Heuristic and Exact Algorithms for Scheduling Aircraft Landings

Andreas T. Ernst,[1] Mohan Krishnamoorthy,[1] Robert H. Storer [2]

[1] CSIRO Mathematical and Information Sciences, Private Bag 10, Clayton South MDC, Clayton VIC 3169, Australia

[2] Department of Industrial Engineering, Lehigh University, Bethlehem, 18015 Pennsylvania

Received 11 June 1998; accepted 3 December 1998

Abstract: The problem of scheduling aircraft landings on one or more runways is an interesting problem that is similar to a machine job scheduling problem with sequence-dependent processing times and with earliness and tardiness penalties. The aim is to optimally land a set of planes on one or several runways in such a way that separation criteria between all pairs of planes (not just successive ones) are satisfied. Each plane has an allowable time window as well as a target time. There are costs associated with landing either earlier or later than this target landing time. In this paper, we present a specialized simplex algorithm which evaluates the landing times very rapidly, based on some partial ordering information. This method is then used in a problem space search heuristic as well as a branch-and-bound method for both single- and multiple-runway problems. The effectiveness of our algorithms is tested using some standard test problems from the literature. © 1999 John Wiley & Sons, Inc. Networks 34: 229–241, 1999

Keywords: aircraft landing; machine job scheduling; problem space search; simplex algorithm

1. INTRODUCTION

In this paper, we study the problem of landing planes at a busy airport. Given a set of planes in the radar horizon of an air-traffic controller (ATC), the problem is one of determining a landing time for each plane such that each plane in this ATC horizon lands within a prespecified landing time window and such that landing separation criteria specified for each pair of planes in this horizon are adhered to. This problem is a slightly unusual application that fits into the general category of machine-scheduling problems. Indeed, it is a special type of machine-scheduling problem with sequence-dependent processing times and with earliness and tardiness penalties for jobs not completed on target. This problem was presented first by Beasley et al. [5].

As in [5], we are only concerned with modeling and solving the problem of scheduling aircraft landings in this paper. However, the approach presented in this paper is easily adaptable to a problem featuring only takeoffs or, indeed, a problem involving a mix of takeoffs and landings on the same (or on different) runways. In addition, in this paper, we are concerned with the static case of scheduling aircraft landings. The dynamic case of scheduling aircraft landings was dealt with by Beasley et al. [4].

Consider the planes within the radar range (or horizon) of an ATC at an airport. Typically, the ATC will be in charge of determining approach paths, runway allocation,

Correspondence to: M. Krishnamoorthy; e-mail: Mohan.Krishnamoorthy@cmis.csiro.au

CCC 0028-3045/99/030229-13

and landing times of several planes in the radar horizon. Each of these planes has a preferred landing time as well as earliest and latest possible landing times. In addition, there are aerodynamic considerations that arise because of the turbulence created by landing aircraft. These considerations impose a constraint that the landing time between any two aircraft pairs in the horizon must be greater than some minimum interval. This minimum separation is dependent on the aircraft types and could be different for different pairs of aircraft in the horizon.

The problem of assigning to each plane a landing time such that all planes land as close as possible to their preferred landing time will be called the Aircraft Landing Problem (ALP). There are many variants of the ALP and many approaches for solving it. Most previous work in this area present simulation models (see Andreussi et al. [2]), queueing models (see Milan [17]), or heuristics (see Dear [10], Dear and Sherif [11, 12], Venkatakrishnan et al. [30]). A dynamic programming-based approach was considered by Psaraftis [21, 22]. Beasley et al. [5] provided a detailed review of the literature in this area. The work in [5] was based on an earlier mixed integer linear programming formulation and a genetic algorithm approach by Abela et al. [1]. There are, however, many (optimization) problems of interest in the area of air-space management and in air-traffic control. Some of these were discussed in, for example, Bianco and Odoni [6], Odoni et al. [18], and Winter and Nuber [31].

The ALP can be thought of as a machine job scheduling problem with sequence-dependent processing times and also with earliness and tardiness penalties for each job. However, there is a subtle but important difference between the ALP and more common types of machine job scheduling applications. In the ALP, the separation constraint must not only be satisfied between immediately successive planes in the sequence, but between all pairs of planes. For example, if planes i and j and j and k require a separation of 10 min and i and k require a 30-min separation, then landing i, j, and k at 08:00, 08:10, and 08:20, respectively, would not be a feasible solution. In standard machine job scheduling problems, on the other hand, this type of separation requirement between nonsuccessive jobs makes little sense.

Psaraftis [21, 22] and Bianco et al. [7] adopted a job-shop scheduling approach for solving a version of the ALP. Bianco et al. [9] and Bianco et al. [8] adopted a job-shop scheduling view of the ALP. They solved the single-runway problem using a mixed integer linear program. They provided a tree-search procedure with embedded Lagrangean lower bounds and also developed a heuristic approach for the problem. Their paper also presented a set of ALP data and provided computational results for all of the approaches. The results provided in [8] are, perhaps, the most complete set for the ALP apart from [5]. Most approaches for solving the job-shop scheduling problem with sequence-dependent processing times are heuristic in nature, although there exist some formulations, complexity results, worst-case performance bounds, and also some exact approaches. Some of these are found in, for example, [13–16, 19, 20, 24, 29].

The remainder of the paper is structured as follows: In Section 2, we provide a mathematical formulation of the ALP. In section 3, we present a specialized simplex method for finding optimal values of broadcast landing times given a determination of some or all off the sequencing variables. This algorithm can be used to quickly calculate lower bounds in a branch-and-bound scheme or to evaluate solutions in a heuristic for the plane-landing-sequence variables. We develop a heuristic approach based on problem space search. This is described in Section 4. The heuristic upper bound and the lower bounds that are obtained from the simplex algorithm are then combined in a branch-and-bound method together with several types of preprocessing steps to yield an effective exact solution algorithm. In Section 6, we then show how to extend the exact algorithm to the case of multiple runways. Finally, we present some numerical results in Section 7.

2. FORMULATION OF ALP

As in [5], we are only interested in modeling the *Decision* problem here. In other words, we are only interested in deriving, for each plane, a *broadcast* landing time. Along with this, the ATC must also determine solutions to the *Control* problem of *how* to land the planes. For example, a set of broadcast times may require some planes to overtake other planes in the static horizon, while some of the determined broadcast times may require the allocation of different trajectories and flight paths to various planes. We do not incorporate, into our decision model, any considerations that arise from these control problems.

A precise formulation of the ALP requires the following parameters:

- N the set of planes $N = \{1, \ldots, n\}$,
- E_i the earliest landing time of plane $i \in N$,
- L_i the latest landing time of plane $i \in N$,
- T_i the target (preferred) landing time of plane $i \in N$,
- S_{ij} the required separation time between planes i and j if i lands before j, $S_{ij} \geq 0 \ \forall i, j \in N$,
- g_i the penalty cost for plane i landing ahead of the target time,
- h_i the penalty cost for plane i landing after the target time.

We require the following decision variables in our decision model. We note that we deviate slightly from the variable notations that are used in [5]:

t_i the landing time of plane i,

b_{ij} a binary variable indicating that plane i lands before j.

A mathematical formulation of ALP can now be stated as follows:

$$\min \quad \sum_{i \in N} g_i \max\{0, T_i - t_i\} + h_i \max\{0, t_i - T_i\} \quad (1)$$

$$\text{Subject to:} \quad t_i + S_{ij} \leq t_j + M b_{ji} \quad \forall\, i \neq j \in N, \quad (2)$$

$$b_{ji} + b_{ij} = 1 \quad \forall\, i \neq j \in N, \quad (3)$$

$$E_i \leq t_i \leq L_i \quad \forall\, i \in N, \quad (4)$$

$$b_{ij} \in \{0, 1\}. \quad (5)$$

Here, M is a sufficiently large number. Needless to say, the formulation as it stands would not be of much use for calculating solutions to the ALP. It is only provided as a compact and precise mathematical definition of the problem. A slight variation to this problem is the multiple-runway case. Here, the planes can be assigned to one of two or more runways. In most applications involving multiple runways, the separation constraints need only be satisfied for planes landing on the same runway. The multiple runway problem was also discussed in [5].

3. A SIMPLEX ALGORITHM TO DETERMINE LANDING TIMES

We first observe that the ALP involves two decision problems: (1) a sequencing problem (which determines the sequence of plane landings) and (2) a scheduling decision problem (which determines the precise landing times for each plane in the sequence, subject to the separation constraints). We observe that *if* we are given the sequence in which planes land, then we can develop an algorithm for resolving the scheduling decisions. In this section, we show how the simplex algorithm can be specialized to the problem of finding the optimal landing times of all planes given a partial ordering of the planes. The algorithm described in this section is used later in a heuristic approach. It is also used to calculate lower bounds that are then embedded in a branch-and-bound algorithm to find an optimal solution.

Consider the problem of determining the landing times of N planes to land. Let $\preccurlyeq$ represent an ordering of planes. Note that $\preccurlyeq$ can either be a partial ordering or a total ordering. Additionally, we define $i \prec j$ if plane i appears before j in the sequence. In other words, $i \prec j$ if $i \preccurlyeq j$ and $i \neq j$. Thus, a partial ordering of the N planes defines a set of ordered pairs $U = \{(i, j) : i \prec j\}$.

We now want to find landing times t_i for each plane i such that

1. $t_i \in [E_i, L_i]$.
2. $t_i + S_{ij} \leq t_j$ if $i \prec j$ (however, the separation constraint need not be satisfied if $(i, j) \notin U$.
3. The cost g_i (h_i) for landing before (after) T_i is minimized.

We define variables x_i and y_i to, respectively, be the landing time of plane i before or after T_i that plane i should land. In other words, $t_i = T_i + y_i - x_i$. Now, the *primal landing time* problem can be written as

Problem PLT:

$$\text{Min} \sum_{i=1}^{N} g_i x_i + \sum_{i=1}^{N} h_i y_i$$

$$\text{Subject to: } x_i - x_j + y_j - y_i \geq T_i + S_{ij} - T_j \quad \forall\, (i, j) \in U \quad (6)$$

$$E_i - T_i \leq y_i - x_i \quad \forall\, i = 1, \ldots, N \quad (7)$$

$$y_i - x_i \leq L_i - T_i \quad \forall\, i = 1, \ldots, N \quad (8)$$

$$\mathbf{x}, \mathbf{y} \geq \mathbf{0}. \quad (9)$$

This is a linear program which could be solved by any standard LP algorithm. However, this would not exploit the special structure of this type of problem.

Define dual variables f_{ij}, s_i^+ and s_i^- corresponding to (6), (7), and (8), respectively, as the three sets of inequality constraints in PLT. Now, the dual of PLT can be written as

Problem DLT:

$$\text{Max} \sum_{i=1}^{N} s_i^+ (E_i - T_i) + \sum_{i=1}^{N} s_i^- (T_i - L_i) + \sum_{i<j} f_{ij} (T_j - T_i - S_{ij})$$

$$\text{Subject to: } \sum_{j>i} f_{ij} - \sum_{k<i} f_{ki} \leq g_i + s_i^+ - s_i^- \quad \forall\, i = 1, \ldots, N \quad (10)$$

$$\sum_{j>i} f_{ij} - \sum_{k<i} f_{ki} \leq -h_i + s_i^+ - s_i^- \quad \forall\, i = 1, \ldots, N \quad (11)$$

$$s^+, s^-, f \geq 0. \quad (12)$$

It is obvious that at most one of x_i and y_i will be nonzero in the optimal solution of PLT; hence, at most one of the constraints (10) and (11) can be active for any i. Also note that the left-hand side of Eqs. (10) and (11) are just the network-flow equations for a network in which directed arcs join i to j if $i \prec j$.

We will now proceed to construct a simplex algorithm, analogous to the network simplex algorithm, which solves DLT. The first step is to consider the structure of basic feasible solutions of this problem.

A *basic* solution consists of a set of *active* arcs forming a forest which spans the set of nodes (a single node implies a trivial tree). Each tree in the forest has a root node r. This node may be in one of three *states: early, late,* or *target.* Each nonroot node i can be in one of two states early or late. Hence, a basic solution is characterized by a forest of arcs and a state for each node. These have the following meaning:

- *If an arc* (i, j) is active, then planes i and j land with the minimum separation distance between them. Correspondingly, the dual variable f_{ij} carries some flow (all other arcs have zero flow). For basic feasible solutions, this flow must be positive.
- *If a nonroot node* i is early (late), then the plane lands before (after) T_i. In other words, x_i (y_i) is basic in PLT. In the dual, it means that i has a supply of g_i (demand of h_i). The divergence equation holds for all nonroot nodes.
- *The root node* r of any tree is used to make up the net difference between supply and demand over the whole tree. If it is in the early state, then s_r^+ makes up the difference (for feasible solutions, the amount of flow leaving r is greater than g_r). Similarly, if it is in the late state, then s_r^- makes up the difference (the amount of flow entering r should be greater than h_r). Finally, if it is in its third state (target), then the divergence equations are not applied to this node (the net difference between supply and demand should be between g_r and $-h_r$). In terms of the primal solution, the three states correspond to plane r landing at times E_r, L_r or T_r, respectively.

In Figure 1, we present an example of a single tree forming a basis and the variables involved in it.

Given a basis, the x and y variables can be calculated for each tree by starting with the root node and working down to the leaves using Eq. (6). Similarly, the flows f can be calculated for each tree by starting at the leaf nodes and working back to the root node, where the s^+ or s^- variable is determined. Based on this representation of basic feasible solutions, we can now develop a simplex algorithm for DLT.

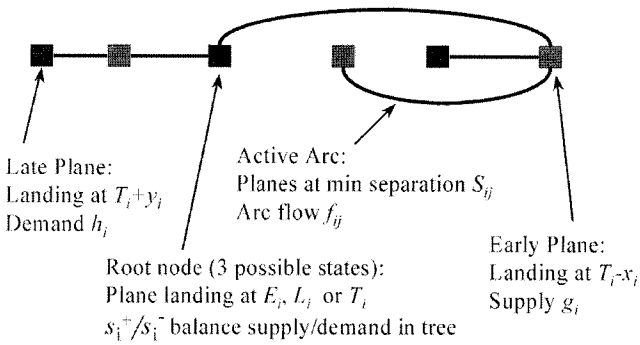

Fig. 1. Graph illustrating basic solutions.

Initialization: Start with a basic feasible solution for the dual problem. The easiest way to find such a solution is to land each plane at its target time. In other words, the basis consists of N trees each having only a root node with status target.

Reduced Costs: We calculate the x and y variables as described above. For any plane i if it lands outside of the interval $[E_i, L_i]$, then the corresponding s_i^+ or s_i^- has a positive reduced cost. Similarly, if for some $i \prec j$ the separation constraint is not satisfied, then f_{ij} has a positive reduced cost.

Selection of Incoming Variable: Two different selection mechanisms are used:

- In the heuristic, where a complete sequence is given, we first check whether each plane lands in its interval and satisfies the separation constraint to the next plane in the sequence. Only if no such violations are found do we check for each plane the separation to planes which are two or more positions later in the sequence. The first variable with a positive reduced cost is selected.
- In the exact method, we check all of the constraints for each plane in turn. However, rather than returning the first found, we check the amount by which the constraint is violated. If this amount exceeds a fifth of the amount of the incoming variable in the previous iteration (in other words, if the reduced cost coefficient is sufficiently "large"), then the variable is returned immediately. If no such variable is found, then one with the largest violation is returned.

Let k be the last plane that we considered in our search for an incoming variable. Then, for the next pivot, we start by considering plane $k + 1$. The main aim is (1) speed of the above calculations for finding an incoming variable, and (2) identifying large reduced costs that tend to yield larger improvements in the objective value.

Pivoting: Regardless of what variable is picked, the pivot operation is always identical—with one exception. We will describe the exception first.

If f_{ij} is picked to enter the basis and i and j are in the same tree, then the pivot operation is identical to that of the network simplex algorithm. We simply push flow around the cycle in the direction $i \rightarrow j$ until the flow in one of the arcs in the cycle becomes zero. This arc is then deactivated to give a new tree.

In all other cases, we push flow along a path with two different end nodes, say r and p. Assume that f_{ij} is entering the basis and that i and j are in different trees, with root nodes r and p, respectively. The path from r to p is made up of the path from r to i, arc (i, j) and the path from j to p. If p is in the same tree as r, and if s_p^+ is entering the basis, then the path is from r to p. Similarly, if s_p^- is entering the basis, then we use the same path in the opposite direction. In each case, the aim is to push flow along the new path (by changing the flows in the path as well as the supply at the end nodes), until one of the flows becomes zero or one of the end nodes changes status. Hence, this type of pivot can result in several different changes to the structure of the basis: (1) a tree breaking in two if arc (i, j) becomes nonbasic and r and p were originally in the same tree; (2) the root node of the tree changes, where p becomes the root node instead of r; and (3) two trees being joined.

The simplex algorithm then consists simply of repeated iterations of the above steps until an optimal solution is found. Note that apart from a speed advantage over a general simplex algorithm it also has the benefit that no matrix storage is required; hence, memory usage is significantly reduced.

4. A PROBLEM SPACE SEARCH HEURISTIC

In developing an effective branch-and-bound scheme, it is useful to obtain a good upper bound on the solution. For the ALP, we developed a problem space search (PSS) heuristic. The PSS is meta-heuristic. It attempts to combine a (usually) simple constructive heuristic with a genetic algorithm (or, perhaps, some other heuristic search algorithm). The main idea of PSS is that we do not attempt to search through the space of feasible solutions directly. Instead, we use a base heuristic as a mapping function between a space of perturbations and the solution space. Thus, it is sufficient to use the base heuristic and some method of searching through the perturbation space (in this current application, through a genetic algorithm), to find a good solution to the original problem. For more details and some variations on this basic idea of PSS, see Storer et al. [27].

In implementing PSS, we need to first choose a subset of the full set of input data to perturb. Often, it is advantageous to leave some of the input data unperturbed. Moreover, due to the number of repeated calls that need to be made to the base heuristic, a successful and effective implementation of PSS requires the base heuristic to be computationally efficient. Furthermore, the base heuristic must allow any solution to be generated, provided that a suitable perturbation is used. PSS has been successfully used in solving a variety of scheduling problems as well as other combinatorial optimization problems (see [25–28]).

The implementation of the PSS heuristic that we develop in this paper, for the ALP, consists of three components. The first component computes the optimal landing times (continuous solution) given a sequence of planes (integer solution). This is done using the specialized simplex algorithm described in Section 3. The second and third components are the constructive heuristic for generating a good sequence of planes and the PSS meta-heuristic, respectively. These last two components are described below.

4.1. The Constructive Base Heuristic

A sequence for landing the planes is constructed using a greedy approach based on a set of priorities. In each iteration k of the algorithm, the plane with the lowest priority is picked to be landed next, where the priority for each of the remaining planes is calculated as

$$p_j^k = \delta T_j + \epsilon \bar{E}_j^k + \alpha_j. \tag{13}$$

Here, δ and ϵ are two arbitrary positive weights and α_j is a perturbation of the priority. $\bar{E}_j^k$ is defined as the earliest time that plane j can land given the previous sequence of planes, that is, if the partial sequence $s_0, \ldots, s_{k-1}$ of planes has been constructed already, then

$$\bar{E}_j^k = \max\{E_j, \max_{i<k}\{\bar{E}_{s_i}^i + S_{s_i,j}\}\}$$

We then define the next plane to land as $s_k = \arg\min_{j \notin \{s_0,\ldots,s_{k-1}\}} p_j^k$.

Note that the earliest possible landing time for plane s_k is given by $E_{s_k}^k$.

This heuristic will not necessarily find a feasible landing sequence (it is possible that $\bar{E}_{s_k}^k > L_{s_k}$ for some k). This heuristic could be modified to reduce the likelihood that an infeasible sequence will be generated. However, in practice, the algorithm only rarely fails to provide a feasible solution as long as the perturbations $\boldsymbol{\alpha}$ are not too large.

The rationale behind this heuristic is that the natural way for the planes to land is in order of their target times; however, if their earliest possible landing time is quite late, then it may be better to use a plane which can land earlier even if its target time is a little later. Obviously, we use $\delta > \epsilon$. In fact, for the results presented below, $\delta = 8$ and ϵ

$= 1$. The perturbations α_j for each plane j are an input to this heuristic.

4.2. The PSS Meta-Heuristic

The PSS heuristic uses a genetic algorithm to search through the space of perturbation vectors $\boldsymbol{\alpha}$. We only use a simple GA scheme to search the perturbation space. Several sophisticated GA variations exist in the literature (see, e.g., [23]). The basic algorithm that we implement is as follows:

Algorithm 1

```
  Initialize population size n_p with random
individuals α_1, ..., α_{n_p}
for i = 1 to n_b do
    repeat
      Choose two members of the population
      as parents α_a, α_b
      Perform a single-point crossover
      with mutation to obtain α_c
    until α_c produces a feasible landing se-
    quence
    Evaluate α_c using the simplex algorithm
    Replace the worst member of the popu-
    lation with α_c
  next i
  Report the best solution found

end algorithm 1.
```

The following parameter settings are used for the results in this paper:

- The genes of the initial population as well as the mutations are generated using an exponential distribution for positive and negative α_i with an even chance of positive or negative values.
- The parameter λ used for the exponential distribution is

$$\lambda = \frac{1}{16}(\max_j\{\delta T_j + \epsilon E_j\} - \min_j\{\delta T_j + \epsilon E_j\}).$$

  This ensures that the perturbations α are of a similar size as the differences in priority values. The probability of a gene being mutated at birth is 0.15.
- The population size n_p is 200 and the number of births $n_b = 10{,}000$.

4.3. PSS for Multiple Runways

The PSS heuristic described (as described in this section so far) can be extended quite easily to take multiple runways into account. An initial attempt was to employ the constructive heuristic for assigning planes to runways. This did not provide promising results. Hence, the task of deciding on runways for each plane was added to the genetic algorithm.

Each gene of the population of multiple-runway solutions consists of a list of ordered pairs (α_i, r_i), where α_i is the perturbation used by the constructive heuristic as before and r_i is the runway index for plane i. For the initial population and mutations, the runway is chosen at random using a uniform probability distribution.

5. AN EXACT ALGORITHM FOR LANDINGS ON A SINGLE RUNWAY

We consider the problem of determining the optimal landing schedules of planes on a single runway. The basic idea of this method is to relax the total ordering requirement on the planes. For any pair of planes i and j which are not ordered, we can branch by including either (i, j) or (j, i) in U.

Note that we do not really need to get a total ordering but only include enough elements in U such that the solution of the subproblem satisfies the separation constraints for all pairs of planes (even if they are not in U). Before describing the branch-and-bound algorithm in detail, we describe some preprocessing methods that help to simplify the problem.

5.1. Preprocessing

Several types of preprocessing are performed to improve the performance of the exact algorithm. Most of these depend on obtaining a good upper-bound Z_U from the heuristic. The aim in all of these is to reduce the number of choices that need to be considered at the root and during the branch and bound (most of these procedures can also be applied before each node in the branch-and-bound tree is evaluated).

5.1.1. Tightening Intervals

This simple preprocessing procedure simply uses the upper-bound Z_U to reduce the interval $[E_i, L_i]$ as follows:

$$E_i \leftarrow \max\{E_i, T_i - g_i/Z_U\}$$

$$L_i \leftarrow \min\{L_i, T_i + h_i/Z_U\}.$$

Effectiveness: While this is not likely to make a big difference to the branch-and-bound method, it is a very cheap form of preprocessing and may assist some of the following preprocessing steps to eliminate some of the binary choices.

5.1.2. Upper-bound-based Fixing

Consider two planes i and j. What would be the cost of landing plane j before plane i if there are no other planes to be landed? Obviously, if $T_j + S_{ji} < T_i$, the cost is zero. Otherwise, we have to displace at least one of the planes

from its target time. Assume that $h_i < g_j$ (the case where $h_i \geq g_j$ is similar), then it is cheaper to move i back than to bring the landing time of j forward. Hence, the minimum cost is achieved by

$$t_i = \min\{T_j + S_{ji}, L_i\},$$

$$t_j = t_i - S_{ji}.$$

The cost of landing the planes is $h_i(t_i - T_i) + g_j(T_j - t_j)$. If this cost exceeds the upper bound, or $t_j < E_j$, then plane j cannot land before plane i; hence, $i \prec j$.

A second pass of this type of fixing can be made once some of the other preprocessing methods have given the ordering of some of the planes. In the second pass, we simply fix (j, i) and solve the lower-bound problem using the simplex method to see if this lower bound exceeds the upper bound. The advantage of this method is that it gives better lower bounds using information from the other preprocessing steps. The disadvantage is that it is relatively expensive.

Effectiveness: This method is very effective for the test data sets used in this paper. In most of the problems, it fixes a large number of pairs (i, j). In some of the examples, it determines the order (almost) completely, significantly reducing the work for the branch-and-bound algorithm. The second pass appears to be particularly effective for more difficult problems. However, the second pass is not worth including in the branch-and-bound method as it is too expensive to perform it for every node.

5.1.3. Fixing Based on Data

Consider two planes i, j with identical data ($E_i = E_j$, $L_i = L_j$, $T_i = T_j$, $g_i = g_j$, $h_i = h_j$ and $S_{ik} = S_{jk}$, $S_{ki} = S_{kj} \forall k$). In this case, it is irrelevant in which order i and j land so we can insert (i, j) into U arbitrarily.

More generally, i can be assumed to land before j in the (an) optimal solution if all of the following conditions are satisfied:

1. Plane i is earlier than plane j. More precisely, $E_i \leq E_j$, $T_i \leq T_j$ and $L_i \leq L_j$.
2. Landing i before j does not require more time, that is, $S_{ij} \leq S_{ji}$.
3. It is no more expensive (in terms of displacement of i and j) to land i before j than vice versa. Mathematically, ($h_i \geq h_j$ or $L_i \leq T_j$) and ($g_i \leq g_j$ or $T_i \leq E_j$). This eliminates the possibility that it may be much cheaper to, say, delay i for a very long time so as to land j on time (or even early).
4. Reversing the order and landing j before i would not reduce the separation to other planes. If some third plane k is comparable to both i and j (i.e., $k \prec i, j$ or $i, j \prec k$), then we only need to check the separation in one direction; otherwise, we check both. More precisely,

$$\forall\, k : (i, k) \notin U \text{ or } (j, k) \notin U, \quad S_{kj} \geq S_{ki}$$

and

$$\forall\, k : (k, i) \notin U \text{ or } (k, j) \notin U, \quad S_{jk} \leq S_{ik}.$$

If *all* the four conditions above are satisfied, then (i, j) can be inserted into U. There are many conditions that need to be satisfied, making it unlikely that a randomly generated problem would satisfy all of them for very many pairs. However, these pairs, if identified, have the potential to reduce processing times significantly. Assume, for example, that planes i and j have identical data. Then, they satisfy the above condition and would be fixed to land in some arbitrary order. However, if they were not included in U, then the branch-and-bound algorithm may branch on (i, j) and (j, i) at the root node, producing two identical subtrees, thereby doubling the number of nodes that have to be searched.

Effectiveness: This procedure usually manages to pick up a few pairs and seems to improve the processing time somewhat. There are some exceptions to this and these are highlighted in Section 7.

5.1.4. Inference

Since $\preccurlyeq$ is a partial ordering of the planes, it is transitive. In other words, $i \prec j$ and $j \prec k$ implies that $i \prec k$. Note that it is pointless to add (i, k) to U if $S_{ij} + S_{jk} \geq S_{ik}$. (A needless increase to the size of set U will only slow down the simplex algorithm.)

Effectiveness: This method was completely ineffective as a preprocessing procedure. Presumably, this is because the previous procedures would fix the third pair also if it fixed the first two pairs needed to make the inference.

5.2. Branch and Bound

The branch-and-bound algorithm is relatively straightforward. We start with a set U determined by the above preprocessing procedures. At each infeasible node, we pick the pair of planes i, j for which the separation constraint is violated by the greatest amount of time. In other words, we pick the pair (i, j) for which $v_{ij} = t_i + S_{ij} - t_j$ and v_{ji} are both positive and $v_{ij} + v_{ji}$ is maximal. Then, we branch by adding (i, j) and (j, i), respectively, to the set U. Since the upper bounds produced by our PSS heuristic are usually very tight, if not optimal, we are only interested in using the branch-and-bound procedure to prove the optimality of the

upper bound. Hence, we use the more memory-efficient depth-first search method. Note that since the basis requires so little memory it is easy to store it in the nodes of the branch-and-bound tree as a starting point for the child nodes.

To improve the performance of the algorithm further, we also apply some variable fixing methods during the branching process. There are two such methods:

Inference: This is the same method as in the preprocessing except that we only check for any triplet of nodes involving the pair (i, j) added to the set U most recently.

Cost-based Fixing: This method is identical to that described in Subsection 5.1.2 but tightened by use of the current lower bound. It makes use of the following observation: If we considered all of the planes in the same tree in the optimal solution of a subproblem, then their optimal landing times could not be improved even if we removed all of the other planes in the rest of the forest. Hence, when we consider landing j before i, we can add to the lower bound the cost of landing all the planes other than those in the trees containing i and j, at the time given by the current lower-bound solution. In other words, changing the order in which i and j land cannot improve the landing times of any plane other than those that are in the same tree, hence, the remaining trees are still valid lower bounds even if (j, i) were added to U.

Effectiveness: Both of these methods generally fix a large number of pairs during the running of the algorithm. Naturally, these additions to U are only valid for the descendents of the node at which U was modified.

6. AN APPROACH FOR SOLVING MULTIPLE-RUNWAY PROBLEMS

So far in this paper, we have concentrated on the single-runway problem. We now extend and modify the single runway results to cope with the problem of determining a schedule of plane landings on multiple runways. The components of the algorithm are the simplex method for determining optimal landing times, a heuristic for assigning planes to runways, and a branch-and-bound method which makes some use of preprocessing, similar to that used for the single-runway problems. As before, we denote by U the set of all the pairs of planes (i, j) which have to land in that order *on the same runway*, that is, U only contains pairs of planes which we have constrained *a priori* to land on the same runway. Let $R = \{1, 2, \ldots, |R|\}$ be the set of runways, indexed by r.

6.1. Assigning Planes to Runways

Assume that we are given the landing times for each plane as well as a set U. Any algorithm that tries to allocate the planes to runways has to do the following:

1. For any $(i, j) \in U$, planes i and j have to be assigned to the same runway.
2. All runways must be used, unless it is possible to land all of the planes on fewer runways than are available without violating the separation constraints.
3. If no feasible solution can be found, the algorithm has to indicate which planes conflict with each other, thus preventing a solution from being found.

The set U defines groups of planes that have to land on the same runway. For each of these groups containing more than two planes, we need to check that the separation constraint is satisfied. For example, it is possible that (i, k) and (j, k) are in U, so that i, j, and k have to land on the same runway, but i and j have been assigned the same landing time. If an infeasible pair of planes is found, then we return the pair with maximum conflict. The branch-and-bound algorithm can then branch on the two possible orderings of the pair. [In the above example, we would branch by including (i, j) or (j, i) in U.]

Next, we order the planes according to the landing times assigned to them by the simplex algorithm. We then go through the list of planes in chronological order and try to assign to each plane a runway on which it will not conflict with previous planes. If there are several runways on which plane i could land, then we choose the runway for which the gap to the previous planes is smallest (breaking ties arbitrarily). More precisely, we choose a runway r which minimizes

$$G_i^r = t_i - \max_{j \in A_i^r} \{t_j + S_{ji}\}, \tag{14}$$

where A_i^r is the set of planes already assigned to runway r, and t_j, the landing time of plane j. On the other hand, if G_i^r is negative for each runway r, then we cannot find a feasible solution. Let C_i be a set of $|R| + 1$ planes consisting of i and one plane j in each A_i^r which attains the maximum in (14). In order to land all of the planes on the available runways, at least two of the planes in C_i must land on the same runway. Rather than stopping as soon as we find the first plane i that cannot be landed, we assign i to the runway r with the largest (least negative) G_r^i and continue assigning planes to runways. Once all planes have been assigned to runways, we choose the set C_i for which $\max_{r \in R} g_r^i$ is smallest (most negative) to continue the branch-and-bound process. Note that the above process is a heuristic assignment method. Even if this method produces suboptimal assignments in some cases, the branch and bound will

ensure that the optimal solution is found eventually as described below.

6.2. Preprocessing

Some use can be made of the preprocessing techniques that we developed for the single-runway problems. The method of tightening landing intervals based on the upper bound still applies. We define a set V of planes (i, j) such that *if* i and j land on the same runway *then* j must land after i. The two methods described in Sections 5.1.2 and 5.1.3, which order pairs of planes based on the upper bound and based on the problem data, respectively, can only be used to generate elements of V. They both require the assumption that the planes land on the same runway.

The inference method of Section 5.1.4 *cannot* be used. This is because V is not a partial ordering of the planes. For example, $(i, j) \in V$ and $(j, k) \in V$ does not imply that $(i, k) \in V$ because i landing before k holds only if i, k, *and* j all land on the same runway. It is possible that both $(i, j) \in V$ and $(j, i) \in V$ for some pair of planes i, j. This means that the two planes have to land on different runways, since landing them on the same runway would lead to a contradiction. The set V, defined during preprocessing, can be used to cut down the size of the branch-and-bound tree as shown in the next section.

6.3. Branch and Bound

We start the search for an optimal solution with an empty set U and with V given by the preprocessing. The branch-and-bound method for the multiple-runway problem differs from that for the single-runway case in that a node may have either two descendants or $|R|^2 + |R|$. The first case occurs when two planes i, j in the same group are found to violate the separation constraint. In this case, the pairs (i, j) and (j, i) are added to U (and V) to produce two new nodes of the branch-and-bound tree. The second scenario is where we have a set C_i of $|R| + 1$ planes that clash produced by the heuristic method of assigning planes to runways. In this case, we generate a descendant node for each pair (j, k) with $j, k \in C_i$ and $j \neq k$.

Note that it is possible that one or more of these pairs (j, k) could be allocated to the same runway without changing the landing times (in other words, the heuristic may have made a poor choice in allocating planes to runways before coming to plane i). In this case, we may choose to run the heuristic again but try to put j and k on the same runway to see if this will lead to a feasible solution. However, experimental experience suggests that this is hardly worth doing. Hence, we only run the heuristic once for each node.

We again use a simple depth-first method for processing the branch-and-bound tree. As with the algorithm for single-runway problems, there are several ways to reduce the size of the tree. For each new node, created by adding the pair (i, j) to U, we can do the following:

Cutting Based on Upper Bound: This is the standard way of cutting the branch-and-bound tree. Whenever the lower bound calculated by simplex is greater than the upper bound (or the subproblem is infeasible), we can abandon the current node and backtrack.

Cutting Using *V*: We can make use of V to cut branches at which pairs of planes have been placed on the same runway in an order that cannot lead to an optimal solution. Whenever (i, j) is to be added to U and $(j, i) \in V$, we can immediately cut of that branch.

Inference: This can be applied not only to the set U but also to V for conclusions based on (i, j). For any pair $(i, j) \in U$, if j and k land on the same runway, then so do i and k. Hence, from $(i, j) \in U \subseteq V$ and $(j, k) \in V$, we can conclude that $(i, k) \in V$.

Cost-based Fixing: As before, we can use the upper bound to add pairs to V (which may then be used to cut branches further down in the tree).

7. RESULTS

This section contains some computational results of the algorithms presented in this paper. We test the algorithms for single and multiple runways on the ALP test data sets provided by the OR problem library maintained by Beasley [3] (problems number 1–8) as well as two new problems (problems 9 and 10). All programs were run on a DEC 3000/700 (200 MHz Alpha CPU). The exact algorithms were programmed in C and compiled with maximum compiler optimization.

The problem space search heuristic was tested on both single- and multiple-runway problems. For each problem, the heuristic was run 20 times to give a good indication of the average behavior of the heuristic. Table I shows the results for the single-runway problems. The upper gap from the optimal solution (denoted as Gap), defined as $(Z_U - Z_{opt})/Z_{opt}$, is given in Table I. We provide the worst upper Gap, the best upper Gap, as well as the average upper Gap over 20 trials. The last two columns in the table give the average number of births (individuals evaluated) and the average running time over all trials. As can be seen, the heuristic manages to produce solutions in a reasonable amount of CPU time. In most cases, the solution is consistently optimal. In the two problems where this is not the case (problems 8 and 10), it, nevertheless, manages to produce a reasonable solution with only occasional outliers where the heuristic gets stuck in an local minimum.

Table II shows the results of the upper-bounding heuris-

TABLE I. Heuristic results for a single runway

Prob No.	No. Planes	Min Gap (%)	Avg Gap (%)	Max Gap (%)	No. Births	CPU (sec)
1	10	0.00	0.00	0.00	5075	0.71
2	15	0.00	0.00	0.00	6339	1.24
3	20	0.00	0.00	0.00	5463	1.63
4	20	0.00	0.00	0.00	5226	2.14
5	20	0.00	0.00	0.00	5466	2.32
6	30	0.00	0.00	0.00	5230	5.67
7	44	0.00	0.00	0.00	6304	9.87
8	50	0.00	0.99	10.51	11160	20.54
9	30	0.00	0.00	0.00	6215	7.95
10	44	0.00	2.57	7.42	15636	34.42

tic for the multiple-runway problems. For each data set, we start with two runways and solve with an increasing number of runways until it is possible to land all planes without delay. In most cases, this requires three or four runways, the exception being problem 7, where two runways are sufficient. Instead of showing the upper gaps, we show the actual minimum, average, and maximum objectives. This is because the percentage gap as defined above is meaningless when the objective is zero, as is the case in many of the examples. For each of the problems, the minimum objective value found is optimal. The running time is similar to that required by the single-runway problems. However, the solution quality is less consistent.

In many of the dual runway examples, the heuristic occasionally fails to find the optimal solution. This is presumably because the perturbations are used in the base

TABLE II. Heuristic results for a multiple runways

Prob No.	No. Planes	No. Runways	Min Cost	Avg Cost	Max Cost	No. Births	CPU (sec)
1	10	2	90.00	90.00	90.00	4075	0.62
1	10	3	0.00	0.00	0.00	4079	0.57
2	15	2	210.00	210.00	210.00	4080	1.16
2	15	3	0.00	0.00	0.00	4085	1.06
3	20	2	60.00	74.00	100.00	4470	2.04
3	20	3	0.00	0.00	0.00	3482	1.49
4	20	2	640.00	640.00	640.00	4237	2.22
4	20	3	130.00	130.00	130.00	4176	1.95
4	20	4	0.00	0.00	0.00	2696	1.14
5	20	2	650.00	650.00	650.00	4478	2.40
5	20	3	170.00	170.00	170.00	4512	2.09
5	20	4	0.00	0.00	0.00	973	0.42
6	30	2	554.00	569.45	758.00	5984	6.60
6	30	3	0.00	3.45	39.00	750	0.76
7	44	2	0.00	0.00	0.00	1235	2.57
8	50	2	135.00	184.25	255.00	8361	23.08
8	50	3	0.00	3.00	15.00	2613	7.02
9	30	2	219.00	219.05	220.00	5331	6.37
9	30	3	0.00	0.00	0.00	486	0.54
10	44	2	660.00	864.00	1080.00	8577	26.26
10	44	3	63.00	94.30	178.00	7742	16.73
10	44	4	0.00	1.90	20.00	2676	5.55

TABLE III. Single-runway problems

Prob No.	No. Planes	Opt. Cost	Lower Gap (%)	Preprocessing			B&B Fixed Per Node		No. Nodes	CPU (sec)
				Cost I	Data	Cost II	Cost	Inference		
1	10	700	0.00	31	9	5			0	0.00
2	15	1480	20.27	60	36	7	0.00	0.00	4	0.01
3	20	820	0.00	146	37	6			0	0.01
4	20	2520	28.17	70	99	17	0.00	0.17	6	0.02
5	20	3100	2.58	57	105	25	0.50	0.50	2	0.03
6	30	24442	0.00	408	7	20			0	0.04
7	44	1550	0.00	924	22	0			0	0.01
8	50	1950	93.38	813	5	50	17.65	1.36	2,723	9.06
9	30	3721	89.20	17	154	141	1.81	1.70	35,410	33.77
10	44	20391	96.17	0	462	137	0.08	40.15	4,702,168	2:14:09

heuristic only to determine the order. The assignments of planes to runways is affected only indirectly by the perturbations. It would be possible to modify the base heuristic to add perturbations for the construction of assignments of planes to runways. However, such an approach would require not only additional coding but also a greater effort by the genetic algorithm since the parameter space would be significantly increased in this manner. Given that the average solution quality is still reasonable, the extra effort seems unjustified.

The results of the exact algorithm for single-runway problems are summarized in Table III. The column headings have the following meaning:

For each problem (Prob no.), we give the number of planes (No. planes) involved and the optimal objective value (Opt cost). We also provide the initial lower gaps (Lower Gap %). Next, we list how many pairs (i, j) were added to U in the preprocessing based on

Cost I: cost during phase one (see Section 5.1.2),
Data: problem data (see Section 5.1.3), and
Cost II: cost during phase two (see Section 5.1.2).

This, together with the lower gap, provides an idea of the effectiveness of the preprocessing. Here, the lower gap is defined as the relative deviation of the lower bound from the optimal solution, expressed as a percentage. Naturally, without preprocessing, $U = \emptyset$ and, hence, the lower bound is zero (Gap = 100%).

Next, we show the effectiveness of fixing variables during the branch and bound (B&B fixed per node) using reduced-cost coefficients (Cost) and using the inference rule (Inference). This is done by showing the *average* number of pairs (i, j) added to U per node of the branch-and-bound tree (excluding the root node). Finally, the last two columns of Table III gives the number of nodes required to obtain the optimal solution (No. nodes) as well as the total running time.

As can be seen from the table, our exact algorithm solves all of these problems with ease, with exception of the last two. A significant component of the success of the algorithms is obviously the effectiveness of the preprocessing. It can be seen from the lower gaps that this preprocessing significantly improves the lower bounds. The remaining fractional variables can then be resolved very quickly using a few branch-and-bound nodes. Even for Problem 8, where the lower bound is quite poor, the speed of the simplex method for generating lower bounds makes it possible to search a relatively large branch-and-bound tree quickly.

The effectiveness of the different variable fixing methods both during preprocessing and in the branch-and-bound part of the algorithm varies greatly depending on the problem data. This is only to be expected. However, it is interesting to note that the two preprocessing methods based on the objective value and based on problem data complement each other: For problems in which one does rather poorly, the other will do better and vice versa.

It is worth discussing problems 9 and 10 and why the performance for these problems is significantly different than for the other problems. In both of these problems, the aircraft are essentially of two types (A and B) with separation requirements between any pair of planes only depending on their type. Furthermore, all of the planes have $g_i = h_i = 1$, $E_i = T_i$, and $L_i = \infty$. Hence, for these problems, the objective is to minimize the sum of the delays (with no early landings allowed). Each of the planes has only a marginally different target landing time T_i. Thus, there is little to distinguish one solution from another, forcing the branch-and-bound algorithm to search a large number of nodes with almost identical costs. The difference between problems 9 and 10 is that in the latter the target landing times are closer together—the 44 planes are trying to land over (almost) the

TABLE IV. Multiple-runway problems

Prob No.	No. Planes	No. Runways	Opt. Cost	Preprocessing		B&B Fixed Per Node		No. Nodes	CPU (sec)
				Cost	Data	Cost	Inference		
1	10	2	90	48	0	0.00	0.00	1	0.00
2	15	2	210	104	1	0.00	0.00	2	0.00
3	20	2	60	192	2	0.00	0.00	1	0.00
4	20	2	640	158	24	0.75	0.53	533	0.36
4	20	3	130	196	2	3.33	0.18	6	0.01
5	20	2	650	155	27	0.78	0.62	669	0.44
5	20	3	170	194	1	2.54	0.00	24	0.02
6	30	2	554	422	4	3.67	0.49	154	0.22
8	50	2	135	1215	0	9.45	0.09	11	0.06
9	30	2	219	410	11	4.42	0.43	137	0.21
10	44	2	660	668	139	7.29	0.92	94614	292.05
10	44	3	63	967	0	10.14	0.07	14	0.05

same period as the 30 in problem 9 with the same separation requirements.

The results for the exact algorithm for multiple runways is given in Table IV. Here, we only report the results for all runway numbers which lead to a nontrivial solution. In other cases, the solution was found "instantaneously." In other words, Problem 7 is not represented in Table IV because all 44 planes in Problem 7 would land on target (thereby producing an optimal solution value of 0) if we present it with two runway options to land on.

The algorithm generally has no difficulties solving these problems; the only one which requires a bit of effort is problem 10 with two runways. The reason for this is the same that was offered for the difficulty in solving Problem 10 in the single-runway case. Note that initially the set V is empty. Hence, the initial lower bound is zero and we cannot use the second phase of the cost-based preprocessing method. However, even when this increases, the number of nodes required compared with the single-runway case, as, for example, in problem 6, the total running time does not change greatly.

The results for both the single- and the multiple-runway case compare favorably with those obtained exact algorithm described by in [5]. They only provide results for Problems 1–8. Their MILP-based algorithm was tested on the same computer that we used for our tests. For some of the larger single-runway problems, our new approach is significantly faster. For example, the single-runway cases of Problems 5 and 8 are solved by the approach of [5] in 922 and 111.9 seconds, respectively. Our approach solves these problems in 2.35 and 29.6 seconds, respectively (including the time for our heuristic). The results for the multiple runway problems are even more different. The MILP approach of [5] solves Problem 5 with two runways and Problem 8 with two runways in 11510.4 and 3450.6 seconds, respectively. Our approach is able to solve these problems in 2.84 and 23.14 seconds, respectively. It is interesting to note that the MILP-based approach of [5] took longer on the multiple-runway problems (due to the significantly increased number of variables) even though the results above show that the combinatorial decisions of these problems are actually easier to resolve.

8. CONCLUSIONS

In this paper, we developed a specialized lower-bounding method for the ALP. This method is based on the simplex algorithm, but makes use of the specific problem structure of the LP relaxation of the ALP to produce a much faster algorithm. This yields a fast method for finding the landing times of planes in the static horizon of an ATC, given a partial ordering of the aircraft. This fast method for finding landing schedules is used both in a heuristic as well as a branch-and-bound method. We developed a heuristic based on PSS. This heuristic is extremely effective for all of the test problems. The branch-and-bound method depends heavily on the success of several preprocessing methods that are described in this paper. In all but two (artificially derived difficult) problems, the exact method is very efficient in solving both the single- and multiple-runway problems in a reasonable amount of computational time. Our methods are also an improvement on a comparable method that exists in the literature for such problems.

It is of interest to see if the results that we describe in this paper can be transferred to yield good algorithms for a similar problem in the machine-scheduling literature—the problem of scheduling jobs on machines with sequence-dependent processing times. This is a topic of current research. Moreover, it is of interest to see if some of the

results that we have described in this paper can be used to solve the practical problem of scheduling planes in a "dynamic" ATC horizon, in which some planes land (and, hence, depart from the ATC horizon) and new planes appear (into the ATC horizon) at a steady rate.

REFERENCES

[1] J. Abela, D. Abramson, M. Krishnamoorthy, A. De Silva, and G. Mills, Computing optimal schedules for landing aircraft, Proc 12th National ASOR Conf, Adelaide, Australia, 1993, pp. 71–90.

[2] A. Andreussi, L. Bianco, and S. Riccardelli, A simulation model for aircraft sequencing in the terminal area, EJOR, 8 (1981), 345–354.

[3] J.E. Beasley, Or-library: Distributing test problems by electronic mail, J Oper Res Soc 41 (1990), 1069–1072.

[4] J.E. Beasley, M. Krishnamoorthy, Y.M. Sharaiha, and D. Abramson, Dynamically scheduling aircraft landings—the displacement problem (submitted).

[5] J.E. Beasley, M. Krishnamoorthy, Y.M. Sharaiha, and D. Abramson, Scheduling aircraft landings—the static case, Trans Sci (to appear).

[6] L. Bianco and A.R. Odoni (Editors), Large scale computation and information processing in air traffic control, Springer-Verlag, 1993.

[7] L. Bianco, B. Nicoletti, and S. Ricciardelli, "An algorithm for optimal sequencing of aircraft in the near-terminal area," Optimisation techniques: Proc 8th IFIP Conf on Optimization Techniques, I. Stoer (Editor), Lecture Notes in Control and Information Sciences, Vol. 7, Springer-Verlag, Wurzburg, 1977, pp. 443–453.

[8] L. Bianco, S. Ricciardelli, G. Rinaldi, and A. Sassano, Scheduling tasks with sequence dependent processing times, Naval Res Log 35 (1988), 177–184.

[9] L. Bianco, G. Rinaldi, and A. Sassano, "A combinatorial optimization approach to aircraft sequencing problem," Flow Control of Congested Networks, L. Bianco, A.R. Odoni, and G. Szego (Editors), NATO ASI Series, Series F: Computer Systems and Systems Science, Vol. 38, Springer-Verlag, Berlin, 1987, pp. 323–339.

[10] R.G. Dear, The dynamic scheduling of aircraft in the near terminal area, Research report R76-9, Flight Transportation Laboratory, MIT, 77 Massachussetts Ave., 33-412, Cambridge, MA 02139, 1976.

[11] R.G. Dear and Y.S. Sherif, An algorithm for computer assisted sequencing and scheduling of terminal area operations, Microelect Reliab 29 (1989), 743–749.

[12] R.G. Dear and Y.S. Sherif, An algorithm for computer assisted sequencing and scheduling of terminal area operations, Trans Res Part A—Policy Pract 25 (1991), 129–139.

[13] P.M. Franca, M. Gendreau, G. Laporte, and F.M. Muller, A tabu search heuristic for the multiprocessor scheduling problem problem with sequence dependent processing times, IJPE, 43 (1996), 79–89.

[14] A. Guinet, Scheduling sequence-dependent jobs on non-identical parallel machines to minimise completion time criteria, Int J Prod Res 31 (1993), 1579–1594.

[15] C. Jordan and A. Drexl, A comparison of constraint and mixed-integer programming solvers for batch sequencing with sequence-dependent setups, ORSAJC 7 (1995), 160–165.

[16] Y.H. Lee, K. Bhaskaran, and M. Pinedo, A heuristic to minimize the total weighted tardiness with sequence-dependent setups, IEE Trans 29 (1997), 45–52.

[17] J. Milan, The flow management problem in air traffic control: a model of assigning priorities for landings at a congested airport, Trans Plan Technol 20 (1997), 131–162.

[18] A.R. Odoni, J.M. Rousseau, and N.H.M. Wilson, "Models in urban air transportation," Operations Research and the Public Sector: Handbooks in Operations Research and Management Science, vol. 6, S.M. Pollock, M.H. Rothkopf, and A. Barnett (Editors), Elsevier, 1994, pp. 107–150.

[19] I.M. Ovacik and R. Uzsoy, Rolling horizon algorithms for dynamic parallel machine scheduling with sequence-dependent setup times, Int J Prod Res 33 (1994), 3173–3192.

[20] I.M. Ovacik and R. Uzsoy, Rolling horizon algorithms for single-machine dynamic scheduling problems with sequence-dependent setup times, Int J Prod Res 32 (1994), 1243–1263.

[21] H.N. Psaraftis, A dynamic programming approach to the aircraft sequencing problem, Research report R78-4, Flight Transportation Laboratory, MIT, 77 Massachussetts Ave., 33-412, Cambridge, MA 02139, 1978.

[22] H.N. Psaraftis, A dynamic programming approach for sequencing groups of identical jobs, Oper Res 28 (1980), 1347–1359.

[23] C.R. Reeves, Genetic algorithms for the operations researcher, INFORMS J Comput 9 (1997), 231–250.

[24] P.A. Rubin and G.L. Ragatz, Scheduling in a sequence dependent setup environment with genetic search, Comput Oper Res 22 (1995), 85–99.

[25] R.H. Storer, S.W. Flanders, and S.W. Wu, Problem space local search for number partitioning, Ann Oper Res 63 (1996), 465–487.

[26] R.H. Storer, K. Naphade, and S.D. Wu, Problem space search algorithms for resource constrained project scheduling, Ann Oper Res 20 (1997), 307–326.

[27] R.H. Storer, S.D. Wu, and R. Vaccari, New search spaces for sequencing problems with application to job shop scheduling, Mgmt Sci 38 (1992), 1495–1509.

[28] R.H. Storer, S.D. Wu, and R. Vaccari, Local search in problem and heuristic space for job shop scheduling, ORSA J Comput 7 (1995), 453–467.

[29] K.C. Tan and R. Narasimhan, Minimizing tardiness on a single processor with sequence-dependent processing times: A simulated annealing approach, Omega 25 (1997), 619–634.

[30] C.S. Venkatakrishnan, A. Barnett, and A.R. Odoni, Landings at logan airport: Describing and increasing airport capacity, Trans Sci 27 (1993), 211–227.

[31] H. Winter and H.G. Nuber (Editors), Advanced technologies for air traffic flow, Springer-Verlag, Berlin, 1994.