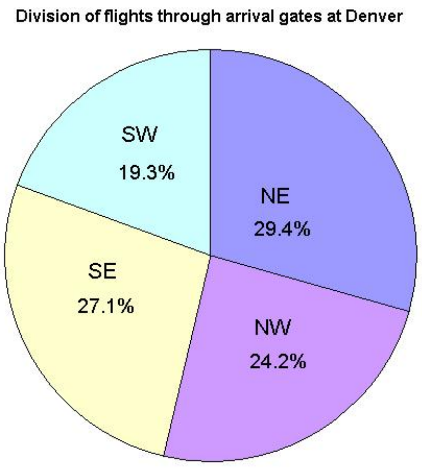

Figure 11. Percentage of flights through the different gates.

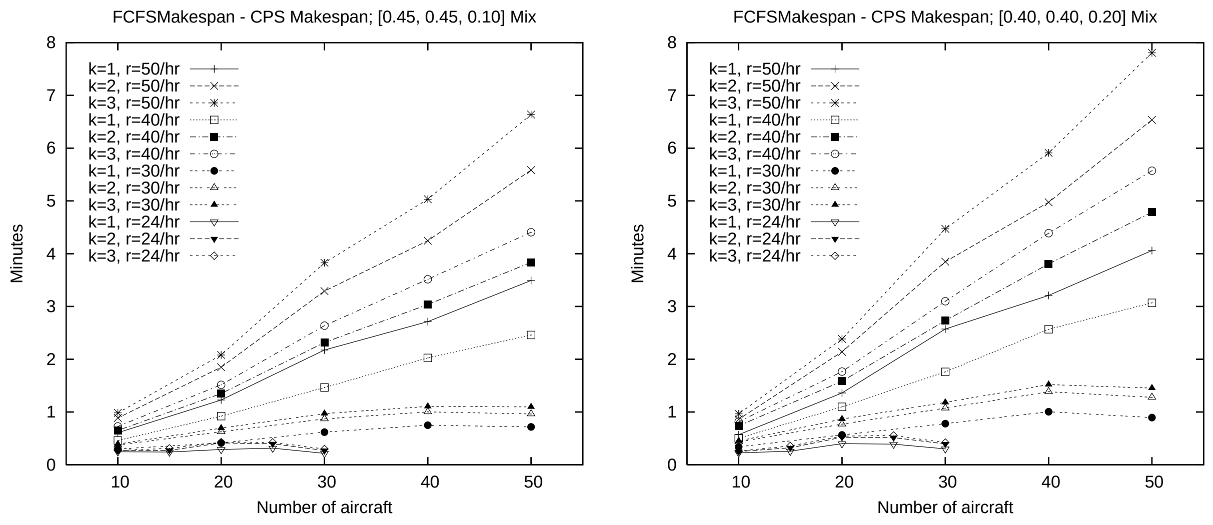

Figure 12. The difference between the FCFS makespan and the CPS makespan for two different aircraft mixes.

Procedure	Average delay (in min)	Landing time of last aircraft (in mins, past 9:00 AM)
FCFS	2.9908	144.18
1-CPS	2.0015	139.92
2-CPS	1.9355	139.06
3-CPS	1.7977	139.06
Optimal (using CPLEX)		139.06

Table 5. Average delay and makespan for scenarios shown in Figs. 13–15.

2500 ft, involve spacing requirements that do not satisfy the triangle inequality (Equation 1). As we have mentioned earlier, while our approach extends to these scenarios, a detailed description of these extensions is beyond the scope of this paper.

There are two possibilities to the multiple runway sequencing problem, depending on whether or not we wish to schedule the runways, along with computing the optimal landing sequence and landing times.

1. *Pre-assigned runways*

Aircraft are sometimes assigned to runways based on the direction of arrival. For example, in Denver International airport, when all aircraft land while flying West ("Land West all"), Runway 25 and Runway 26 are in use. In times of congestion, Northern arrivals land on Runway 26, and Southern arrivals land on Runway 25. In such cases, the algorithm needs to compute only the landing sequence and times, and not

Time period	Arrival Rate	# aircraft, n	k-CPS	Time (sec)		
				Read & Preprocessing	Dynamic Programming	Total
9:00 - 9:30	22 ac/hr	11	1-CPS	< 0.01	< 0.01	< 0.01
			2-CPS	< 0.01	< 0.01	< 0.01
			3-CPS	0.29	0.01	0.30
9:30 - 10:00	34 ac/hr	19	1-CPS	< 0.01	< 0.01	< 0.01
			2-CPS	< 0.01	< 0.01	0.01
			3-CPS	1.98	0.02	2.00
10:00 - 10:30	36 ac/hr	23	1-CPS	< 0.01	< 0.01	< 0.01
			2-CPS	0.03	< 0.01	0.03
			3-CPS	2.79	0.03	2.81

Table 6. Problem size and computation time for Denver International Airport example.

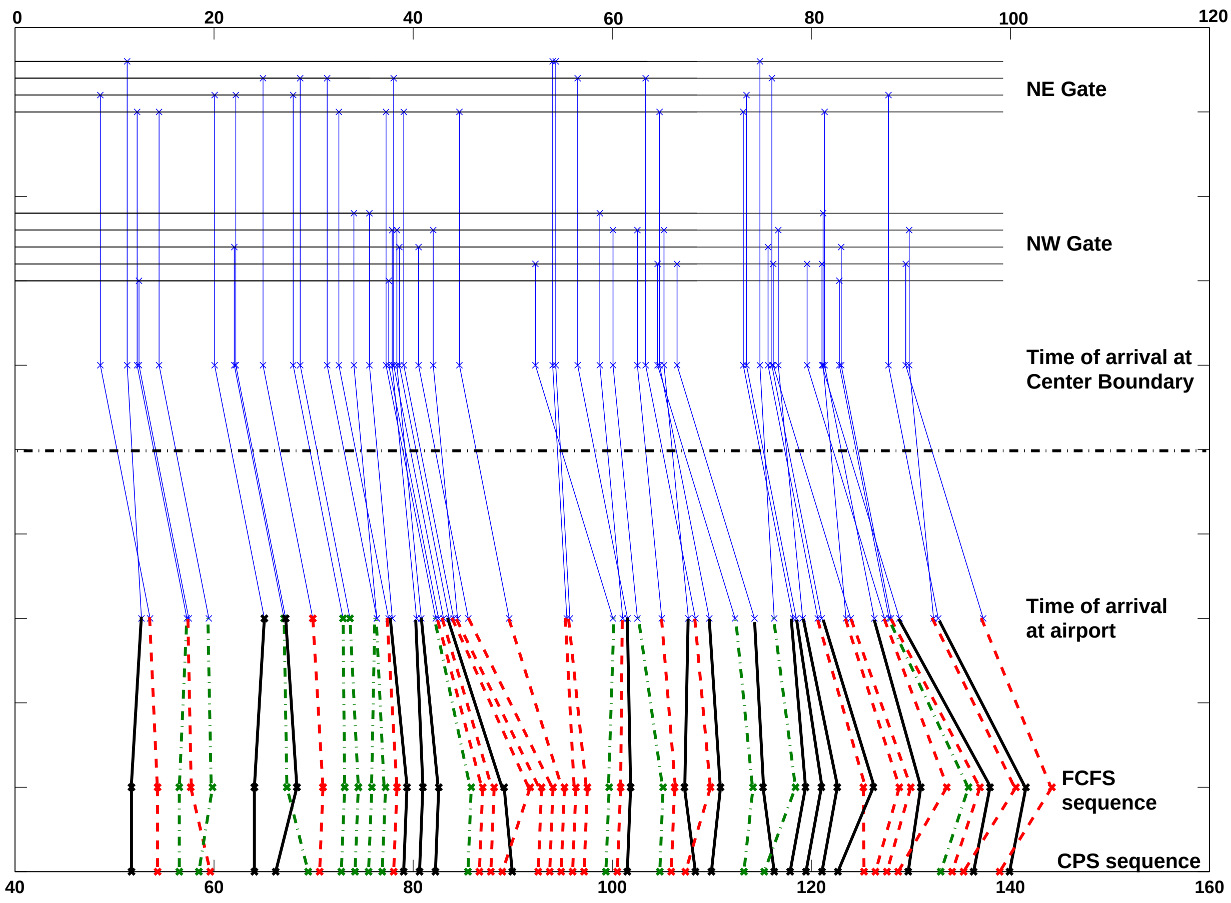

Figure 13. Simulated arrival traffic between 9:00 AM and 10:30 AM, with the 1-CPS solution.

the runway assignments.

2. Runway assignments and arrival sequencing

As we have seen, the inter-arrival spacing could be constrained, even in the case of parallel runways. In such cases, we need to schedule not only the landing times and the landing sequence, but also assign runways to aircraft optimally.

B. Scheduling and sequencing arrivals and departures

Similar to the requirements for minimum spacing between arrivals, the FAA specifies minimum separations between departures, between a departure and an arrival preceding it, and between an arrival and a departure preceding it, both for single-runway operations and parallel runway operations (de Neufville and Odoni[17]). Our approach extends to combined sequencing of arrival-departure operations.

C. Incorporating airline preferences

As we have seen in Section II, we can incorporate precedence relations in the landing sequence, while executing CPS. Airlines themselves could be the source of such precedence relations, arising from the *banking strategies* of the airlines. Airlines often schedule their flights at an airport such that passengers from banks of arriving flights connect to one or more departing flights. In such cases, not only are the airlines interested in reducing delays, they would like their aircraft to arrive before the connecting flight departs. Such precedence also arise when an airline schedules a flight to arrive at an airport, and shortly after that, the same aircraft is used on a flight to a different airport. While scheduling operations, especially arrival-departure combined sequences as in Section B, we can find a k-CPS sequence such that airline precedence relations are satisfied,

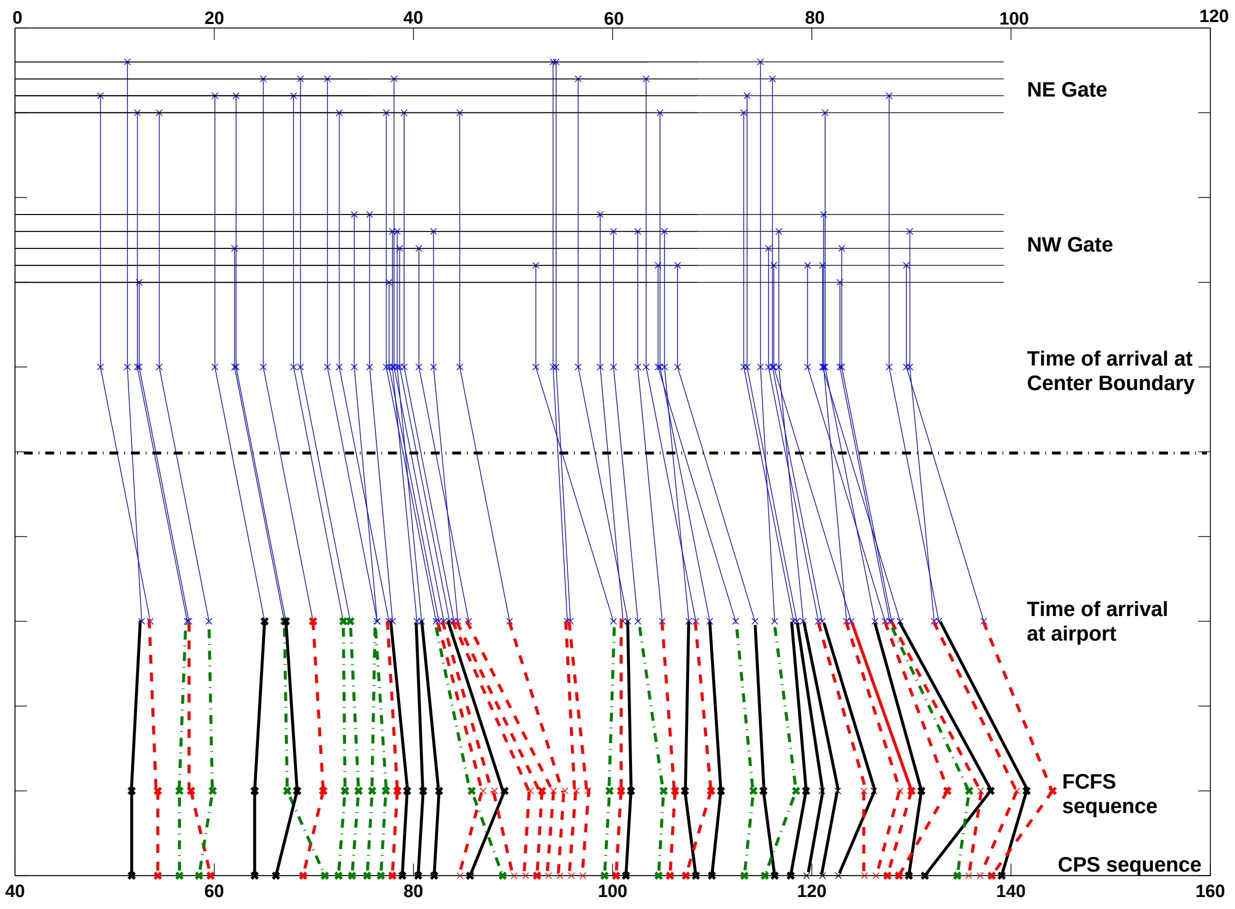

Figure 14. Simulated arrival traffic between 9:00 AM and 10:30 AM, with the 2-CPS solution.

if such a sequence is feasible. We can also quite easily determine if an airline's precedence preferences are not feasible.

D. Surface management

Another application of the techniques we have proposed in this paper is in the queuing for departure of aircraft at an airport. Aircraft depart from their gates ("push back") and form a queue at the edge of the departure runway. This queue is typically processed FCFS. An important question is how to establish an efficient departure sequence at the runway. Airlines request push back times, and pilots have even been known to push back early in order to achieve a higher position in the queue. In most airports, because of the taxiway geometry, ground controllers have very limited ability to reorder aircraft.[23] This is a limitation that can be resolved using the CPS algorithm that we have proposed in this paper. The maximum number of position shifts that can be achieved in the surface management problem appears to be $k = 1$ or 2, at most airports (Idris et al.[24]).

VI. Observations and conclusions

We presented an approach for minimizing makespan in the presence of CPS for a single runway, and demonstrated its effectiveness in a real-world setting at Denver International Airport. During congested periods of time, air traffic controllers would like to maximize the throughput of the runway system, making makespan minimization a desirable objective. Our most important contribution is that the approach we present can handle precedence constraints that could arise from operational constraints or airline preferences, and take into account restrictions on possible arrival times of aircraft. Our procedure delivers provably optimal solutions in a short time frame, making it ideal for deployment in a real-time scenario. The algorithm

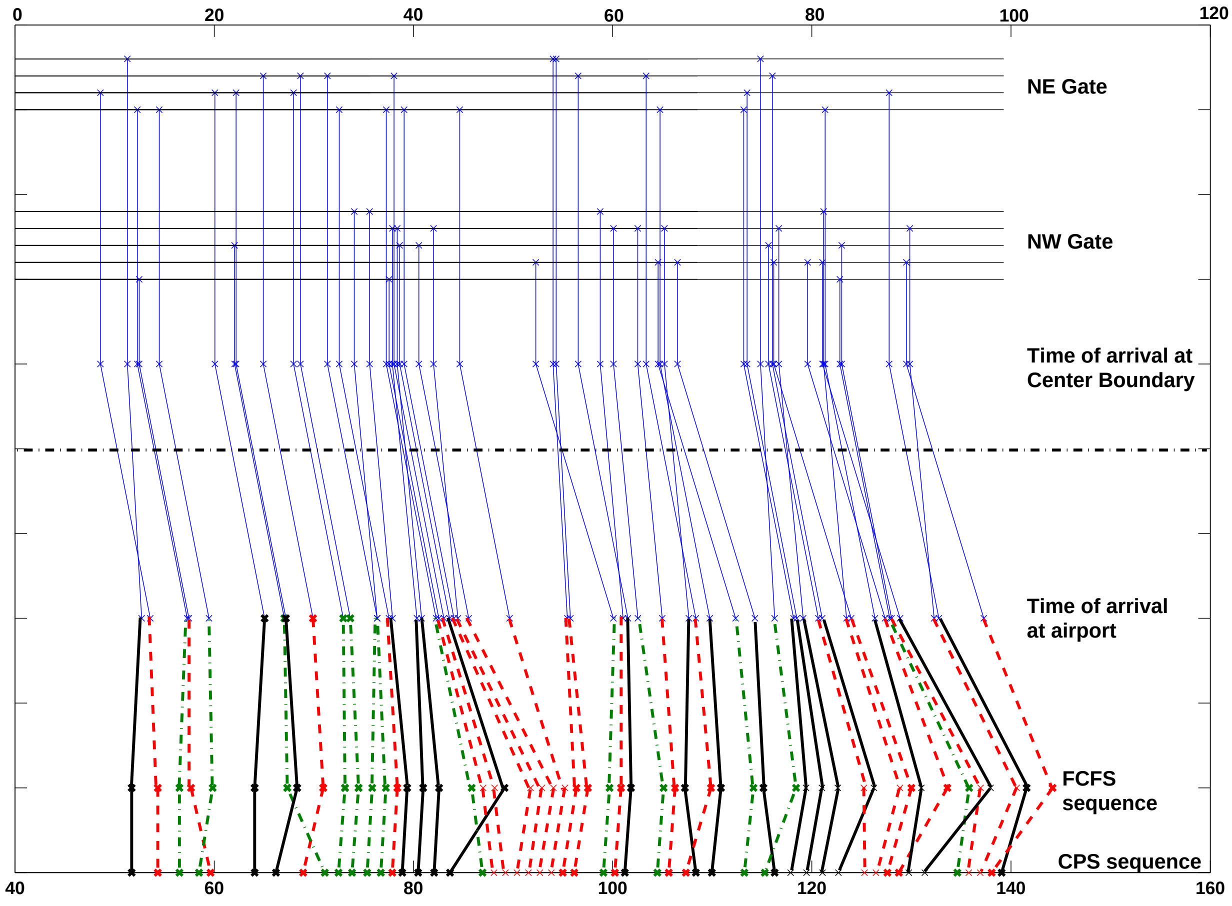

Figure 15. Simulated arrival traffic between 9:00 AM and 10:30 AM, with the 3-CPS solution.

is easy to implement and requires no special-purpose software packages for optimization.

Our initial experience indicates that for reasonable arrival rates, 3-CPS achieves solutions that are quite close to the optimal schedule. A useful property of the optimal CPS sequence seems to be that it also decreases average delay, although it is possible to construct specific instances in which decreasing the makespan increases the average delay.

We envision that our procedure could replace existing heuristic techniques for computing CPS sequences in decision support tools such as CTAS. It is also possible to extend the techniques discussed in this paper to other interesting air traffic management problems such as surface management at an airport.

References

[1]Willemain, T. R., Fan, H., and Ma, H., "Statistical Analysis of Intervals between Projected Airport Arrivals," DSES Technical Report No. 38-04-510, Rensselaer Polytechnic Institute, 2004.

[2]Neuman, F. and Erzberger, H., "Analysis of Delay Reducing and Fuel Saving Sequencing and Spacing Algorithms for Arrival Spacing," NASA Technical Memorandum 103880, October, 1991.

[3]Wickens, C. D., Mavor, A. S., Parasuraman, R., and McGee, J. P., editors, *The Future of Air Traffic Control: Human Operators and Automation*, National Academy Press, Washington, D.C., 1998, Report of the Panel on Human Factors in Air Traffic Control.

[4]Erzberger, H., Davis, T. J., and Green, S. M., "Design of Center-TRACON Automation System," *AGARD Meeting on Machine Intelligence in Air Traffic Management*, 1993, `http://www.ctas.arc.nasa.gov/`.

[5]Odoni, A. R., Rousseau, J.-M., and Wilson, N. H. M., "Models in Urban and Air Transportation," *Operations Research and the Public Sector: Handbooks in Operations Research and Management Science*, edited by S. M. Pollock, M. H. Rothkopf, and A. Barnett, Vol. 6, Elsevier Science, 1994.

[6]Venkatakrishnan, C. S., Barnett, A., and Odoni, A. R., "Landings at Logan Airport: Describing and Increasing Airport Capacity," *Transportation Science*, Vol. 27, No. 3, 1993, pp. 211–227.

[7]Dear, R. G., "The Dynamic Scheduling of Aircraft in the Near Terminal Area," Flight Transportation Laboratory Report R76-9, M. I. T, September 1976.

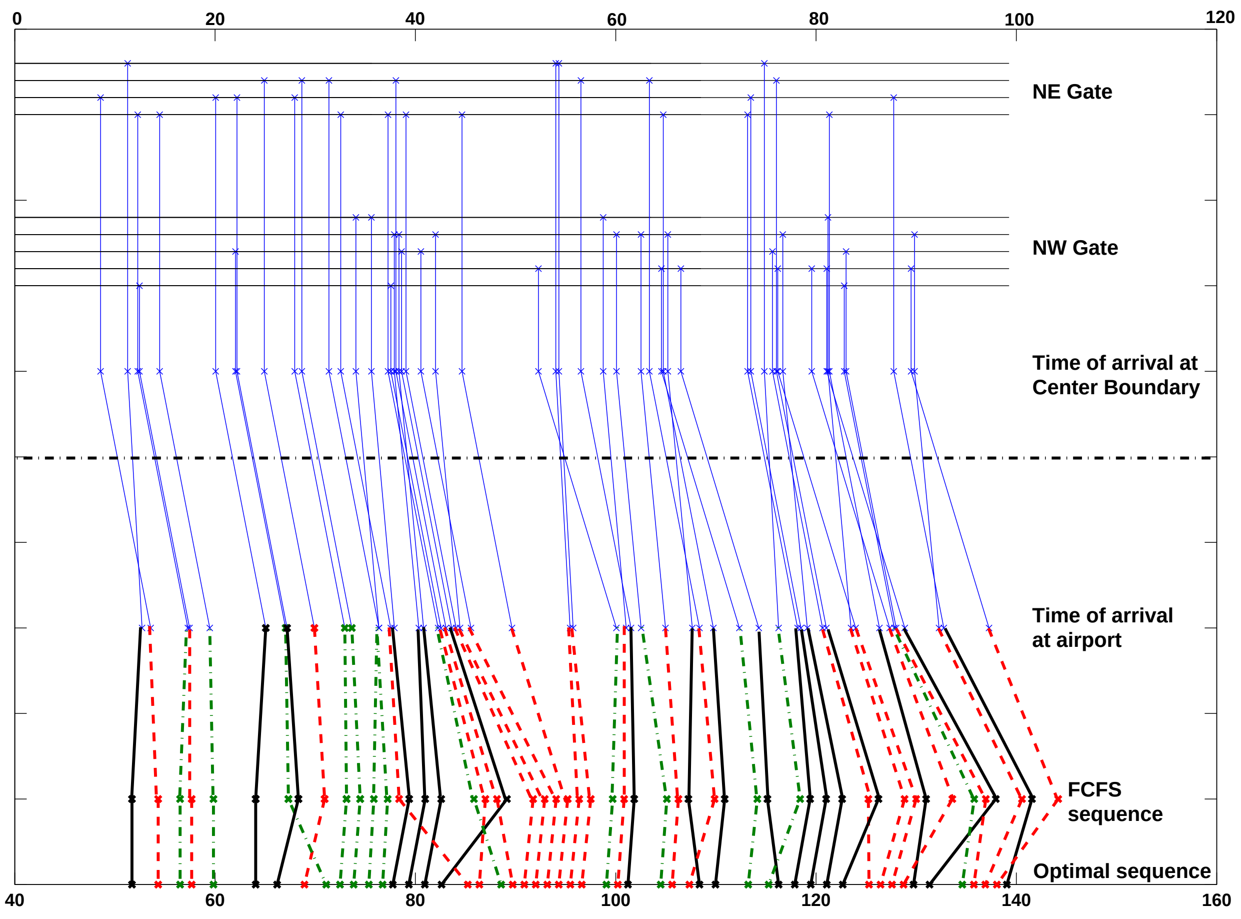

Figure 16. Simulated arrival traffic between 9:00 AM and 10:30 AM, with the optimal solution.

[8]Dear, R. G. and Sherif, Y. S., "An Algorithm for Computer Assisted Sequencing and Scheduling of Terminal Area Operations," *Transportation Research, Part A, Policy and Practice*, Vol. 25, 1991, pp. 129–139.

[9]Psaraftis, H. N., "A Dynamic Programming Approach for Sequencing Groups of Identical Jobs," *Operations Research*, Vol. 28, 1980, pp. 1347–1359.

[10]Trivizas, D. A., "Optimal Scheduling with Maximum Position Shift (MPS) Constraints: A Runway Scheduling Application," *Journal of Navigation*, Vol. 51, No. 2, May 1998, pp. 250–266.

[11]Volckers, U., "Arrival Planning and Sequencing with COMPAS-OP at the Frankfurt ATC-Center," *American Control Conference*, 1990.

[12]Garcia, J., "MAESTRO - A Metering and Spacing Tool," *American Control Conference*, 1990.

[13]Garey, M. R. and Johnson, D. S., *Computers and Intractability, A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York, 1979.

[14]Beasley, J. E., Krishnamoorthy, M., Sharaiha, Y. M., and Abramson, D., "Scheduling Aircraft Landings - The Static Case," *Transportation Science*, Vol. 34, No. 2, 2000, pp. 180–197.

[15]Bayen, A., Tomlin, C., Ye, Y., and Zhang, J., "An Approximation Algorithm for Scheduling Aircraft with Holding Time," *IEEE Conference on Decision and Control (CDC)*, 2004.

[16]FAA, Federal Aviation Administration, "Air Traffic Control: Order 7110.65P. Includes Change 3, August 4, 2005. (Appendix A)," `http://www.faa.gov/ATpubs/ATC/Appendices/atcapda.html`.

[17]de Neufville, R. and Odoni, A., *Airport Systems: Planning, Design and Management*, McGraw-Hill, 2003.

[18]Bureau of Transportation Statistics, accessed December 2005, `http://www.bts.gov`.

[19]National Oceanic and Atmospheric Administration: National Weather Service, "ACARS Plot of ZDV Traffic over a 24-hour Period," December 2002, `http://www.crh.noaa.gov/zdv/zdvtraffic.jpg`.

[20]Federal Aviation Administration, FAA, "Airport Arrival Demand Chart," January 13 2006, `http://www.fly.faa.gov/Products/AADC/aadc.html`.

[21]Fourer, R., Gay, D. M., and Kernighan, B. W., *AMPL: A Modeling Language for Mathematical Programming*, The Scientific Press (now part of Wadsworth Publishing), 1993.

[22]ILOG, Inc., "ILOG CPLEX 9.0 – Users Manual," December 2003, `http://www.ilog.com`.

[23]Atkins, S. and Brinton, C., "Concept Description and Development Plan for the Surface Management System," *Journal of Air Traffic Control*, Vol. 44, No. 1, January-March 2002.

[24]Idris, H. R., Anagnostakis, I., Delcaire, B., Hall, W. D., Clarke, J.-P., Hansman, R. J., Feron, E., and Odoni, A. R., "Observations of Departure Processes At Logan Airport to Support the Development of Departure Planning Tools," *Air Traffic Control Quarterly Journal*, Vol. 7, No. 4, 1999.