Optimal Scheduling with Maximum Position Shift (MPS) Constraints: A Runway Scheduling Application

Dr Dionyssios A. Trivizas

This paper presents results from a doctoral dissertation (reference 12), on the problem of scheduling with Maximum Position Shift (MPS) constraints. The problem typically arises in a queueing system, when the service rate depends on the sequence in which customers are handled by the server. Such a queueing system is the airport runway system, where it has been shown (references 3, 10, 12) that rearranging the sequence of landings and takeoffs affects the runway capacity – that is, the rate of runway operations. An optimal Dynamic Programming (DP) scheduling algorithm is developed that takes as input the set of currently known flights, their desired time of operation, basis of the First Come First Served (FCFS) sequence, and a matrix of Air Traffic Control (ATC) minimum time separations, in order to produce an optimal schedule defined as runway and threshold time assignments maximizing runway capacity.

1. INTRODUCTION. The Runway Scheduling Problem (RSP) has received a lot of attention over the last two decades from the FAA and research institutions. The RSP is important, because even small improvements in capacity of the order of 5 percent are critical for safety and fuel economy at times of congestion, when a minute of system delay translates to thousands of passenger minutes. It has been found (references 3, 10, 12) that optimal scheduling may increase capacity by up to 20 percent. A good runway schedule provides air traffic controllers a near-term plan that helps assign flights conflict-free spacetime paths in the airport terminal area. Moreover, a fast RSP solution may be used for airport capacity assessment in a global traffic management system.

In an ATC environment, the schedule changes dynamically in order to accommodate new flights wanting to use the runways. Therefore, solution speed is crucial to its real-time application, and this paper concentrates on the computational aspects of RSP.

Computationally, RSP is a variation of that notorious-for-its-complexity Travelling Salesman Problem (TSP), where a salesman seeks to find the shortest route to visit each of a number of cities once. This is because the minimum safe time separation (distance) between successive flights (cities) depends on their type, in our case characterised by the final approach speed and aircraft weight.

RSP is characterised by the Maximum Position Shift (MPS) constraints which limit the backward shift of a flight in the queue, so as to avoid the possibility of any flight being indefinitely postponed upon new arrivals.

In complexity theory, TSP belongs to the class of Non-deterministic Polynomial (NP) complete problems for which there are no exact polynomial

https://doi.org/10.1017/S0373463397007625 Published online by Cambridge University Press

algorithm solution. Solving optimally a TSP with n cities requires an amount of computation proportional to 2^n, prohibitive for real-time applications. Fortunately, the MPS constraints reduce the computation effort drastically, to a linear function of the number n of flights.

The paper develops the concept of the Permutation Tree (PT), the finest grain representation of the problem's solution space, comparing the two major approaches for searching for the optimal permutation: the Branch and Bound (BB) Depth First Search (DFS), and the Dynamic Programming (DP) Breadth First Search (BFS).

Despite the considerable pruning of the solution space by the MPS constraints, BB-DFS is a brute force method that repeats computations. It is shown here to sustain the prohibitive exponential worst-case performance. The DP approach, however, exploits the MPS structure, reducing drastically the computation effort to a linear function of n, with a constant of proportionality of 2^{MPS}. This is achieved through a systematic breadth first search of the solution space, to an incremental shortest-path fashion, using a sophisticated coding.

The solution space is shown to have a stage invariant structure, which collapses into a generalized cross-section. This cross-section is the backbone of the 'Parallel MPS Sequencer', a parallel computation engine that sweeps through the solution space with up to 2^{MPS} CPUs in lock step, in order to produce the optimal MPS feasible schedule in time linear to the number of flights.

The paper starts with definitions of the RSP, elaborating on the objective function, a mathematical expression of delay cost, examining its form and special features. These features derive from the generalisation of RSP that allows mixing takeoff and landings as well as operations on a system of interacting runways.

In both cases, the extended RSP cost matrix, containing the minimum safe time separations between successive operations on the same airport runway system, violates the triangular inequality, because it is usually possible to insert one or more takeoffs between successive landings on the same runway without stretching their separation. Similarly, one may be able to insert operations on a crossing runway. The paper shows how these features are accommodated by the parallel sequencer, which takes further account of the distance each flight has to travel within the airport terminal area, in its flight to runway assignments.

2. THE RUNWAY SCHEDULING PROBLEM (RSP). Runway scheduling has three mathematical models of increasing complexity: RSP1 allows only landings or only takeoffs on a single runway; RSP2 allows mixed takeoffs and landings on a single runway; and RSP3 allows multiple runways. Each of these problems can be extended to its dynamic or real-time version.

2.1. *Definition of RSP1*. In abstract terms, the runway scheduling problem can be stated as follows:

given:

(a) a set $S = \{i: 1 \leqslant i \leqslant n\}$ representing flights indices in the FCFS discipline;
(b) a set $P = \{i: 1 \leqslant i \leqslant n\}$ representing positions (or slots) in the queue;
(c) a set of Maximum Position Shift Constraints:

$$|i-j| \leqslant \text{MPS},\ i \in S, j \in P; \tag{1}$$

https://doi.org/10.1017/S0373463397007625 Published online by Cambridge University Press

(d) a cost matrix containing safe minimum time separations between consecutive flights on the same runway.

$$c_{ij},\ i, j \in S \tag{2}$$

find a MPS feasible permutation:

$$\pi_{(S,\,\mathrm{MPS})} = (i_1, i_2, \ldots, i_j, \ldots, i_n) \tag{3}$$

where $$i_j \neq i_k,\ i_j, i_k \in S \quad j, k \in P$$

that minimises:

$$Z(\pi_{(S,\,\mathrm{MPS})}) \tag{4}$$

Due to wake vortex considerations, whereby lighter aircraft stay further behind heavier ones, differences in speed and weight among aircraft make the cost matrix asymmetric. As a result, the time required to operate a set of aircraft, and therefore runway capacity, defined as rate of operations, depends on the traffic mix and sequence.

The simplest cost function equivalent to the length of a TSP tour, is the total time to operate the set of aircraft, or Time of the Last Operation (TLO), given by:

$$\mathrm{TLO} = Z(\pi_{(S,\,\mathrm{MPS})}) = \sum_{\substack{0 \leqslant k < n \\ i_k \in \pi_{(S,\,\mathrm{MPS})}}} c_{ik},\ i_{k+1} \tag{5}$$

where i_0 is the flight that has just operated.

Total Weighted Delay (TWD) is another cost function defined as the weighted sum of individual flight delays:

$$\mathrm{TWD}(\pi_{(S,\,\mathrm{MPS})}) = \sum_{i_k \in S} w_{i_k} d_{i_k} \tag{6}$$

$$d_{i_k} = \sum_{\substack{0 \leqslant m < k \\ i_m \in \pi_{(S,\,\mathrm{MPS})}}} c_{i_m},\ i_{m+1} - e_{i_k}, \quad 1 \leqslant k \leqslant n \tag{7}$$

where w_i and d_i are the weights (i.e. relative importance) and delay of flight i respectively, and e_i is the earliest possible time that flight i could arrive at the runway threshold.

2.2. *Definition of RSP2.* RSP2 is the generalisation of RSP1 that allows mixed takeoff and landing operations. The cost matrix c_{ij} may now violate the triangular inequality:

$$c_{ij} \leqslant c_{ik} + c_{kj}, \quad i, j, k \in S \tag{8}$$

if i and j are landings and k is a takeoff. It may thus be possible to insert one or more takeoffs between successive landings without stretching their minimum safe separation. This requires the following modification of the TLO and d_i:

$$z_k = \min_{\substack{0 \leqslant m < k < n \\ i_m \in \pi_{(S,\,\mathrm{MPS})}}} \{z_{k-m} + c_{i_k, i_{k-m}}\} \tag{9}$$

$$d_{i_k} = z_k. \tag{10}$$

https://doi.org/10.1017/S0373463397007625 Published online by Cambridge University Press

2.3. *Definition of* RSP3. Introducing operations on multiple runways produces RSP3. The cost matrix, augmented to take into account the runway of operation for each of flight, becomes:

$$c_{i,j,r_i,r_j},\ r_i, r_j \in R \tag{11}$$

where r_i and r_j are the runways assigned to consecutive flights i and j, and R is the set of available runways. Moreover, the solution has to produce a runway assignment for each flight thus introducing decision variables:

$$x_{ir} = \begin{cases} 1, & \text{if flight i is assigned to runway r} \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

Wake vortex effects are eliminated if the landings occur on crossing runways. Time separation has to account only for the time the first aircraft needs to clear the runway crossing point. This time is generally smaller than the inter-landing separation on the same runway, and as a result the triangular inequality is violated just as in RSP2.

2.4. *Static versus dynamic* RSP. In contrast to the static RSP versions, described above, in the dynamic RSP the complete set of flights is not known in advance. New flights must be accommodated as they appear over time. The optimal solution using static RSP may only be found at the end of a busy period, when all the flights are known; too late!

A simple dynamic algorithm updates the tentatively optimal schedule, resolving the static RSP periodically, upon accumulating a number of new requests. More sophisticated algorithms, accounting for anticipated traffic mix, may also be possible but are beyond the scope of this paper.

3. THE PRIORITY ASSIGNMENT GRAPH (PAG). The Priority Assignment Graph (PAG) is a bipartite graph of the form X^{MPS}_{n-d} defined as:

$$X^{\text{MPS}}_{n-d} = (S_{n-d}, P_{n-d}, A_{(n-d,\,\text{MPS})}) \tag{13}$$

where $S_{n-d} \subseteq S$ and $P_{n-d} \subseteq P$ are disjoint node sets with $n-d$ nodes. $A_{(n-d,\,\text{MPS})}$ is a set of arcs, showing the assignments feasible under the MPS constraints (1):

$$A_{(n-d,\,\text{MPS})} = \{a_{ij} : |i-j| \leqslant \text{MPS},\ i \in S_{n-d}, j \in P_{n-d}\} \tag{14}$$

For every value of d, $1 \leqslant d \leqslant n$, the MPS constraints allow X^{MPS}_{n-d} to have a number of possible types. Figure 1 shows the 'complete' type: $G^2_n = (S_n, P_n, A_{(n,2)})$.

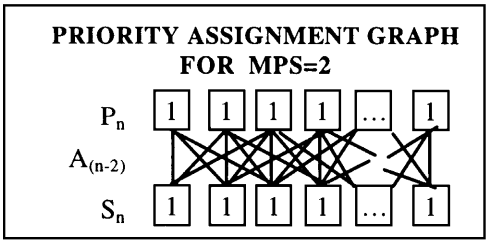


Fig. 1. A G^2_n priority graph

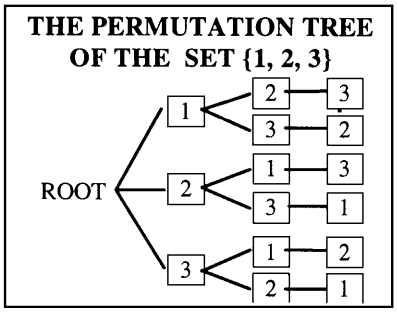


Fig. 2. The permutation tree

https://doi.org/10.1017/S0373463397007625 Published online by Cambridge University Press

The PAG is, by construction, a representation of the MPS-feasible solution space because every complete assignment, i.e. mating of the elements of S with the elements of P defines a distinct permutation $\pi_{(S,\,\mathrm{MPS})}$ in the FCFS neighbourhood.

4. THE PERMUTATION TREE (PT). The Permutation Tree (PT) is a complete representation of all permutations $\pi_{(S,\,\mathrm{MPS})}$. Figure 2 shows a sample PT of the set {1, 2, 3}. Moving on a path from the root to a leaf, defines incrementally a distinct permutation of S. Nodes at level d reflect ordered subsets with d elements. MPS-feasible nodes map on the set of the PAGS X^{MPS}_{n-d}. The PT is an important underlying concept, because it helps distinguish between the two types of search for the optimal permutation: the Depth First Search (DFS) and the Breadth First Search (BFS).

5. BRANCH AND BOUND (BB) DEPTH FIRST SEARCH (DFS) ON THE PT. A Depth First Search (DFS) solution branches down the PT, incrementally evaluating the cost function. When the accumulated cost exceeds the cost of the currently 'best' permutation, the search path is bounded ('pruning' the remaining sub-tree) and the solution backtracks in order to explore other permutations through un-visited (unmarked) nodes. When a leaf is encountered, the currently best permutation is updated.

The effort required by a BB-DFS solution may be estimated by examining a recursive step. This effort is shown, below, to be exponential in the number of flights for MPS values of 1 and 2.

5.1. *MPS = 1*. Figure 3 shows the recursive step on the evolution of a G^1_n graph. Assigning flight 1 to slot 1, we get G^1_{n-1}, a graph of the same type with $n-1$ nodes. However, assigning flight 1 to position 2, results in B^1_{n-1}, different than G^1_{n-1} where flight 2 may only be assigned to position 1, hence resulting in a G^1_{n-2}. Denoting by g^1_n the effort required to construct a full assignment with a G^1_n, we get from Figure 3 the following well known Fibonacci equation:

$$g^1_n = g^1_{n-1} + g^1_{n-2} + \mathrm{const.}$$

whose solution is $g^1_n = c\left(\frac{1+\sqrt{5}}{2}\right)^n$.

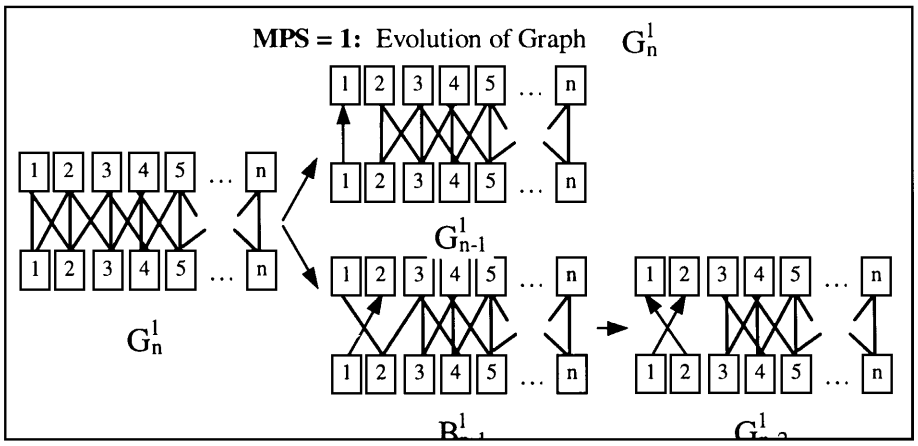


Fig. 3. PAG evolution with MPS = 1

https://doi.org/10.1017/S0373463397007625 Published online by Cambridge University Press

5.2. *MPS = 2.* As with the case for MPS = 1, Fig. 4 shows the recursive steps on the evolution of a G_n^2 where six graph types coded with letters *G*, *B*, *C*, *D*, *E* and *F* appear. One may safely assume that the BB search for the optimal assignment on the PT requires a worst-case computational effort of the form $g_n^{\mathrm{MPS}} = c(fn(\mathrm{MPS})^n$. This is an exponential function in the number *n* of available flights.

6. NUMBER OF PAG TYPES. Figure 3 shows two graph types X_n^1 for MPS = 1, i.e. $(X \in \{G, B\})$, whereas Fig. 4 reveals six graph types for X_n^2 for MPS = 2, i.e. $(X \in \{G, B, C, D, E, F\})$. In general the number of PAG types X_{n-d}^{MPS} is given by the following theorem:

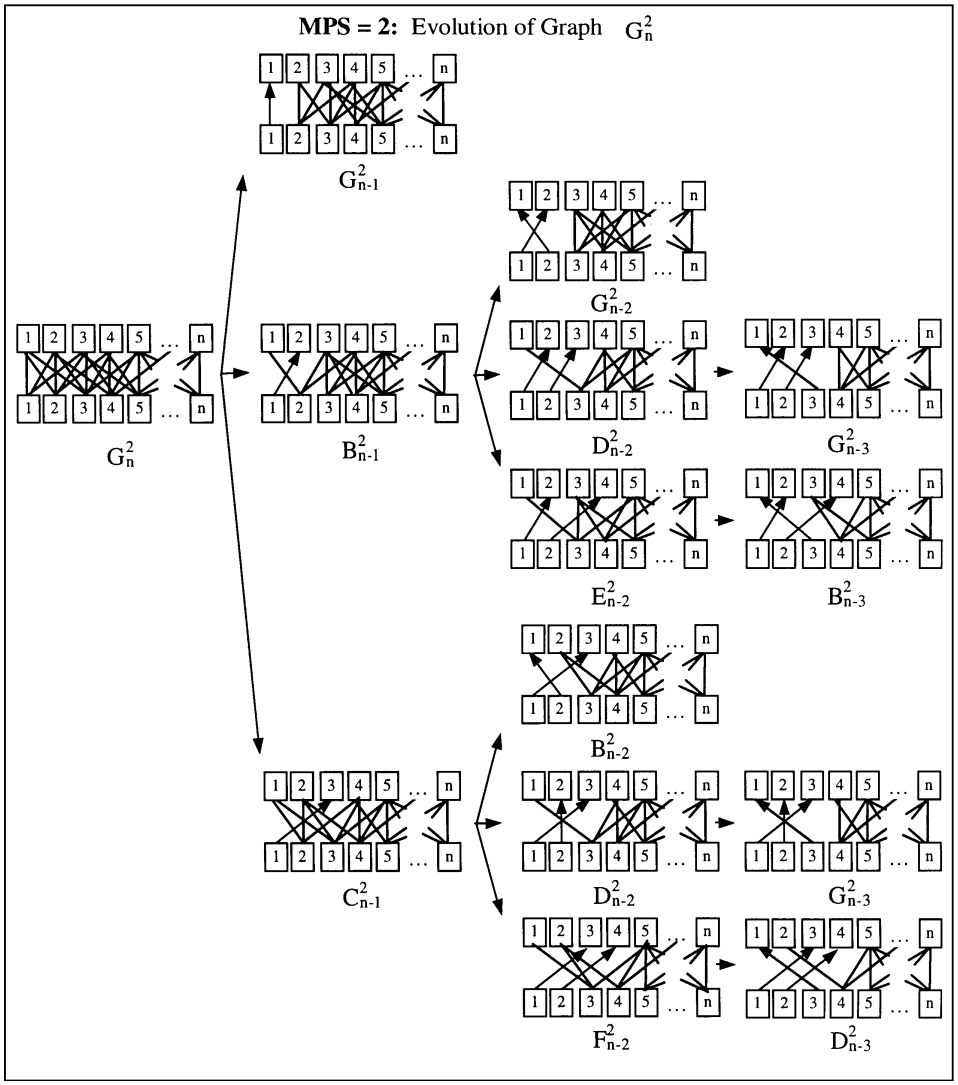


Fig. 4. PAG evolution with MPS = 2

https://doi.org/10.1017/S0373463397007625 Published online by Cambridge University Press

Theorem 1. The PAG types for given MPS = k, when a number d of flights have been selected, correspond uniquely to the MPS-feasible subsets of S, the set of available flights, with cardinality d. The number of these subsets is given by:

$$N(\text{MPS} = k) = {}_{2k}C_k = \frac{(2k)!}{(k!)^2} \tag{15}$$

where ${}_{2k}C_k$ is the well known binomial coefficient.

Proof. The PAG corresponds by construction to a MPS-feasible subset of S. In order to count the number of feasible subsets with d elements, consider the diagram in Fig. 5, showing flights in FCFS order. When stage d satisfies the condition '$n-k \leqslant d \leqslant k$' a feasible subset $S_d \subseteq S$ may be formed by choosing k out of the $2k$ elements in the range $(d-k+1,\ d+k)$ as shown in equation (15). Within each of these subsets, there may be up to $k+1$ optimal permutations (paths on the PT), each of them ending with a different flight j, such that:

$$d-k \leqslant j \leqslant d \quad \text{and} \quad i \neq j \ \square$$

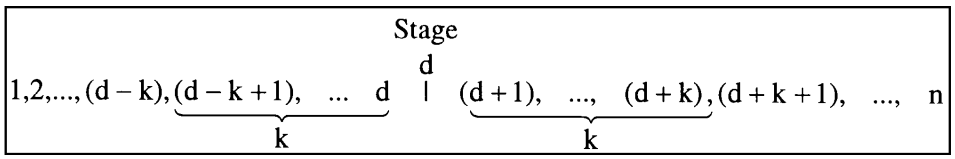


Fig. 5. MPS feasible partitions of S with d elements

7. CODING THE PAG: THE LABEL VECTOR. We now drop the arbitrary letter notation, introducing a systematic coding of the PAG and its corresponding feasible subset of S, with MPS-dimensional 'label-vectors' defined by Theorems 2 and 3 below. Each label-vector determines the elements of its corresponding subset $S_d \subseteq S$, and its evolution, as elements are added to it from its complement subset $\overline{S}_d$.

Theorem 2. There are ${}_{2k}C_k$ integer k-dimensional label-vectors of the form:

$$v = (v_1, v_2, \ldots, v_j, \ldots, v_k) \tag{16}$$

with

$$k \geqslant v_j \geqslant v_j + 1 \geqslant 0, \quad 0 < j < k \tag{17}$$

Theorem 3. The PAG types for MPS = k correspond uniquely to an equal number of label-vectors defined by equations (16) and (17).

7.1. 'CHESSBOARD PROOF'. The proof of Theorems 2 and 3 derives from the interpretation of label-vectors, as the position deficit of not-yet selected flights. This is illustrated using the 'Chess-board', a table PAG representation, as shown in Figs 3 and 4 for MPS values of 3 and 5 respectively. Chessboard columns are labelled by the remaining flights in $\overline{S}_d$, ordered in the FCFS discipline, and rows labelled by the remaining positions in the queue. Grey squares denote MPS-feasible flight-position assignments, and striped squares denote the position or 'opportunity' deficit.

7.1.1. *Position deficit.* To the left, on Fig. 6, the diagonally symmetric chessboard $v = (0, 0, 0)$ corresponds to a 'G-type' PAG. To the right, the chessboard $v = (3, 2, 1)$ at stage $d = 7$, with columns labelled by remaining flights $5, 7, 9, 11, \ldots$, has an asymmetric structure indicated by striped squares.

https://doi.org/10.1017/S0373463397007625 Published online by Cambridge University Press

Observe that flights 6, 8 and 10 are missing. They have been selected before flight 5 whose only feasible option is now position 8. The position deficit, thus, counts for every remaining flight how many later arrivals have been selected to land/takeoff before it. Similarly, flights 7 and 9 have two and one striped square respectively. Clearly, the MPS = k, only the first k remaining flights may have a non-zero position deficit, and therefore the label-vector has only k meaningful positions.

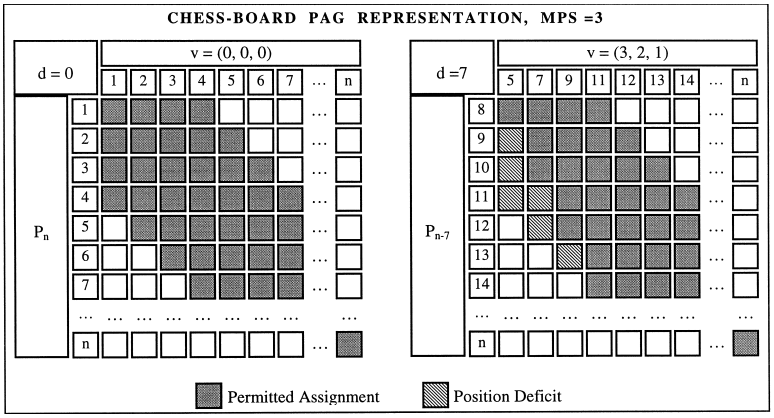


Fig. 6. Chess-board PAG representation

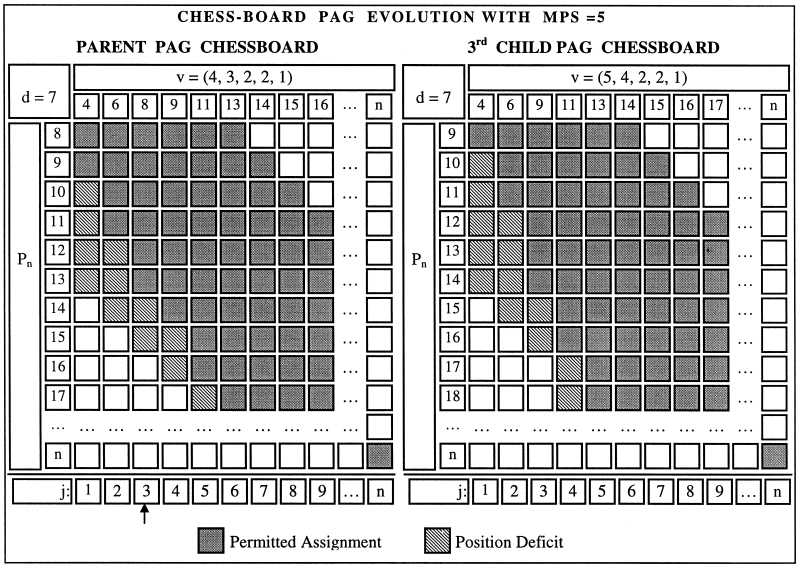


Fig. 7. PAG evolution with MPS = 5

7.2. RECOVERY OF AVAILABLE FLIGHTS. Figures 6 and 7 reveal the relation of the set of remaining flights with stage, and label vector. This relation is crucial in the search for the optimal permutation, since for every label-vector

https://doi.org/10.1017/S0373463397007625 Published online by Cambridge University Press

at a given stage, it recovers the remaining flight FCFS indices, allowing access to their information record. One may verify from Figs 6 and 7 that the index f_j, of the jth available flight, is given by:

$$f_j = d + j - v_j, \quad 0 \leqslant j \leqslant \text{MPS} \tag{18}$$

$$f_j \leqslant f_{j+1}, \quad f_j \in S \tag{19}$$

7.3. THE FLIGHT TO POSITION ASSIGNMENT PROCESS

The PAG children. The search for the optimal permutation proceeds by finding the optimal assignment for each of the remaining flights. Equation (18) allows us to identify each of the MPS + 1 candidate flights. Removing each of these flights will result in a different chessboard; that is, a different subset of S. Therefore, every chessboard may have at most MPS + 1 feasible descendants, or, 'children', whose label-vectors are given by the following theorem:

Theorem 4. The jth child v^j of a parent label vector v^p is given by:

$$v_i^j = \begin{cases} \left.\begin{matrix} v_i^P + 1, & 0 \leqslant i < j \leqslant \text{MPS} \\ v_{i+1}^P, & j < i \leqslant \text{MPS} \end{matrix}\right\}, & v_0^P < \text{MPS} \\ v_{i+1}^P, \quad 0 \leqslant i < \text{MPS}, & v_0^P = \text{MPS} \end{cases} \tag{20}$$

Proof. Selecting the jth child of label vector v^j means that the remaining flight f_j is chosen to operate next. Clearly flights f_k, with $k < j$, having smaller FCFS index than f_j (that is, have joined the queue before f_j), have an extra flight jump before them. Therefore their position deficit increases by 1. The remaining child label vector is filled with the position deficits of the remaining flights with FCFS index higher than f_j.

Clearly, equation (20) shows that when the first component of the parent label-vector v^P equals the value of MPS, the first remaining flight has been shifted backwards in the queue as far as possible. Choosing $j > 0$ would result in a child label vector whose first element would be larger than the maximum position shift.

Corollary. Label-vectors whose first component equals the MPS value have only one child. All others have MPS + 1 children.

7.4. THE LABEL VECTOR TREE (LVT). The label vectors may be ordered lexicographically. Moreover, Fig. 8 further reveals a tree structure showing a LVT for MPS = 3. Tracing the path from each leaf to the root of the LVT produces the components of a distinct label-vector. Note that the LVT for MPS = 3 contains the LVT's for lesser MPS values, indicated by the shading in Fig. 8. The LVT is more than a mathematical curiosity, in that it saves memory in storing LVT's for large values of MPS.

8. THE SOLUTION OR STATE SPACE. As explained earlier, a path from the root to some node at level d of the PT, corresponds uniquely to a distinct ordering of some subset S_d. Lumping all the PT nodes with the same S_d into a 'state of computation' $X(d, v)$, produces the solution space, where the solution proceeds in an incremental way, eliminating a vast number of paths, keeping only optimal ones.

https://doi.org/10.1017/S0373463397007625 Published online by Cambridge University Press

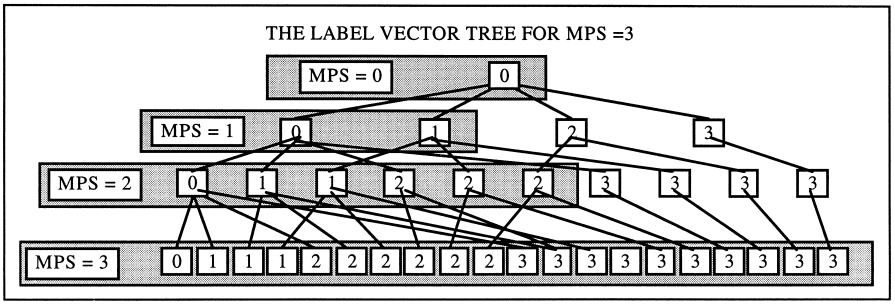


Fig. 8. The label vector tree

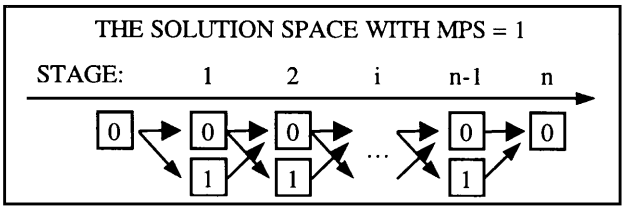


Fig. 9. Solution spaces for MPS = 1

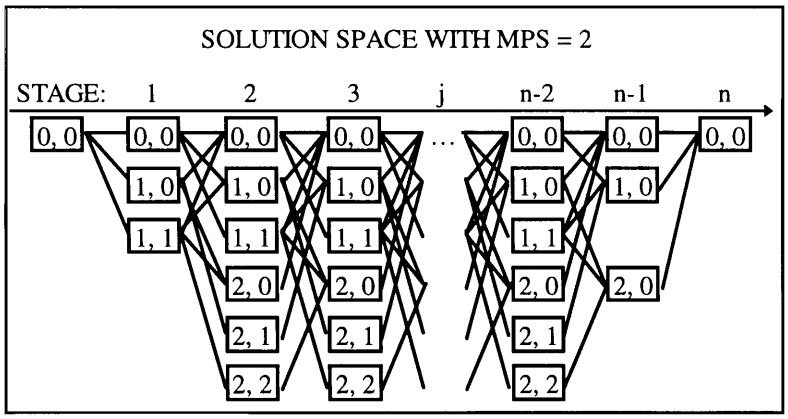


Fig. 10. Solution spaces for MPS = 2

Figures 9 and 10 show the solution space for MPS values of 1 and 2. Label-vectors code the states of computation, thus providing storage for intermediate results. Theorem 4 determines the arcs in the solution space, which correspond to remaining flights whose indices are given by equation (18). The MPS constraints limit the number of arcs incident to a state to, at most, MPS + 1.

The solution space is organised in columns of states, called stages of computation. The number of states per stage expands in the first MPS stages from 1 to a maximum of ${}_{2k}C_k$, and contracts in the last MPS stages back to 1.

https://doi.org/10.1017/S0373463397007625 Published online by Cambridge University Press

Incidentally, in a dynamic environment, where new flights are added to the FCFS sequence at random times, the state space may be viewed as a part of an infinite state space. As a matter of curiosity, notice that there is a 'dual' to the label vector, corresponding to the complement subset, and to theorem 4, that move backwards (right-to-left) on the solution space.

9. SOLUTION BY INCREMENTAL SHORTEST PATHS. If we think of permutations as pointed lists, each arc coming into a state $X_{(d,v)}$ is a pointer to an optimal permutation of the same subset, whose last element is determined by the origin of the arc and equation (18). One may, therefore, use the notation $(f_i, \bar{X}_{(d,v)})$ to describe each of the incoming arcs to that state where $f_i \in \bar{X}_{(d,v)}$ the complement of $X_{(d,v)}$.

At each stage incoming paths are augmented by one flight. At each state, for each remaining flight $r_j = \in X_{(d,v)}$, corresponding to the jth outgoing arc, we produce an optimal permutation $(r_j, \bar{X}_{(d+1,v^j)})$ by appending r_j to the optimal input path $(f_i^*, \bar{X}_{(d,v)})$, by choosing i^* that minimises over i the cost:

$$\min_i \{\text{cost}_{(f_i, \bar{X}_{(d,v)})} + c_{f_i, r_j}\} \tag{21}$$

At the end of stage $d = n$, the complete optimal path is traced backwards in the solution space. To help understand this process, we draw an analogy to the incremental shortest paths method. The next flight r_j may be viewed as a city for which we want to find its shortest path to the origin, the root of PT in our case. Input permutations $(f_i, \bar{X}_{(d,v)})$ may be viewed as shortest paths to all the 'neighbouring cities' of r_j. To find the shortest path from r_j to the root, we need only consider, for each neighbour, the sum of the distance of the neighbour to the origin, plus the distance of the neighbour to r_j.

10. THE PARALLEL SEQUENCER. Notice in Figs 9 and 10:

(a) that at each stage, the state computations are independent of each other and may be performed asynchronously and/or in parallel.
(b) the 'stage invariance' of the solution space, whereby label vectors and arc structure remain the same from one stage to the next.

We may therefore use a computational engine, modelling the generalised cross-section of the solution space, containing only two stages (columns), at a time, labelled as 'current' and 'next'. Stage current contains pointers to the optimal paths with d flights, used to compute optimal paths with $d+1$ elements collected in stage 'next'.

This engine, the 'Parallel Sequencer', shown in Figure 11 for MPS = 2, may use up to ${}_{2k}C_k$ processors, crunching in parallel, to produce the optimal permutation in time linear to n, the number of available flights.

Figure 12 shows a table representation of the Parallel Sequencer for MPS = 3, as an indexed array of label vectors and 'addressee list'; that is, indices modelling the arcs of the solution space.

10.1. *Garbage collection*. The Parallel Sequencer generates a backward pointed tree of 'schedule records' that gets pruned along the way to a single leaf.

The vast number of generated intermediate records which depends on the

https://doi.org/10.1017/S0373463397007625 Published online by Cambridge University Press

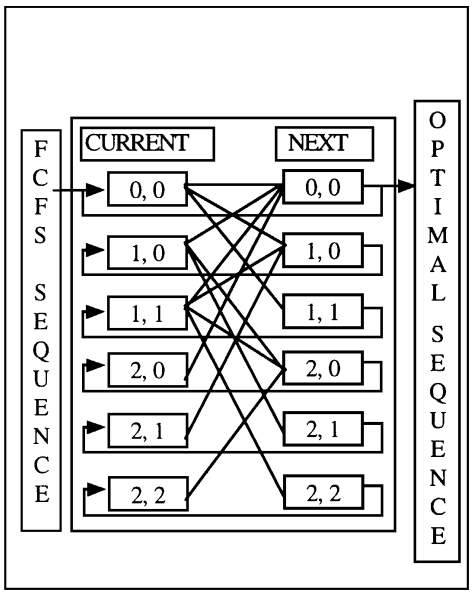


Fig. 11. Parallel sequencer for MPS2

INDEX	v	ADDRESSEE LIST
0	(0, 0, 0)	0, 1, 2, 3
1	(1, 0, 0)	0, 4, 5, 6
2	(1, 1, 0)	1, 4, 7, 8
3	(1, 1, 1)	2, 5, 7, 9
4	(2, 0, 0)	0, 10, 12, 12
5	(2, 1, 0)	1, 10, 13, 14
6	(2, 1, 1)	2, 10, 13, 15
7	(2, 2, 0)	4, 10, 16, 17
8	(2, 2, 1)	5, 11, 16, 18
9	(2, 2, 2)	7, 13, 16, 19
10	(3, 0, 0)	0
11	(3, 1, 0)	1
12	(3, 1, 1)	2
13	(3, 2, 0)	4
14	(3, 2, 1)	5
15	(3, 2, 2)	7
16	(3, 3, 0)	10
17	(3, 3, 1)	11
18	(3, 3, 2)	13
19	(3, 3, 3)	16

Fig. 12. Parallel sequencer for MPS3

number of flights considered for scheduling and the value of MPS, may exhaust the computer memory resources. Fortunately, a recursive 'garbage collection' algorithm may be used to reduce memory requirements by recycling schedule records after every stage of computation.

11. THE TRIANGULAR INEQUALITY. The definitions of RSP2 and RSP3 introduce the triangular inequality in the matrix of minimum time separations. The situation is depicted in Figure 13, showing schematically a takeoff (T) fitting loosely between two consecutive landings, a heavy L_1 and a light L_3.

The structure of the Parallel Sequencer deals with this situation by allowing more than one 'optimal paths' to be transmitted along each arc of the solution space. Theoretically, this is equivalent to expanding the definition of the RSP1 optimal 'sub-path' from $(f_i, \bar{X}_{(d,v)})$ to:

$$\frac{(L_1, T_1, T_2, \ldots, T_j, L_2,}{\text{permutation-suffix}} \quad \frac{\bar{X}_{(d,v)})}{\text{remaining flights}} \tag{22}$$

that is, the state is defined by the ordering of an arbitrary number j of takeoffs up to a previous landing. Notice, that this definition (22) of optimal path in $\bar{X}_{(d,v)}$ implies that there is a secondary optimisation taking place between successive landings on the same runway, that puts the included takeoffs in the right order. The incremental cost function used in this case was given above in equations (9) and (10).

https://doi.org/10.1017/S0373463397007625 Published online by Cambridge University Press

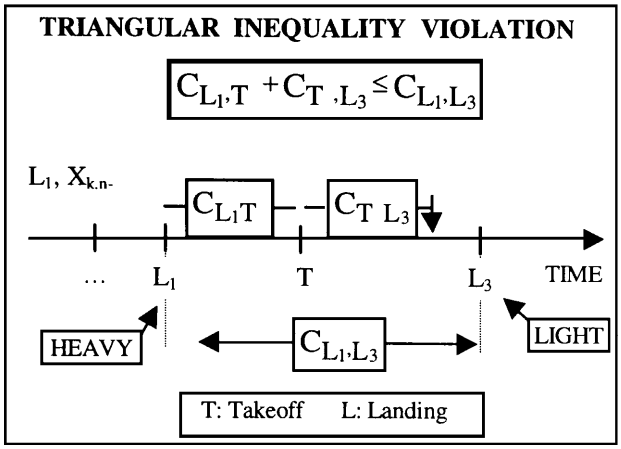


Fig. 13. Triangular inequality

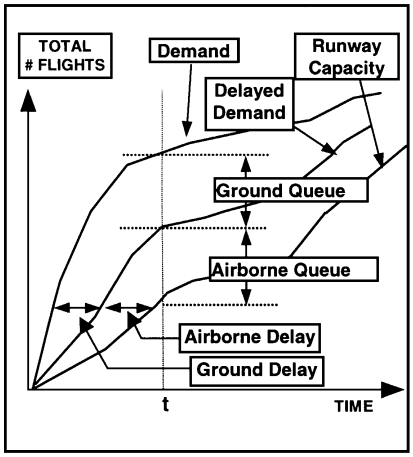


Fig. 14. Schematic of the flow management problem

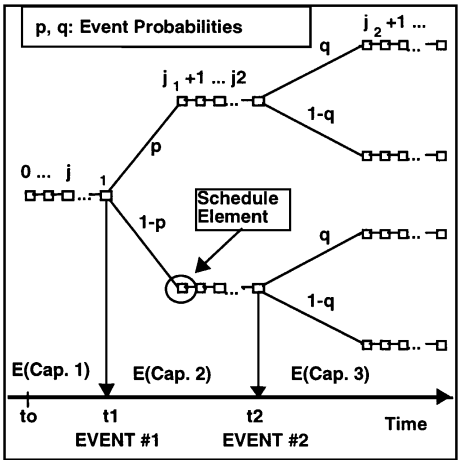


Fig. 15. Probabilistic runway capacity prediction based on anticipated events

12. FLOW MANAGEMENT AND AIRPORT CAPACITY PREDICTION. In a system of inter-related airports, flow management tries to keep the runways busy by a controlling the supply of traffic so that:

– it can follow the random fluctuations in runway rate of operations, while
– keeping as few flights as possible in the Terminal Manoeuvring Area (TMA)

This is a formidable problem. The top curve on Fig. 14 shows schematically the total number of flights that would like to use the airport runway system. During peak hours, this may greatly exceed the number that can safely operate for given separation minima.

Flow management spreads the queue all the way to the departure airport, admitting in the TMA enough flights, middle 'delayed demand' curve, so as to

https://doi.org/10.1017/S0373463397007625 Published online by Cambridge University Press

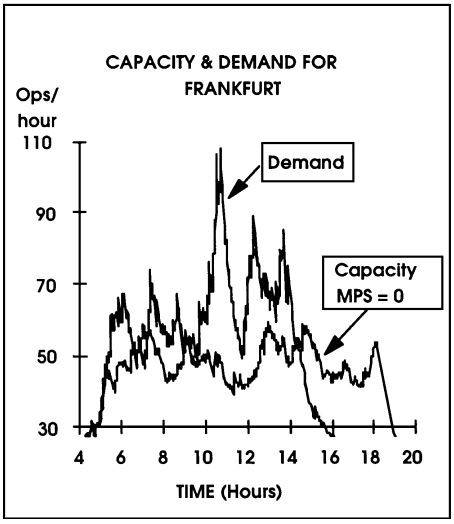


Fig. 16. Demand and runway capacity without optimisation

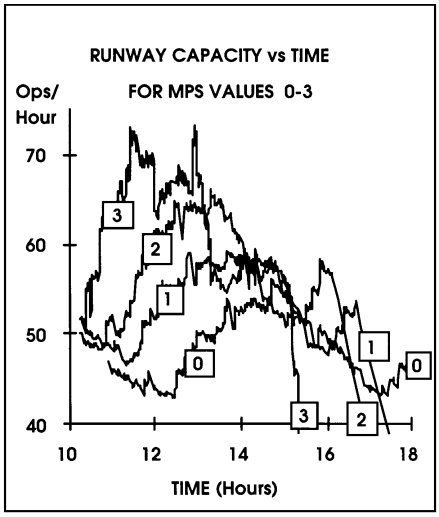


Fig. 17. Runway capacity at various levels of optimisation

avoid the risk of starving the runways, while exploiting runway capacity fluctuations, and saving fuel. Clearly, flow management needs good runway scheduling as well as an airport capacity prediction model.

As shown in Fig. 15, with reasonably fast computers we can make accurate, average or worst-case capacity predictions, exploiting our *a priori* knowledge on the probability of occurrence of anticipated events.

Using the fast runway scheduler one may create a schedule tree which will bifurcate at every anticipated event at time, t_i. Based on this schedule tree, one may be able to compute average and worst-case capacity.

13. SIMULATION RESULTS. We present here results from a runway capacity study conducted with a prototype Global Air Traffic Simulation 'TMSIM' using real traffic and airport data, from EUROCONTROL and the FAA in the US. The study, entitled 'TMSIM: A Runway Capacity Study for the Frankfurt and Chicago O'Hare Airports' is at Reference 13.

We show here only results concerning Frankfurt. The TMSIM run for Frankfurt used actual data for all European airports, whereas intercontinental traffic appeared at Europe's boundaries.

The demand data are a busy day's worth of actual flight plans. For each flight plan a flight object is generated and scheduled for departure, en-route and arrival, through a global traffic management routine that uses the Optimal Parallel Sequencer presented in this paper. The results are based on recorded runway threshold times, and desired operation times, averaged through a sliding time-window technique.

The curves, to the left on Fig. 16, show the runway capacity (rate of operations) response to a time-scaled demand curve without optimisation. Time scaling was used to condense the demand curve without disturbing its orientation.

https://doi.org/10.1017/S0373463397007625 Published online by Cambridge University Press

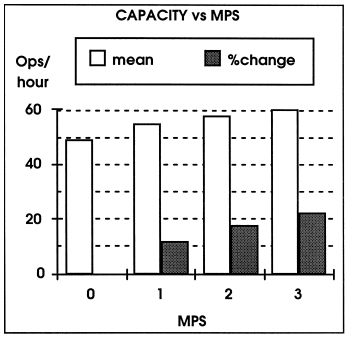


Fig. 18. Mean capacity improvements of 12, 18 and 22%

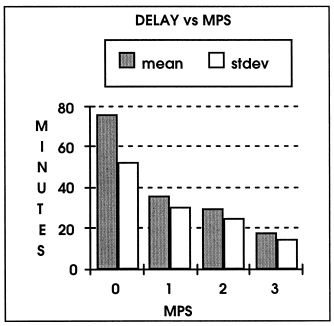


Fig. 19. Delay reductions

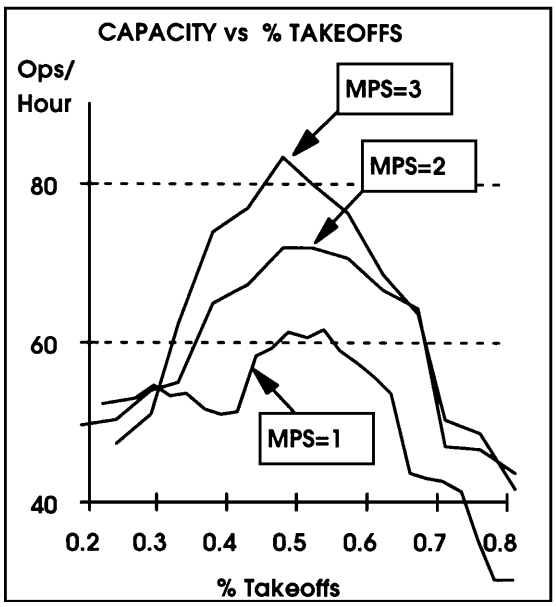


Fig. 20. Runway capacity as a function of take-off content

The curves to the right on Fig. 17 show the capacity improvements that result from optimal runway scheduling, contrasting the performance of scheduling based on the First Come First Served (FCFS) (MPS = 0) discipline, with that for MPS values of 1, 2 and 3.

Figure 18 shows mean capacity improvements of 12 percent, 18 percent and 22 percent for MPS values of 1, 2 and 3 respectively.

Figure 19 shows the impressive delay reduction which follows the '1/x' behaviour with increasing value of MPS.

Finally, Fig. 20 plots runway capacity versus traffic mix expressed as percentage takeoffs. It is interesting to see that the runway capacity curves are bell-shaped and reach their peak value at the balanced traffic mix of 50 percent

https://doi.org/10.1017/S0373463397007625 Published online by Cambridge University Press

takeoffs. The 50 percent takeoffs point is also where the optimization is more effective.

14. CONCLUSIONS. The static runway scheduling problem has been thoroughly analysed, producing an optimal, fast and parallel runway scheduling algorithm.

Simulation results obtained with real traffic data and airport configuration are very encouraging, showing that even a modest values MPS = 3 produces up to 20 percent increase in capacity.

Further work is needed in order to:

1. Study the dynamic versions of RSP.
2. Get more insight into the real problem through simulation studies.
3. Study the capacity prediction potential, using the RSP solution. Such a tool is useful for global air traffic coordination.
4. Examine the possibility of hardware implementation for the parallel RSP scheduler using 2MPS CPUS.
5. Study the potential application of the Parallel Sequencer, given its efficiency of computation, to other scheduling and routing problems with time-window constraints.

REFERENCES

[1] Aho, A. V., Hopcroft, J. E. and Ullman, J. D. (1974). *The Design and Analysis of Computer Algorithms*. Addison-Wesley.

[2] Bertsekas, D. P. *Dynamic Programming: Deterministic and Stochastic Models*. Prentice Hall Inc., Englewoods Cliffs, New Jersey.

[3] Dear, Roger George (1976). Dynamic scheduling of aircraft in the near terminal area, *MIT Flight Transportation Laboratory*. *FTL-report R76-9*.

[4] Liew, C. L. (1968). *Introduction to Combinatorial Mathematics*. McGraw Hill.

[5] Odoni, A. R. (1969). An analytical investigation of air traffic in the vicinity of terminal areas. *MIT*, *Operations Research Center*, *Technical Report no. 46*, *Dec.*

[6] Odoni, A. R. The flow management problem in Air Traffic Control. In *Flow Control of Congested Networks* (A. R. Odoni, L. Bianco, G. Szego eds.), pp. 269–288. Springer-Verlag, Berlin.

[7] Papadimitriou and Steiglitz (1982). *Combinatorial Optimization – Algorithms and Complexity*. Prentice Hall.

[8] Pararas, John D. (1982). Decision support systems for automated terminal area air traffic control. *MIT*, *Flight Transportation Laboratory*, *FTL R82-3*.

[9] Psaraftis, Harilaos (1983). An exact algorithm for the single vehicle many-to-many dial a ride problem with time windows. *Transportation Science*, **17**.

[10] Psaraftis, Harilaos (1978). A dynamic programming approach for the aircraft scheduling problem. *MIT*, *Flight Transportation Laboratory*, *FTL Report R78-4*.

[11] Simpson, Robert W. (1982). Critical examination of factors which determine the operational capacity of runway systems at major airports. *MIT*, *Flight Transportation Laboratory Report*.

[12] Trivizas, Dionyssios A. (1987). *Parallel parametric combinatorial search – its application to runway scheduling*. Doctoral Dissertation, MIT. Flight Transportation Laboratory.

[13] Trivizas Dionyssios A. (1994). TMSIM: a runway capacity study for Frankfurt and Chicago O'Hare airports. This *Journal*, **47**, 70.

[14] Terrab, M. (1990). *Ground holding strategies for air traffic control*, Doctoral Dissertation, MIT, Operations Research Center.

https://doi.org/10.1017/S0373463397007625 Published online by Cambridge University Press

[15] Wang, H. (1989). An exact dynamic programming approach to the overall flow control problem in air traffic management. *Working paper*, Wellesly College, Wellesly Massachusetts.
[16] *National Airspace System Plan* (1988). FAA, US, Department of Transportation.

KEY WORDS

1. Air Traffic Control. 2. Management. 3. R&D.

https://doi.org/10.1017/S0373463397007625 Published online by Cambridge University Press