A DYNAMIC PROGRAMMING APPROACH

TO THE AIRCRAFT SEQUENCING PROBLEM

by

Harilaos N. Psaraftis

FTL Report R78-4
October 1978

Flight Transportation Laboratory
Massachusetts Instittue of Technology
Cambridge, Massachusetts

A DYNAMIC PROGRAMMING APPROACH TO

THE AIRCRAFT SEQUENCING PROBLEM

ABSTRACT

In this report, a number of Dynamic Programming algorithms for three versions of the Aircraft Sequencing problem are developed. In these, two alternative objectives are considered: How to land all of a prescribed set of airplanes as soon as possible, or alternatively, how to minimize the total passenger waiting time. All these three versions are "static", namely, no intermediate aircraft arrivals are accepted until our initial set of airplanes land. The versions examined are (a) The single runway-unconstrained case, (b) The single runway-Constrained Position Shifting (CPS) case and (c) The two-runway-unconstrained case. In the unconstrained case, no priority considerations exist for the airplanes of our system. By contrast, CPS prohibits the shifting of any particular airplane by more than a prespecified number of positions (MPS) from its initial position in the queue. All three algorithms exploit the fact that the airplanes in our system can be classified into a relatively small number of distinct categories and thus, realize drastic savings in computational effort, which is shown to be a polynomially bounded function of the number of airplanes per category. The CPS problem is formulated in (b) in a recursive way, so that for any value of MPS, the computational effort remains polynomially bounded as described above.

All algorithms of this work are tested by various examples and the results are discussed. Implementation issues are considered and suggestions on how this work can be extended are made.

ACKNOWLEDGEMENTS

This work was supported in part by the Federal Aviation Administration under contract DOT-FA77WA-4007.

The author is particularly indebted to Professors Amedeo Odoni, Christos Papadimitriou and John Devanney for their continuous assistance. Thanks are also due to Professor Robert Simpson for his comments.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont'd)

LIST OF FIGURES

LIST OF FIGURES (CONT'D)

LIST OF FIGURES (CONT'D)

LIST OF TABLES

INTRODUCTION AND OUTLINE

This report is excerpted from the author's Ph.D. dissertation "Dynamic Programming Algorithms for Specially Structured Sequencing and Routing Problems in Transportation" (Department of Ocean Engineering, M.I.T., September 1978). Thus, Chapters 1 to 6 as well as Appendices A to D of the report are essentially identical to those of the dissertation. Chapter 7 of the report corresponds to Sections 11.0 to 11.5 of Chapter 11 of the dissertation and Appendix E of the report covers the first half of Appendix E of the dissertation.

The main purpose of this work is to investigate the problem of sequencing aircraft landings at an airport, or what we shall call the Aircraft Sequencing Problem (ASP). This is a very important problem at the present time, and is currently under investigation by a number of organizations (FAA, NASA, etc.).

Our investigation encompasses the development of analytical models describing the above real-world problem, as well as the design, testing and refinement of novel, specialized and efficient solution procedures tailored to specific versions of this problem. It also includes interpretation of the results of the above procedures and discussion of implementation issues as well as of directions for further research.

In addition to the above, this investigation provides an opportunity to relate this research to some currently "hot" theoretical issues in the areas of computational complexity and algorithmic efficiency and to illustrate their importance in the area of combinatorial optimization.

From a methodological point of view, the thrust of this work is on exact and rigorous optimization approaches rather than on heuristics. This is not to be interpreted as a lack of interest for this rapidly growing area of optimization, but rather as an attempt to investigate the potential savings that may result from exact, specialized solution procedures which exploit the special characteristics (or, what we shall call in this report, the special structure) of the problems at hand.

Now that this attempt has been concluded, our findings in this respect are, we believe, interesting and significant.

The material in the report is organized into four parts:

a) Part I presents the necessary background for the investigation to follow.

b) Part II is devoted to the development of Dynamic Programming algorithms for several versions of the ASP.

c) Part III reviews the results of this work and suggests directions for further research.

d) Finally, Part IV includes several appendices with additional technical material on the ASP and a description of the computer programs implementing the algorithms developed in Part II.

Specifically, the organization of the report goes as follows:

1) Part I: General Background

In Chapter 1, we introduce and review the fundamental issues concerning the computational efficiency of algorithms. As a first step, we discuss the generally established classification of algorithms into "efficient" (or polynomial") and "inefficient" (or "exponential"). According to this classification, an "efficient" algorithm can solve a given problem in running time which is a polynomially bounded function of the size of the problem's input. We argue that for problems of sufficiently large size, "inefficient" algorithms are not likely to be of any practical use. As a second step, we refer to the class of notoriously difficult problems in combinatorial optimization which are known as "NP-complete".

For these problems, no "efficient" algorithms are known to exist, but also nobody to date has been able to prove the impossibility of such algorithms. We discuss several remedies to deal with such problems, namely heuristics and exploitation of special structure, if such a structure exists.

Chapter 2 formulates a famous NP-complete problem, the Travelling Salesman Problem (TSP) and presents a well-known Dynamic Programming approach to solve it. The TSP will be seen to constitute a "prototype" for the Aircraft Sequencing Problem we examine in subsequent chapters. As expected, the D.P. algorithm for the TSP is an "exponential" algorithm. Also, it is not necessarily the best way to solve this problem in its general form. It possesses, however, certain characteristics which will be exploited in our specially structured problem later. At this point, we compare the D.P. approach with some other well-established approaches for the solution of the TSP.

2) Part II: The Aircraft Sequencing Problem (ASP)

In Chapter 3 we formulate an important real-world problem which derives from the TSP and exhibits a special structure that will eventually enable us to solve it in a viable way, the Aircraft Sequencing Problem (ASP). We describe the physical environment of the problem, that is the near terminal area around airports and we introduce the ASP as a decision problem of the air traffic controller. This problem consists of the determination of an appropriate landing sequence for a set of airplanes waiting to land, so that some specific measure of performance is optimized. The special structure of the problem is due to the fact that while the total number of airplanes may be substantially large, these can be "classified" into a relatively small number of "categories" (e.g. "wide-body jets," "medium-size jets," etc.). Another characteristic of the problem, which in fact makes it non-trivial, is that the minimum permissible time interval between two successive landings is far from being constant for all pairs of aircraft categories. Two alternative objectives are considered: Last Landing Time (LLT) minimization and Total Passenger Delay (TPD) minimization. LLT minimization implies that all existing aircraft land as soon as possible. TPD minimization is concerned with minimizing the sum of the "waiting-to-land" times for all passengers in our system, an objective which is identical to the minimization of the average-per-passenger delay. The version of the problem which is described in this chapter, is "static," namely no aircraft arrivals are permitted (or taken into consideration if they occur) after a point in time $t=0$.

In Chapter 4 we develop a modified version of the classical D.P. algorithm for the TSP (that was presented in Chapter 2) to solve the unconstrained case of the single runway ASP. Unconstrained means that the air traffic controller is not (for the moment) bothered with priority considerations, being free at any step to assign the next landing slot to any of the aircraft not landed so far. The algorithm we have developed evaluates with equal ease either one of the alternative objectives we have suggested in Chapter 3. More importantly however, drastic savings in computational effort are achieved. These savings result from our taking advantage of category classifications. The algorithm exhibits a running time which is a polynomial function of the number of aircraft per category. It is an exponential function of the number of distinct categories but for the ASP this number is small (3 or 4 or, at most, 5). The computer outputs for several cases for which the algorithm was tested, exhibit sufficiently interesting patterns to stimulate further investigation concerning the underlying structure of the ASP. Thus, while in most cases, all the aircraft of a given category tend to be grouped in a single "cluster", in other seemingly unpredictable cases this pattern is upset. There are also cases where seemingly negligible changes in the problem input, produce global changes in the optimal sequence. An extensive investigation of these interesting phenomena is left for Appendices A through D.

In Chapter 5, we introduce priority considerations into our problem. Specifically, the Constrained Position Shifting rules are introduced. According to these, the air traffic controller is no longer free to assign the next landing slot to any of the remaining aircraft, but is limited to shift the landing position of any aircraft up to a maximum of a prespecified number of single steps upstream or downstream from the initial position of the aircraft in the original first-come, first-serve sequence of arrivals in the vicinity of the near terminal area. This number is known as Maximum Position Shift (MPS) and it is a new input to our problem. An additional input is the (ordered) initial sequence of aircraft. The problem is again assumed "static" with the same alternative objectives as in the unconstrained case. A new D.P. algorithm is developed incorporating our priority constraints in a way specially suited to the recursive nature of our approach. By contrast to existing complete enumeration procedures which can deal with CPS problems only for small values of MPS, our algorithm can solve the CPS problem for any value of MPS and still remain within polynomially bounded execution times with respect to the number of aircraft per category. Computer runs for several cases of this problem have been performed and the results are discussed.

Chapter 6 formulates the problem of sequencing aircraft landings on two identical and parallel runways. We consider again the "static" unconstrained case. The minimization of LLT is seen to constitute a <u>minimax</u> problem. Our alternative objective is, as before, to minimize TPD. We see that this problem essentially involves the <u>partitioning</u> of the initial set of airplanes between the two runways, as well as the subsequent sequencing of each of the two subsets to each of the two runways, independently of one another. The algorithm we have developed for this problem is a post-processing of the table of optimal values created by a single pass of the unconstrained single runway D.P. algorithm (Chapter 4). Despite the fact that we solve the partitioning subproblem by complete enumeration of all possible partitions, the computational effort of the algorithm remains a polynomial function of the number of aircraft per category. Computer runs of this algorithm show some interesting partitioning and sequencing patterns. Specifically, while in some cases the composition of aircraft is more or less the same for the two runways, in other cases the partition becomes completely asymmetric. We discuss these and related issues at the end of the chapter.

3) <u>Part III: Final Results</u>

In Chapter 7, we review the main results of this work, suggest directions for possible extensions and address issues on the implementation of the developed algorithms. In particular, the "dynamic" version of the ASP is seen to constitute a very important extension to this work.

4) Part IV: Appendices

These appendices present additional technical issues related to the ASP. They are organized as follows:

a) Appendix A: Investigation of group "clustering" in the ASP.

b) Appendix B: Derivation of the elements of the time separation matrix in the ASP.

c) Appendix C: Investigation of certain properties of the time separation matrix in the ASP that are connected to group clustering.

d) Appendix D: Development of some equivalence transformations in group clustering and in the case of variable numbers of passengers per aircraft category.

e) Appendix E: Review of the computer programs used for the ASP.

PART I

GENERAL BACKGROUND

CHAPTER 1

ALGORITHMS AND COMPUTATIONAL EFFICIENCY

1.0 Introduction: Performance Aspects of Algorithms

An algorithm is a set of instructions for solving a given problem. These instructions should be defined in such a way, so that the following are satisfied:

1) The instructions should be precisely-defined and comprehensive. Thus, nothing should be left to intuition or imagination, and there should be no ambiguity or incompleteness.

2) The corresponding algorithm should be able to solve not just one, but all of the infinite variety of instances of the given kind of problem.

A very simple example of an algorithm is the procedure we use to multiply two numbers. Although not often realized, this procedure consists of a sequence of steps, each dictating in exact detail the actions that we should take to solve the problem. Using this procedure we can determine the product of any given pair of numbers.

The nature of an algorithm is such, that it is convenient to represent it in computer language. This convenience, together with the rapid growth in the science and technology of the computer during recent years, have resulted in an increasing effort toward the design and analysis of computer algorithms [AHO 74]*, as well as to the development of solution procedures

*References are listed lexicographically at the end of the report for several kinds of problems, that are specifically tailored to computer programming. An example of this approach to optimization and related problems concerning graphs and networks is the excellent work of Christofides [CHRI 75].

The interaction between the theoretical development of an algorithm on the one hand and its computer implementation on the other has been so strong, that it has now become almost impossible to address the first issue without thinking about the other as well. In fact, a synonym for "algorithm" has often been the word "program," which, besides its primary meaning as a set of instructions in computer language, ended up also being used to describe general as well as specific analytical methodologies: Thus, in the area of optimization we have Mathematical programming, Linear programming, Dynamic programming, etc.

It is conceivable that more than one different algorithms can solve the same problem*. In our multiplication problem, for instance, we can find the product of, say, 24 by 36 by adding 36 times the number 24. We can, of course, apply also our well known multiplication procedure and get the same answer much faster. This elementary example illustrates the fact that certain algorithms are <u>better than others</u> for the solution of a given problem.

For problems of considerable difficulty the above fact becomes very important. For such problems, it is much less important to devise an algorithm that just solves the problem, than to find an algorithm that

*It is also conceivable that no algorithm can be devised for certain problems. This was first demonstrated by Turing in the 1930's [TURI37].

does this <u>efficiently</u>. This is true for all optimization problems and in particular for those where the importance of being able to obtain an optimal solution fast is very high or even crucial.

Subsequent parts of this thesis will be devoted to the examination of such problems and the need for powerful and efficient algorithms will be seen to be apparent. For the moment, we shall introduce some issues regarding algorithmic performance.

Temporarily avoiding being explicit on what is an "efficient" algorithm, we start by examining a hypothetical situation: Suppose we have two different algorithms, A and B, for solving the same instance of a given problem P. A plausible comparison between A and B would be to run both algorithms on the same machine and choose the one exhibiting the smallest running time (or cost). The disadvantage of this approach is that, conceivably, this comparison will yield different preferences for problems of different size. For example, if the size of the input to P is n (e.g. the number of nodes in a graph*) it may happen that the running time of algorithm A is equal to $10 \cdot n$ and that of B equal to 2^n. Then according to the above selection criterion, algorithm B is better than A for $n \leqslant 5$, while the opposite happens for $n \geqslant 6$.

One way to remove this ambiguity is to base our choice on the examination of what happens for sufficiently large values of the size of the input ($n \to +\infty$). Using this criterion in the above example, it is clear that algorithm A is "better" than algorithm B. In fact, A will be "better"

---

* A <u>graph</u> is a collection of <u>points</u>, called <u>nodes</u>, which represent entities, and <u>lines</u>, called <u>links</u>, joining certain pairs of the nodes and represent relationships between them. An elementary knowledge of concepts such as this is assumed of the reader.

than B even if one uses A on the slowest computer and B on the fastest, for sufficiently large values of n.

In this respect, we may note that _any_ algorithm whose running time is a polynomial function of the size of the input is "better" than _any_ algorithm whose running time is an _exponential_ function of the size of the input, irrespective of what computers these algorithms are run on.

Computer scientists have more or less agreed that algorithms that consume time which is exponential relative to the size of the input are not practically useful [LEWI 78]. In that spirit, these algorithms have been characterized as "inefficient". By contrast, an "_efficient_" algorith is said to be one whose running time grows as a _polynomial_ function of the size of the input.

It would perhaps be useful to make several remarks concerning this concept:

1) There may be algorithms whose running times are _non-deterministic_. That is, one may not know beforehand exactly after how many steps the algorithm will terminate, this depending in an unknown fashion upon the values of the particular inputs. If this is the case, it is important to distinguish between _worst-case_ performance and _average_ performance of the algorithm, since these two may be very different. For a given size of the problem's input, _an upper bound_ for the algorithm's running time may be obtainable. The _worst-case_ performance occurs for those instances of the problem that make the algorithm's running time reach this upper bound (for many non-deterministic algorithms the generation of such worst-case instances, is an art in itself). On the other hand, the algorithm will not be used only in worst-case instances, but also in other, more easy cases. The aspect of the algorithm's performance that emerges from this consideration

is its average performance.

There seems to be no definite answer to the question of which of the two aspects is more important. Although it would certainly be desirable to have a strong "guarantee" for the performance of an algorithm (and such a guarantee can come only from a good worst-case performance), there may be some "controversial" algorithms which have bad worst-case performances and exceptional average performances. In fact, the best known example of such a controversial non-deterministic algorithm is none other than the famous Simplex method introduced by George Dantzig in Linear Programming [DANT 51, DANT 53]. The controversy lies in the fact that while on the average the running time of this algorithm is proportional to a low power polynomial function of the size of the input, carefully worked out instances of Linear Programs, show that Simplex may require an exponential amount of time [ZADE 73]. Thus, a yet unsolved enigma to mathematical programmers is the question of why an algorithm as "bad" as Simplex (in the sense that there is no polynomial performance guarantee) turns out to work so exceptionally well [KLEE 70].

2) Examining the performance of an algorithm asymptotically as we did for algorithms A and B earlier has the advantage of making the speed of the algorithm its own intrinsic property, not depending on the type of computer being used, or on possible technological breakthroughs in computer design and speed. However, this asymptotic behavior should be studied with caution. A hypothetical and extreme example where the polynomial/exponential discrimination may be illusory is if we compare a deterministic algorithm whose running time goes, say, as n^{80} with a deterministic algorithm for the same problem whose running time goes as 1.001^n. It would clearly be a rash act to adopt the first algorithm because it is

"efficient" (polynomial) and reject the second because it is "inefficient" (exponential). In fact, it is true that $\lim_{n\to\infty} \frac{n^{80}}{1.001^n} = 0$, but the values of n for which $1.001^n > n^{80}$ are so large, that they most probably lie outside the range of practical interest.

This example was hypothetical and extreme. In fact, all known polynomial algorithms for graph-theoretic problems grow much less rapidly than n^{80} (most known algorithms of that category grow as n, nlogn, n^2, n^3 and at most n^5). In addition, all known exponential algorithms grow much more rapidly than 1.001^n (for example, there are algorithms growing like 2^n). Therefore, in subsequent sections we shall adhere to the definition of computational "efficiency" given earlier. We shall excersise caution however, and think twice before "accepting" an algorithm just because it is polynomial or "rejecting" it just because it is exponential.

1.1 The Concept of NP-Completeness in Combinatorial Optimization Problems.

Among problems where the issue of computational efficiency is extremely important are those belonging to the general category of combinatorial optimization problems. Since the Aircraft Sequencing Problem which we will be studying throughout this report belong to that general category, it is important to examine the issue of computational efficiency from a slightly more specialized point of view.

The two examples which follow constitute well known combinatorial optimization problems and will provide motivation for subsequent parts of the thesis:

Let G be the graph of Fig. 1.1. We assume that if we use a particular link of G, we pay the corresponding cost ($1 for link AB, $2 for link BC, etc.). Missing links have infinite cost (e.g. AC).

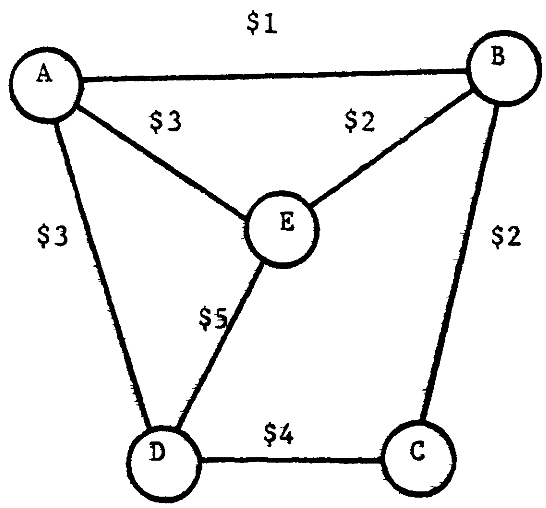


Fig. 1.1.

(MSTP) (TSP)

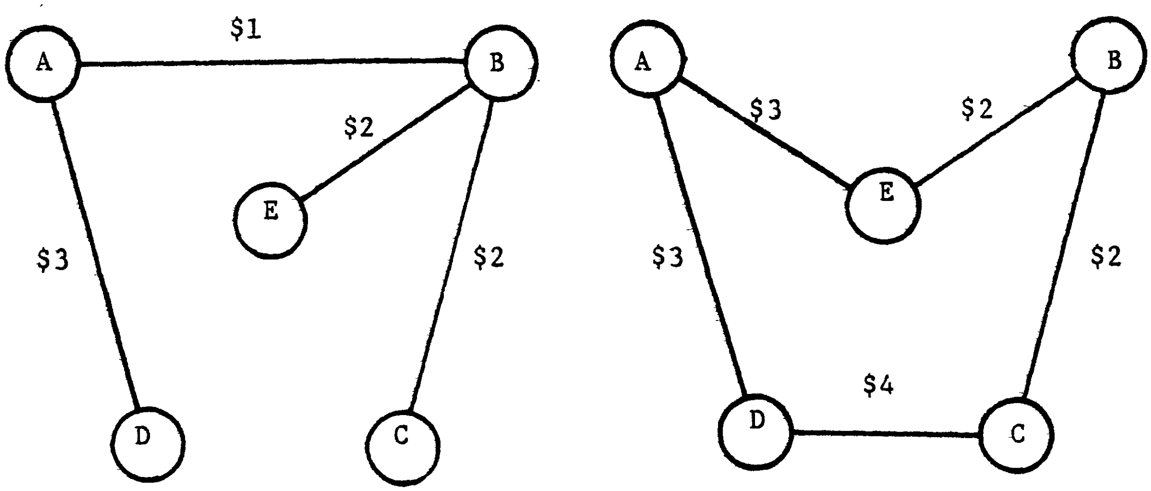


Fig. 1.2 : Optimal solutions to the MSTP and TSP of the graph of figure 1.1.

We consider the following problems concerning G:

Problem 1 (Minimum Spanning Tree Problem or MSTP): Use as many links of G as necessary, to connect* all nodes of G at minimum cost.

Problem 2 (Travelling Salesman problem or TSP): Find a tour in G of minimum cost. (A tour is a sequence of visits to all nodes of the graph exactly once, which returns to the initial node.)

Before proceeding with more details, we exhibit the optimal solutions to these two problems in Fig. 1.2. We may at first glance note that the two problems above seem very similar in structure and we might also expect that they are very similar with respect to solution precedures and computational effort required, as well.

Surprisingly enough however, it turns out that the MSTP is one of the easiest problems known in the field of combinatorial optimization. By contrast, the TSP is one of the toughest ones.

Many "efficient" algorithms exist for the MSTP. The fastest for arbitrary link costs and a general graph structure is the one by Prim [PRIM 57], in which the running time goes as n^2 (n: number of nodes of the graph). For graphs where the nodes are points in a Euclidean plane and link costs are proportional to the Euclidean distance between the two nodes, one can do even better than that. The algorithm of Shamos and Hoey [SHAM 75] exhibits a computational effort that goes as nlogn.

By contrast, no "efficient" algorithm has been developed for the TSP, despite the fact that the problem has been a very popular target for ambitious mathematicians and operations researchers for several decades.

---

* Connectivity means to be able to reach every node of the graph from every other node of the graph, using links that have been chosen.

All algorithms that exist today for the TSP are "inefficient." These failures led to the subsequent classification of the TSP into a relatively wide class of similarly notorious problems, the so called NP-complete problems*. This class of problems has a unique property: If an "effient" algorithm exists for just one of them, then "efficient" algorithms exist for all of them. A consequence of this property is that if it is proven that just one of these NP-complete problems cannot have an "efficient" algorithm, then none of these problems can [KARP 75].

As a result of the above ideas, the effort of researchers on the subject, has been channeled toward two opposite directions: Either toward finding an "efficient" algorithm for a specific NP-complete problem, or, toward proving that this is impossible. The efforts in this latter direction are a natural consequence of the repeated failures in the former.

Unfortunately, however, these new efforts have so far had the same fate as the previous ones. In other words, no one to date has managed to prove the impossibility of devising polynomial algorithms for NP-complete problems and this leaves the status of this case open. Incidentally, it has also been realized that there are some other problems which are one step closer to "darkness" than NP-complete problems are: These are problems for which nobody knows if they are NP-complete or not. (The problem of Graph Isomorphism and the general Linear Programming problem belong to this category.)

Returning to NP-complete problems, current opinion is divided. Some researchers strongly suspect that no "efficient" algorithm can be

*NP for Non-deterministic Polynomial

constructed for them. Others speculate that these problems will eventually yield to polynomial solutions. Finally there are those who believe that the resolution of this question will require the development of entirely new mathematical methods. [KARP 75, LEWI 78].

Some other remarks are worthwhile:

1) The solution of the TSP of the previous example may have seemed trivial because there were only two feasible tours in the graph of Figure 1.1. It is important though to bear in mind that the number of feasible tours in a <u>complete</u> graph with symmetric distances grows explosively with the size of the problem, being in fact equal to $\frac{1}{2}(n-1)!$ As for graphs with missing links like the one we examined, it is conceivable that <u>not even one</u> feasible tour exists. Moreover, the problem of finding whether a given graph has a TSP tour is, in itself, a NP-complete problem.

2) Referring once again to the possibility that the number of feasible solutions to a combinatorial optimization problem may be explosively large, this fact <u>should not</u> be automatically associated with the "toughness" of that problem. As a matter of fact the number of feasible solutions to the MSTP for a complete graph of n nodes is <u>even larger</u> than to the equivalent TSP! (This number is equal to n^{n-2}; a bibliography of over 25 papers proving this result is given in [MOON 67].)

<u>1.2 Remedies for NP-Completeness</u>

There are several things one can do if faced with a NP-complete problem, besides quitting:

1) Accept the high computational cost of an exponential algorithm.

2) Compromise on optimality.

3) If the problem at hand has a special structure, try to take advantage of it.

The first alternative may be viable only under limited circumstances. Excessive storage or CPU time requirements are the most common characteristics of an exponential algorithm even when the size of the problem examined is of moderate magnitude. Thus, large scale problems or problems requiring real-time solutions may make the use of such algorithms not attractive and even prohibitive.

The second alternative is very interesting. The basic philosophy behind it is the following: "If it is so expensive to obtain the exact optimal solution of a problem, perhaps it is not so expensive to obtain a "reasonably good"* solution. In any event, the exact optimal solution may not be so terribly important because the mathematical problem itself is an abstraction and therefore an approximation of the real-world problem."

Following this line of logic, various techniques have been developed in order to obtain "reasonably good" solutions to a given optimization problem, requiring only a fraction of the computational effort required to obtain the exact optimal solution.

A major reason for adopting this approach has been the anticipation and subsequent verification of the fact that many algorithms produce a "reasonably good" solution quite rapidly but spend a disproportionate amount of computational effort for closing the remaining narrow gap between the cost of this "reasonably good" solution and the cost of the exact optimal solution. This is particularly common with algorithms

*We shall give an explicit interpretation of this term later.

which produce one feasible solution per iteration and subsequently move to another "improved" feasible solution (according to some rigorous or heuristic criteria), until no further improvement is possible. (It should be noted that not all optimization algorithms are of this hill-climbing or hill-descending nature.) In these cases, it may often be relatively easy to improve upon a "bad" initial feasible solution (hence the rapid generation of "reasonably good" solutions at the beginning), but hard to improve upon a "reasonably good" solution (hence the substantially greater amount of computation in order to make the last small improvements needed to reach the exact optimum).

Thus, we can immediately see a way to construct an algorithm that only produces a "reasonably good" solution: take an algorithm of the above form and operate it until some termination criteria are met. These termination criteria may be one or more of the following:

First, there may be a "resource" constraint, which is translated to limits in the number of iterations, or in CPU time, cost, etc. If this criterion is followed, the algorithm terminates upon exhaustion of the available resource.

Alternatively, a criterion may concern the quality of the best solution found so far. The algorithm then terminates if this solution is "reasonably good." Different people may interpret the term "reasonably good" in different ways. One of the interpretations usually adopted is the following: A "reasonably good" solution is defined as one whose cost provably cannot exceed the (still unknown) minimum total cost of the exact optimal solution, by more than a prescribed percentage of the latter.

For example one might consider as a "reasonably good" solution one whose cost is within 10% of the optimal cost.

Branch-and-bound algorithms are one class of non-deterministic exponential algorithms which are tailored to incorporate such tolerances from optimality, besides their ability to produce also exact optimal solutions, given sufficient time. It is unfortunate that it is not possible to drop the running times of these algorithms to polynomial by sufficiently relaxing the optimality requirements.

By contrast, it may be possible to accomplish this exponential-to-polynomial reduction in the so-called heuristic algorithms. These are procedures using arbitrary rules in order to obtain a "reasonably good" solution. These rules may be anything from "common sense,""intuitive" or even "random" courses of action, to more elaborate procedures, sometimes employing separate optimization algorithms (drawn from other problems) as sub-routines. The very name of a "heuristic" algorithm suggests that the algorithm is not a rigorous procedure for reaching the exact optimum, but rather a set of empirical rules which hopefully will yield an acceptable solution and more hopefully the optimal solution itself. A solution produced by a heuristic algorithm will be in general sub-optimal.

A heuristic algorithm may or may not possess a performance guarantee in the sense that the cost of its solution is guaranteed to be within prescribed limits with respect to the exact optimum.

Concerning the TSP, Christofides [CHRI 76] has recently developed a heuristic polynomial algorithm which guarantees solutions of cost no more than 150% of that of the optimal solution.

In fact, this upper bound of 150% can be asymptotically reached in carefully worked-out specific problem instances [FRED 77]. It is interesting to note that the specific problems for which this happens have a Euclidean cost structure, structure which could conceivably be exploited to get better results. The fact that not even a heuristic algorithm can guarantee TSP solutions with cost deviations from optimality of less than 50% without requiring exponential running times, is a further indication of the inherent toughness of this problem. It should be pointed out however that 50% is a worst-case performance. The average performance of this algorithm has been astonishingly good (solutions average cost deviations of 5-10% from optimality) and this is a further indication that worst-case performance should not be confused with average performance, since these two may differ substantially.

As a third remedy to NP-completeness, we come to what will be seen to constitute a central concept in this investigation This concept can be broadly named as "exploitation of special structure."

We have already briefly mentioned an example where the special structure of a problem could be used to solve the problem more efficiently: In the MSTP, the very fact that the graph may be Euclidean, can be used to drop the order of computational effort from n^2 to nlogn. By contrast, no similar improvement can be realized for the TSP. In fact, the Euclidean Travelling Salesman Problem is itself NP-complete [PAPA 77].

Similar or more spectacular refinements of general-purpose algorithms to fit specialized problems have constituted a major topic of research in the area of mathematical programming. These "streamlined" versions of the general-purpose algorithms can perform the same job as the latter, but much more efficiently. Examples of this can be found in the plethora of specialized algorithms in use for many network problems in transportation, such as maximum flow, shortest path, transshipment and transportation problems [DIJK 59, DINI 70, EDMO 72, KARZ 74]. These problems are in essence Linear Programs and can be in principle solved by the Simplex method which we have already mentioned earlier. But it turns out that these specialized algorithms are so successful, that using Simplex instead of them can be characterized as a waste of effort despite Simplex' widely recognized success. What essentially has been achieved through the exploitation of the special structure of these problems (and of many others) is that the algorithms that have been developed for the problems are polynomial.

CHAPTER 2

THE DYNAMIC PROGRAMMING SOLUTION TO THE TRAVELLING SALESMAN PROBLEM

2.0 Introduction

It will be seen that the Aircraft Sequencing Problem which will be examined in subsequent chapters of this report is closely related to the classical TSP. Differences do exist with respect to objective functions, special constraints and other more subtle aspects. Nevertheless, one should be able to detect a TSP "flavor" in the ASP.

It seems therefore appropriate at this point to examine how one can solve the TSP. In particular, the classical Dynamic Programming Approach to the TSP will provide the necessary background and motivation for the more specialized and sophisticated solution algorithms which will be developed later.

In the general formulation of the TSP we shall assume that we are dealing with a _directed_, _complete_, and _weighted_ graph G of N nodes.

Directed means that the lines joining pairs of nodes are themselves _directed_. These will be called _arcs_ (e.g. arc (i,j) is a line _from_ node i _to_ node j) in distinction to undirected lines which are called _links_.

Complete means that for every ordered pair of distinct nodes (i,j) there exists an arc from i to j. (It is also conceivable that there exist _loops_, namely arcs going from a node to itself. These arcs will not be of interest here and will be neglected. However, we will encounter them in a subsequent part.)

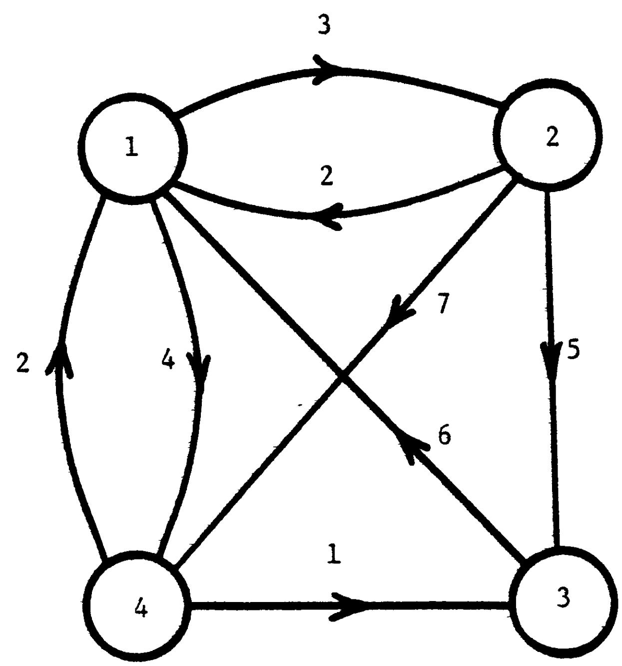
Weighted means that every arc (i,j) is assigned a number, called interchangeably distance, separation or cost, d_{ij}. It is not necessary that the graph is symmetric i.e. that $d_{ij} = d_{ji}$. Also, the cost to go from a node i to another node j of the graph, will depend on what other intermediate nodes will be visited. d_{ij} is the cost of going from i to j directly, so this cost is sometimes called one-step cost between i and j and the corresponding NxN matrix $[d_{ij}]$ one-step cost matrix. Since in a graph with general cost structure the going from i to j directly may not necessarily be the cheapest way, d_{ij} is to be distinguished from d'_{ij}, which is the minimum possible cost to go from i to j, using intermediate nodes if necessary*. A case where $[d'_{ij}]$ coincides with $[d_{ij}]$ is when the latter matrix satisfies the so-called triangle inequality, i.e. $d_{ij} \leq d_{ik} + d_{kj}$ for all (i, j, k). Among other cases, this is true when the nodes of the graph are points in a Euclidean plane and d_{ij} the Euclidean distance between i and j.

We will also assume that if a graph is not complete, i.e. certain arcs are missing, then the cost of these arcs is infinite. Thus, in absence of loops, we can put $d_{ii} = \infty$ for every node i.

An example of a 4-node graph with its corresponding cost matrix is given in Fig. 2.1

The Travelling Salesman Problem then calls for finding a sequence of nodes $\{L_1, L_2, \ldots L_N, L_{N+1}\}$ which forms a tour, so that the total cost associated with it ($\sum_{j=1}^{N} d_{L_j, L_{j+1}}$) is minimized. Since we are dealing with a tour, $L_{N+1} \equiv L_1$.

*It is possible to construct $[d'_{ij}]$ from $[d_{ij}]$ in time proportional to N^3 [FLOY 62, MURC 65].



$$[d_{ij}] = \begin{bmatrix} \infty & 3 & \infty & 4 \\ 2 & \infty & 5 & 7 \\ 6 & \infty & \infty & \infty \\ 2 & \infty & 1 & \infty \end{bmatrix}$$

Fig. 2.1 : A 4-node directed graph and its corresponding cost matrix $[d_{ij}]$.

Without loss of generality we can specify an arbitrary starting node therefore put $L_1 = L_{N+1}$ = node 1.

2.1 The Dynamic Programming Algorithm.

We now present the Classical Dynamic Programming Approach for solving problems of this kind. This method is due to Held and Karp [HELD 62].

A stage in the TSP involves the visit of a particular node. It can be seen that the TSP problem above has N+1 stages: At stage 1 we are at the initial node 1. At stage 2 we move to another node and so forth until we return to node 1 at stage N+1.

The information on which we shall base our decision on what node to visit in the next stage n+1, given that we currently are at stage n, is described by the following state variables:

L: the node we are currently visiting.

$k_1, k_2, \ldots k_N$: a vector describing what nodes have been visited so far. By definition:

$$k_j = \begin{cases} 0 & \text{if node } j \text{ has been visited} \\ 1 & \text{otherwise.} \end{cases}$$

By convention, at the beginning of our tour we are at node 1 but since we have to visit this node again at the end, we put $k_1=1$.

We also define the optimal value function $V(L, k_1, \ldots, k_N)$ as the minimum achievable cost to return to node 1 passing through all unvisited nodes, given our current state is $(L, k_1, \ldots, k_N)$

It is not difficult to see that the difinition of V above implies the following optimality recursion, also known as Bellman's principle of optimality [BELL 60]:

$$V(L, k_1, \ldots, k_N) = \underset{x \in X}{\text{Min}}[d_{L,x} + V(x, k_1', \ldots, k_N')] \tag{2.1}$$

where

$$X = \begin{cases} \{1\} \text{ if } k_2 = k_3 = \ldots = k_N = 0 \text{ and } k_1 = 1 \\ \{i: i \neq 1 \text{ and } k_i = 1\} \quad \text{otherwise} \end{cases} \tag{2.2}$$

and

$$k_j' = \begin{cases} k_j - 1 \quad \text{if } j = x \\ k_j \quad \text{otherwise} \end{cases} \tag{2.3}$$

In (2.2), the set X of potential "next" nodes is described mathematically and in (2.3) the k-vector of a "next" node x of X is derived from $(L, k_1, \ldots, k_N)$. Two facts are clear:

1) At the end of the tour, V is zero.

   So $V(1,0,\ldots,0)=0$. (2.4)

2) The total cost of the optimal solution is given by the value of $V(1,1,\ldots,1)$.

To compute this value we use the technique called <u>backward recursion.</u> According to it, we evaluate (2.1) using (2.2) and (2.3) for all combinations of $(L, k_1, \ldots, k_N)$ in the following manner: We start from $L=1$, $k_1=\ldots=k_N=0$ where (2.4) applies and then we move to <u>lexicographically</u> greater* values of the k-vector, each time examining all values of L

---

*A vector $(a_1,\ldots,a_N)$ is said to be <u>lexicographically greater</u> than another vector $(b_1,\ldots,b_N)$ if either:

(a) $a_1 > b_1$

or (b) there exists an index n between 1 and N such that $a_j = b_j$ for all $1 \leq j < n$ and $a_n > b_n$.

For example, (1,0,0) is lexicographically greater than (0,1,1) because of (a) and (1,0,1) is lexicographically greater than (1,0,0) because of (b).

from 1 to N. Each time we apply (2.1) we store two pieces of information: First the value of V. Second, what has been our best choice as to what node to visit next. We store this last item in an array NEXT $(L, k_1, \ldots k_N)$. This array will eventually serve to identify the optimal tour. This identification takes place after the backward recursion is completed, i.e. after state (1,1,...,1) has been reached.

We know that we are initially at node 1 and our state is (1,1,...,1). The best next node is given by the value of NEXT (1,1,...,1). Supposing for the sake of arguement that this is equal to 3, this means that our state becomes now (3,1,1,0,1,...,1). The best next node is given by the corresponding value of NEXT and so forth until, after N steps, we arrive back to node 1 and our state becomes (1,0,0,...,0).

2.2. Comparison with Other Approaches

We can see that the computational effort associated with this approach grows quite rapidly as N increases, but still far more slowly than the factorial function associated with a complete enumeration scheme.

In fact, (2.1) will be used a number of times of the order of $N.2^N$. The reason is that each of the k's can take two values (0 or 1) and L can take N values (1 to N). Equivalent amounts of memory should be reserved for each of the arrays V and NEXT. This quite rapid growth makes this approach not practical for the solution of TSP's in graphs of more than about 15 nodes. By contrast, other exact approaches have been shown to be albe to handle TSP's of about 60-65 nodes [HELD 70, HELD 71], while several heuristic algorithms handle TSP's of up to 100-200 nodes [LIN 65 LIN 73, KROL 71].

It becomes clear therefore that some explanation should be offered here concerning the purpose of presenting the Dynamic Programming approach.

One perhaps evasive answer to that question is to state that the reasons for which the Dynamic Programming approach has been introduced will eventually become clear in subsequent chapters of this report. In fact it will turn out that this approach, in the form of more sophisticated algorithms, will prove itself useful in tackling the Aircraft Sequencing Problem.

Nevertheless, we can also state *a priori* some features of the Dynamic Programming approach that make it particularly advantageous by comparison to other approaches.

The first feature concerns the versatility of this approach with respect to *the form* of objective functions that can be handled. The only requirement concerning the form of objective functions suitable for Dynamic Programming manipulation is that of *separability:* As this technique is used in *multi-stage* problems, one should be able to *separate* the cost corresponding to a stage into the cost corresponding to a subsequent stage and the "transition" cost to go from the former stage to the latter. This separation is not restricted to be additive.

Let us state at the outset that this *separability* requirement may become very restrictive and even prohibitive if certain forms of objective functions are to be considered. For example, if a *quadratic* objective function is used, then the manipulations required to bring this function to a separable form, will eventually increase the computational effort of the procedure much faster than of a linear objective function.

Despite this restriction, there still remain several forms of objective functions that can be separated, in addition to the one examined in the "classical" version of the TSP presented above (minimize total cost of tour). We shall have several opportunities to examine these alternative forms of objective functions later on. For the moment, for motivation purposes we will present one of them:

Consider the following variation of the TSP: A tour is to be executed by a bus which starts from node 1 carrying a number of passengers. Each of these passengers wishes to be delivered to a specific node of the graph. After completion of all deliveries, the bus has to return to node 1. During the trip, each passenger will "suffer" an amount of "ride time" into the bus till his delivery. What should be the sequence of bus stops which minimizes the sum of ride times (or, equivalently, the average ride time)?

It should be noticed that due to the different form of the objective function, this problem is not the same as the classical TSP seen earlier and in general the optimal solutions to these two problems will not be identical.

In fact, the problem we have just presented belongs to the general category of "mean finishing time" minimization scheduling problems, which unfortunately, are also NP-complete. A more general version of the vehicle routing version of this problem has been studied in detail in [PSAR 78]. A generalized version of this objective function will also be studied in Part II.

The important remark, however, at this stage of our presentation is that Dynamic Programming can handle this new form of objective function with the same ease as it can handle the previous one, being in addition able to consider any linear combination of these two objective functions.

By contrast, other approaches more successful than Dynamic Programming in solving "classical" TSP's, fail completely if alternative forms of objective functions like the one above are considered. An example is the node penalty/subgradient optimization/branch-and-bound approach by the same authors who introduced the Dynamic Programming approach for the TSP. Held and Karp in [HELD 70, HELD 71] have presented an algorithm, able to handle TSP's of 60-65 nodes. It is perhaps not fully appreciated that a critical factor in the success of that approach is the form of the objective function itself. Specifically, Held and Karp ingenuously observed that the identity of the optimal solution to the TSP will not change if an arbitrary set of "penalties" is imposed on all nodes of the graph (so that in addition to the cost incurred in using a particular link of the graph, we also have to pay a penalty for each node we visit). Based on that observation the authors subsequently developed a procedure to identify the combination of node penalties which will enable one to obtain the TSP solution directly from the equivalent MSTP solution.

With respect to this approach, it turns out that the above fundamental observation of the authors simply does not hold if one is examining an objective different from the classical one. So one is immediately forced to reject this approach if these alternative objectives are examined.

The same observation holds for other approaches including several heuristic algorithms. In particular, the recent ingenuous algorithm of Christofides [CHRI 76] cannot be applied if other than classical objectives are examined.

The above argument does not mean that there do not exist specialized algorithms for tackling these alternative objectives; in fact such algorithms do exist [IGNA 65, VAND 75]. Rather, the purpose of the discussion was to emphasize the flexibility of the Dynamic Programming approach regarding certain alternative forms of objectives.

Other features that make the DP approach attractive are the relative ease in computer coding, the capability of examination of specialized constraints and an effective "re-optimization" ability.

It is however premature to get into details concerning these positive and some inevitable negative aspects of the technique at this point. We shall have several opportunities to do this in parallel with the examination of the specific problems which will be presented subsequently.

PART II

THE AIRCRAFT SEQUENCING PROBLEM (ASP)

CHAPTER 3

AIRCRAFT SEQUENCING: PROBLEM DESCRIPTION

3.0 Introduction

It was mentioned in Part I that certain combinatorial optimization problems exhibit structures, which, if adequately exploited, may lead to the development of "specialized" algorithms solving these problems much more efficiently than a general-purpose algorithm could do.

Following this philosophy, the purpose of Part II is to introduce and discuss a version of the Aircraft Sequencing Problem (ASP), as well as to develop a "specialized" algorithm for it. The above problem will be seen to be directly related to the Travelling Salesman Problem (TSP) already introduced in Part I, being in fact itself NP-complete. Nevertheless, the structure of this problem is such, that, through specially tailored algorithms, drastic computational savings can be realized over the effort needed to solve the problem with the classical Dynamic Programming algorithm (also described in Part I). Moreover, the same structure will be shown to allow the inclusion of a special kind of priority constraints, the Constrained Position Shifting rules. We shall examine these and other issues in detail throughout this Part of the report.

Before presenting the mathematical formulation of the ASP let us take a brief look at the real-world problem:

During peak periods of traffic, the control of arriving and departing aircraft in an airport's near terminal area becomes a very complex task.

The modern air traffic controller must, among other things, see to it that every aircraft in that high density area, either waiting to land or preparing to take off, maintains the required degree of safety. The same person also has to decide what aircraft should use a particular runway, at what time this should be done and what manoeuvres should be executed to achieve this. The viable accomplishment of such a task becomes more difficult in view of the fact that aircraft are continuously entering and leaving the system and that at peak periods, the demand for runway utilization may reach, or even exceed the capabilities of the system. It is at such periods that excessive delays are often observed, resulting in passenger discomfort, fuel waste and the disruption of the airlines' schedules. Under such "bottleneck" conditions, an increase in collision risk can logically be expected as well. The interaction between delay and safety goes also in the opposite direction: Because of safety considerations, the sequencing strategy used by almost all* major airports of the world today is the First-Come-First-Served (FCFS) discipline. For reasons which will become apparent below, this strategy is very likely to contribute to the occurrence of excessive delays.

The minimization of delay or some other measure of performance** related to passenger discomfort, without violation of safety constraints is certainly a desirable goal. This goal is, at least theoretically, achievable, for the following reasons:

* London's Heathrow Airport is an exception: A more sophisticated, computer assisted process is used there. [BONN 75.]

**Explicit definitions of these measures of performance will be given as soon as we formulate our version of the problem.

First, safety regulations state that any two coaltitudinal aircraft must maintain a minimum horizontal separation, which is a function of the types and relative positions of these two aircraft.

Second, the landing velocity of an aircraft will not be in general the same as the landing velocity of another aircraft.

A consequence of the variability of the above parameters (minimum horizontal separation and landing velocities) is that the minimum permissible time interval between two successive landings is a variable quantity. Thus, it may be possible, by rearrangement of the initial positions of the aircraft, to take advantage of the above variability and obtain a landing sequence that results in less delay than the FCFS discipline. In fact, an optimal sequence does exist; it is theoretically possible to find it by examining all sequences and select the most favorable one.

The above argument holds only as far as the potential for improvement over the FCFS discipline is concerned. How to find, and, equally importanly, how to implement an optimal - sequencing strategy is another story. The method suggested above for the determination of the optimal sequence is safe, but extremely inefficient, because the computational effort associated with it is a factorial function of the number of aircraft and it will not be possible to evaluate all the combinations in a short time interval (as the nature of this problem demands), even on the fastest computer*.

It should be pointed out that while the main factor that suggests the existence of an optimal landing sequence is the variability of the

*With only 10 aircraft, we would have to make 3,628,800 comparisons and with 15 aircraft 1,307,674,368,000 comparisons! Real-world problems, may involve the sequencing of 100 or more aircraft.

minimum permissible time interval between two successive landings, it is the same factor that makes the determination of this optimal sequence a nontrivial task. If that interval were constant, the Aircraft Sequencing Problem would be trivial to solve.

The real world problem involves many other considerations, especially as far as the implementation of sequencing strategies is concerned. In fact, the relevant literature on the subject is already considerable and growing. An excellent and thorough investigation of this problem has been recently published by Dear [DEAR 76]. We shall address several of these issues, as necessary, in subsequent chapters of this report. For the moment the above brief look into the real world problem is believed to be sufficient in order to proceed to the formulation of the version of the ASP we will examine.

3.1 Problem Formulation

Suppose that the air traffic controller is confronted with the following problem: A number of aircraft are waiting to land at a single runway airport. His task then is to find a landing sequence for these aircraft, so that a certain measure of performance is optimized, while all problem constraints are satisfied.

We now make the problem statement more specific:

1) All the aircraft to be considered are assumed to be "waiting to land." This would mean that then are arranged in a certain holding pattern, waiting for the instructions of the ground-based controller as to when each of them will start its landing manoeuvres. It is assumed that the pilots of all aircrafts are capable and willing to execute the

instructions of the controller given enough prior notice.

2) No intermediate arrivals of new aircraft will be assumed. In other words, the sequencing task of the air traffic controller will end as soon as all the planes of the "reservoir" of aircraft waiting to land, have landed. In view of this assumption, this version of the problem is termed "static." The equivalent real-world situation is obviously different. Airplanes are continuously arriving and are added to the system (the landing queue) in a random fashion. This version of the problem is termed "dynamic" and will concern us in Part III of the report.

3) Concerning the measures of performance, we shall concentrate on two of them. We call the first "Last Landing Time" (LLT). The corresponding objective is to find a landing sequence such that the aircraft that lands last, does this as quickly as possible. We call the second measure of performance "Total Passenger Delay" (TPD). The corresponding objective is to find a landing sequence such that the sum of the "waiting-to-land" times for all the passengers in the system as low as possible. It is not difficult to see that the second objective implies the minimization of the average per passenger delay. The two objectives will in general produce different optimal solutions, so that minimizing the one does not necessarily mean that the other is also minimized.

4) Concerning the problem constraints, we shall for the moment only require the satisfaction of the minimum interarrival time constraints. This means that the time interval between the landing of an aircraft i, followed by the landing of an aircraft j must not be less than a known time interval t_{ij}. We shall examine the derivation of the quantity in detail in Appendix B.

5) Without loss of generality we shall assume that we will start counting time (t=0) with the landing of the so-called zeroth airplane. Since this plane has already landed, it is not included in the "reservoir" of waiting-to-land airplanes, its only influence being how soon can the next (1st) airplane land. On the other hand, it is also conceivable that no such zeroth airplane would be specified (dummy zeroth airplane).

6) The composition of the set of airplanes which are waiting to land is, of course, assumed to be known. For each ordered pair (i,j) of aircraft, the minimum time interval, t_{ij} is also known and so is the number of passengers in each aircraft.

7) An assumption which we shall drop later, is that at any stage of the sequencing procedure, the air traffic controller is free to assign the next landing slot to any of the remaining aircraft. We shall refer to this version of the problem as the "unconstrained" case. This means that initially we shall not bother with priority considerations, i.e. we shall ignore the initial positions which these airplanes had when they arrived at the near terminal area.

8) It is also logical to assume that all aircraft wish to land as soon as possible. According to this, no unnecessary gaps in the utilization of the runway will be allowed. In other words, if aircraft i is followed immediately by aircraft j, the time interval between these two successive landings will have no reason to be strictly greater than t_{ij}, so it will be equal to t_{ij}.

3.2 Graph Representation of the ASP

It is not difficult to see that the problem described above can be depicted by means of the graph of Figure 3.1. The nodes of this graph

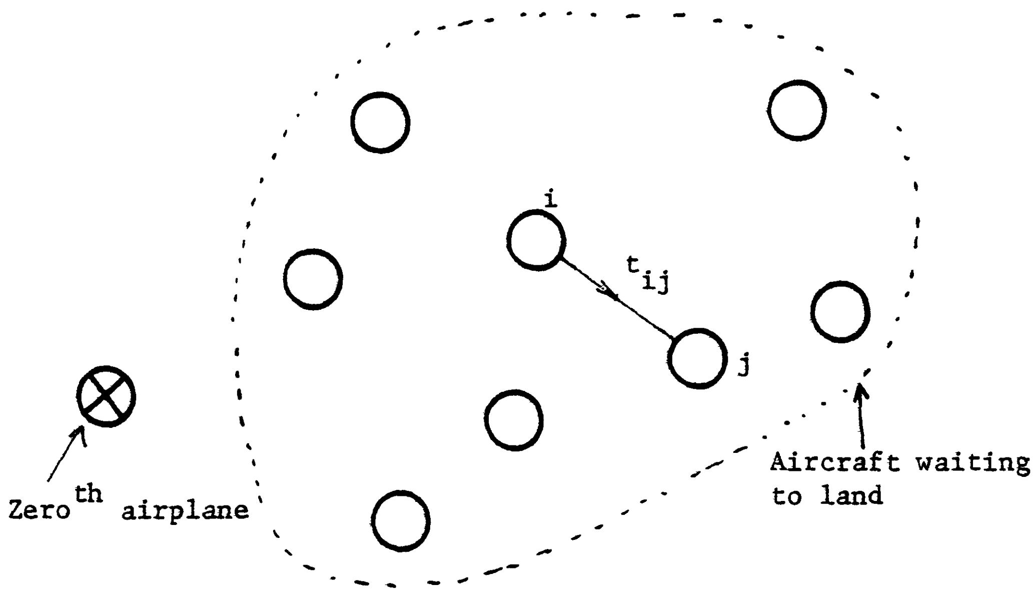


Fig. 3.1 : Graph representation of the ASP.

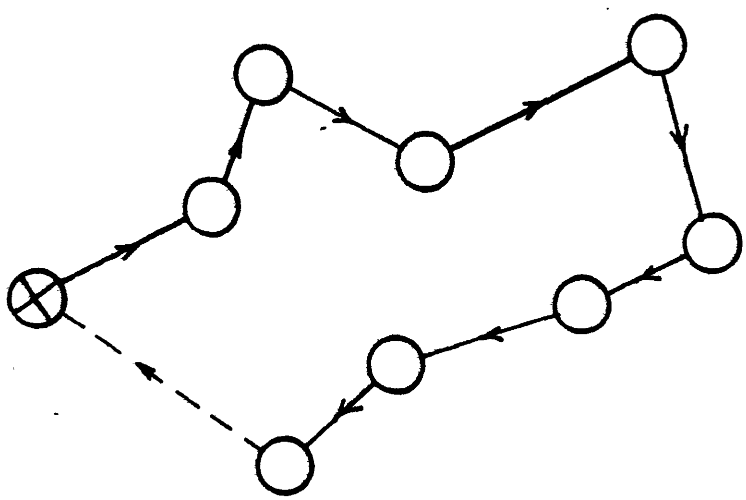


Fig. 3.2 : A sequence of aircraft landings. The dotted line is the zero-cost return arc.

represent the airplanes in the system. A special node (which may be a dummy one) represents the zeroth plane. The graph is complete and weighted. The cost of arc (i,j) is t_{ij}.

The ASP then simply calls for a sequence of visits to all nodes of the graph, exactly once to each, starting from the zeroth-plane node (node 0). Note that the problem, as it stands, does not require a return to the initial node, but one can easily reduce it to such a problem, by putting all costs $t_{i,0}$ equal to zero for all nodes of the graph. A feasible such sequence of visits, together with the zero-cost return arc is shown in Fig. 3.2.

One cannot avoid noticing the fact that if the objective of the problem is to minimize the Last Landing Time, then this problem is a classical case of the Travelling Salesmen Problem examined in Part I. On the other hand, if the objective is to minimize Total Passenger Delay, then the problem cannot be formulated as a classical TSP, but rather as a variation of it, because the penalty one pays by traversing an arc t_{ij}, depends on what nodes have been visited so far and thus is not known in advance.

The recognition that the problem we are trying to solve is likely to be at least as tough as the classical TSP (depending on the objective) may seem disappointing at first glance. If one were to adopt the classical Dynamic Programming Approach to the TSP (already presented in Part I), one could limit the number of airplanes that can be handled to about 15. In [PARD XX], Pardee states that a 14 aircraft problem was estimated to require approximately 180 seconds, a time which is substantially large in view of the need for real-time solutions (the minimum interarrival time between aircraft may be as low as 70 seconds). Thus, one is forced to abandon the classical D.P. Approach for solving the ASP.

Since we already have mentioned that we shall solve the ASP by exploitation of its special structure, let us now see what exactly is this structure and how we can take advantage of it:

A first observation is that we can classify the aircraft that are waiting to land into a relatively small number of distinct "categories." This classification should be such, that all airplanes belonging to the same category, although being distinct entities in themselves have the same or very similar characteristics, as far as minimum time intervals and number of passengers are concerned.

It is of course true that the above quantities are subject to random fluctuations even for two identical aircraft. The number of passengers, for example, in two B727's will in general be different. The same holds for their landing velocities, which not only depend on the individual loading conditions but are also subject to the pilots' discretion and may be influenced by weather conditions as well. Fluctuations in landing velocity translate into fluctuations in the minimum permissible time interval between successive landings. However, it is reasonable to assume that on the average, any two "similar" aircraft (like the two B727's in our example) will exhibit approximately equal values of the above parameters. It is in this spirit that our classification is being made.

Concerning the question of which aircraft are considered "similar" to one another, it is customary to divide the existing types of commercial aircraft into the following categories (although other classification schemes may exist):

First of all we can observe that the graph associated with the ASP can be drastically reduced in size. For example, the graph of Figure 3.3a for a 3-category problem can be reduced to the 3-node graph of Figure 3.3b. And instead of visiting exactly once all nodes of the large graph, we have the equivalent task of visiting each node of the condensed graph a specified number of times.

It should be noted that this does not mean that by working in the condensed graph we necessarily have to visit all the items of a category first, then move to another category and visit all its items etc. It is conceivable, for instance, that we may wish to visit two particular categories alternately and many times each.

Essentially the condensed graph contains all the information of the large one but in a more efficient way. Thus, instead of having to deal with a very large time separation matrix, the matrix in the condensed graph is much smaller. Note also that the diagonal elements of this latter matrix are not equal to infinity but to the minimum interarrival time between two successive landings of planes belonging to the same category. This means essentially that, by contrast to the large graph, the condensed graph does have "loops," as these were introduced in Chapter 2 (Compare Fig. 3.3a with 3.3b)

So let us redefine the ASP (unconstrained case) in a way compatible to the above observations:

A number of airplanes belonging to N distinct categories are waiting to land at a single runway airport. Let k_i^0 be the initial number of aircraft of category i, and P_i the (average) number of passengers (or number of seats) per aircraft of category i. Finally, let t_{ij} be the minimum

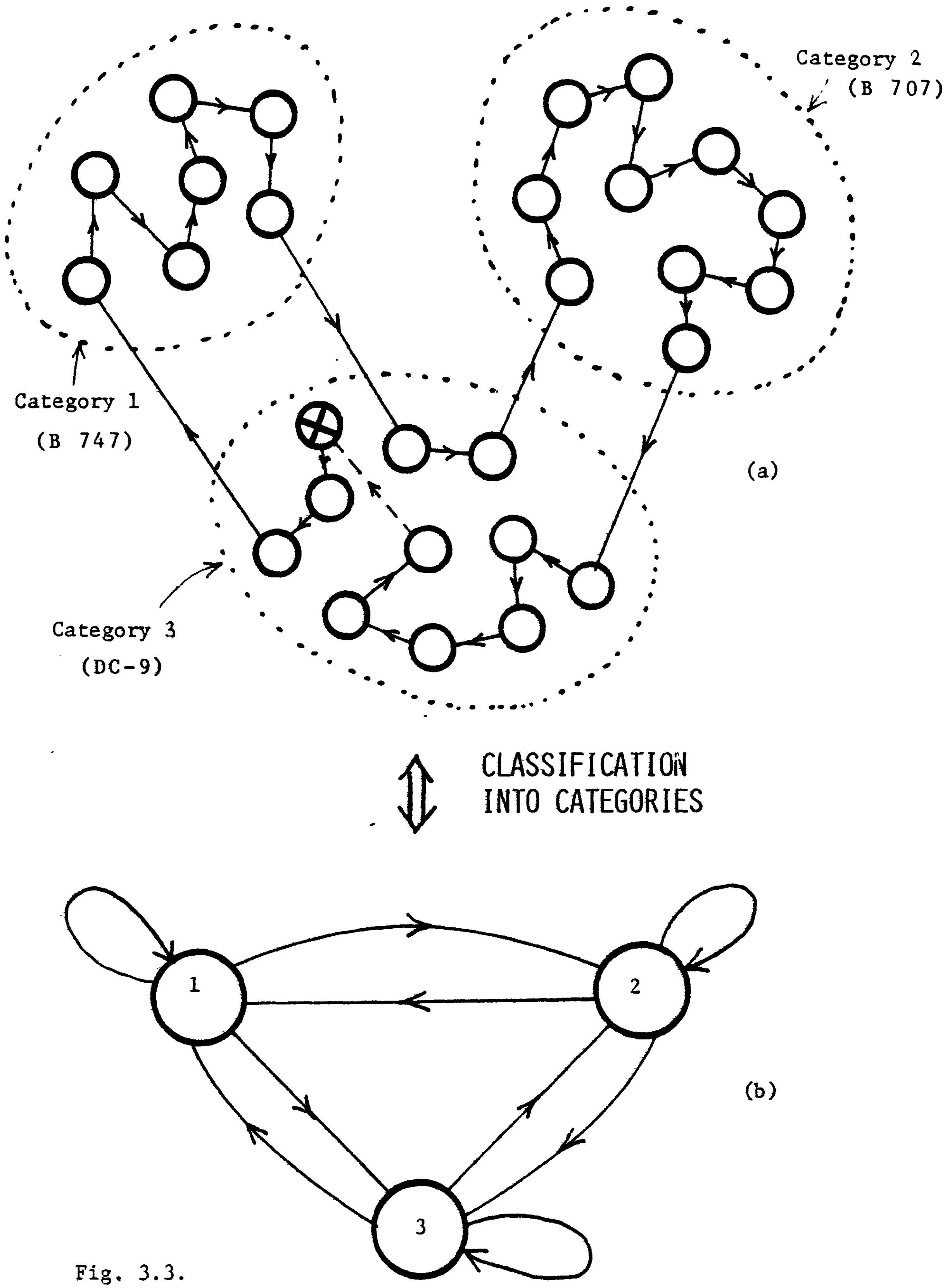

Category 2
(B 707)
Category 1
(B 747)
(a)
Category 3
(DC-9)
CLASSIFICATION
INTO CATEGORIES
1
2
3
(b)

Fig. 3.3.

permissible time interval which must elapse between the landing of a plane of category i, followed by a landing of a plane of category j and i_0 be the category of the zeroth landed airplane (t=0).

All the above quantities are the inputs to the ASP (unconstrained case).

Our goal is to find a landing sequence so that either one of the following two measures of performance is minimized:

1) Last Landing Time (We shall index this objective with Z=1).

2) Total Passenger Delay (We shall index this objective with Z=2)

In the next Chapter we shall describe a modified Dynamic Programming Approach that solves the above problem in a very small fraction of the time needed to accomplish this through the classical D.P. Approach.

CHAPTER 4

ASP - UNCONSTRAINED CASE: DYNAMIC PROGRAMMING SOLUTION

4.0 Introduction: Dynamic Programming Formulation

The modified Dynamic Programming Approach which we shall develop here has many similarities with the one we described in Chapter 2. In order to see how we can formulate the Aircraft Sequencing Problem as a dynamic program, the following considerations are important:

1) Stage-state description: A stage of the problem corresponds to the landing of a particular category of aircraft. At any particular stage the information we will need to make our decision for the next stage will consist of the following N+1 state variables:

1) L: The category which is landing at the current stage, namely the last of the categories landed so far. $L \in \{1,\ldots,N\}$.

2) $k_1,\ldots,k_N$: k_j is the number of airplanes belonging to category j which have not landed so far.

2) Decision variable: We call that x, the next category to land. It is clear that since all not landed aircraft are eligible to land at this next stage, x has to be chosen from the set $X = \{y: 1 \leq y \leq N, k_y > 0\}$.

3) Decision-state transition: We can see that if the state $(x, k_1', \ldots, k_N')$ immediately follows the state $(L, k_1,\ldots,k_N)$, then

$$k_j' = \begin{cases} k_j - 1 & \text{if } j=x \\ k_j & \text{otherwise} \end{cases}$$

for all $j=1,\ldots,N$.

4) Optimality recursions: We define $V_Z(L,k_1,\ldots,k_N)$ as the optimal value of all subsequent decisions that can be taken from the current state $(L,k_1,\ldots,k_N)$, till the end of the sequencing procedure ($k_1=\ldots=k_N=0$). V_Z measures time if Z=1 and total passenger waiting time if Z=2.

Since at any stage we have to choose the best of the elements of the corresponding set X, it is not difficult to see that the definition of V_Z above implies that:

$$V_Z(L,k_1,\ldots,k_N) = \min_{x\in X} [W_Z \cdot t_{L,x} + V_Z(x,k_1',\ldots,k_N')] \tag{4.1}$$

where $$X = \{y: 1\leq y\leq N,\ k_y > 0\} \tag{4.2}$$

$$W_Z = \begin{cases} 1 & \text{if } Z=1 \\ \sum_{j=1}^{N} P_j & \text{if } Z=2 \end{cases} \tag{4.3}$$

and $$k_j' = \begin{cases} k_j-1 & \text{if } j=x \\ k_j & \text{otherwise} \end{cases} \qquad (j=1,\ldots,N) \tag{4.4}$$

5) Boundary conditions: Obviously $V_Z(L,0,\ldots,0)=0$ for $Z=1,2$ (4.5) and for all $L=1,\ldots,N$, since if $k_1=\ldots=k_N=0$ we have no more aircraft to go.

6) Identification of the best "next": We define $NEXT_Z(L,k_1,\ldots,k_N)$ as the best, according to our objective Z, next category to land, given that our current state is $(L,k_1,\ldots,k_N)$.

By definition it is clear that

$$NEXT_Z(L,k_1,\ldots,k_N) = x, \text{ if}$$

$$V_Z(L,k_1,\ldots,k_N) = W_Z \cdot t_{L,x} + V_Z(x,k_1',\ldots,k_N') \tag{4.6}$$

where W_Z is given by (4.3) and k_j' by (4.4).

In case there are more than one x's satisfying (4.6) we break the ties arbitrarily.

4.1 Solution Algorithm

The optimal sequence of landings will obviously depend on the initial composistion of our aircraft "reservoir" $(k_1^0,\ldots,k_N^0)$ and on the zeroth landed plane i_0. Obviously, the state $(i_0, k_1^0,\ldots,k_N^0)$ will be the state at the beginning of the sequencing procedure. What is important to state at this point is that there is a way to avoid solving the problem again and again from scratch for different combinations of $(i_0, k_1^0,\ldots k_N^0)$. In fact, the algorithm we shall suggest is suitable for efficient repeated use, incorporating a tabulation scheme which enables it to "solve" the problem essentially only once and allowing use of the results repeatedly for any initial conditions we wish, with trivial additional computational effort. For this we need upper bounds $(k_1^{max},\ldots,k_N^{max})$ on the values of $(k_1^0,\ldots,k_N^0)$. The algorithm will consist of two parts:

a) The "Optimization" part, the "heart" of the procedure, where tables of V_Z and $NEXT_Z$ are prepared using (4.1) through (4.6). Backward recursion is used, starting from $k_1=\ldots=k_N=0$, where we take into account (4.5) and then moving to higher values of the k_j's lexicographically, up to k_j^{max}, for $j=1,\ldots,N$. For each combination $(k_1,\ldots,k_N)$, we apply (4.1), (4.2), (4.3), (4.4) and (4.6) for all values of L, from 1 to N.

By the end of this part, V_Z and $NEXT_Z$ have been tabulated for all possible combinations of the state variables up to their maximum values, so that this part, which is the most time consuming, does not have to be executed again for this problem.

b) The "Identification" part, which may be repeated as many times as we wish, for any given initial conditions $(i_0, k_1^0, \ldots, k_N^0)$. It will consist of $T = \sum_{j=1}^{N} k_j^0$ iterations, each corresponding to the determination of the best next category to visit. To do this we simply look sequentially at the already tabulated array $NEXT_Z$. Some caution is necessary if $i_0 = 0$, namely if there is no specified zeroth landed plane (dummy).

The "identification" part is formalized as follows:

<u>Step 0</u>: (Initialization)

$m=0$

$k_j = k_j^0 \quad (j=1,\ldots,N)$

$L_m = i_0 \quad$ (L_m: category of the m^{th} landed plane)

<u>Step 1:</u> (Termination check)

If $k_j=0$ for all $j=1,\ldots,N$, then set $T=m$ and STOP;

sequence $(L_0, L_1, \ldots, L_T)$ is optimal; END.

Otherwise continue

<u>Step 2:</u> (Move to best next)

If $L_m=0$, then determine L_{m+1} from:

$$V_Z(L_{m+1}, k_1, \ldots, k_N) = \underset{x=1,\ldots,N}{\text{Min}} [V_Z(x, k_1, \ldots, k_N)]$$

Otherwise $L_{m+1} = NEXT_Z(L_m, k_1, \ldots, k_N)$.

Step 3: (Update)

For all $j=1,\ldots,N$, set:

$$k_j = \begin{cases} k_j - 1 & \text{if } j = L_{m+1} \\ k_j & \text{otherwise} \end{cases}$$

Set $m=m+1$ and go to Step 1.

Some observations on the structure of the algorithm are important:

First, the reader may have noticed the absence of an explicit mention of the stage variable throughout the algorithm. This is due to the fact that this variable, n, is redundant so that one can calculate its value only from the state vector. Thus, if we start counting stages from the end of the sequencing procedure (backwards), then we can set $n \equiv \sum_{i=1}^{N} k_i$, so that at the end of the sequencing procedure, n is equal to zero.

A consequence of this is that the calculations of the "optimization" part of the algorithm are not performed in a stage-by-stage manner. The lexicographic manipulation of the vector $(k_1,\ldots,k_N)$ is the main reason for that. The following short example ($N=2$, $k_1^{max}=3$, $k_2^{max}=2$) will illustrate the order in which the recursion is performed:

Iteration	Current state (L,k_1,k_2) (in LHS of (4.1))	Stage n	"Next" states (x,k_1',k_2') (in RHS of (4.1))
1	(1,0,0)	0	-
2	(2,0,0)	0	-
3	(1,0,1)	1	(2,0,0)
4	(2,0,1)	1	(2,0,0)
5	(1,0,2)	2	(2,0,1)
6	(2,0,2)	2	(2,0,1)
7	(1,1,0)	1	(1,0,0)
8	(2,1,0)	1	(1,0,0)
9	(1,1,1)	2	(1,0,1) , (2,1,0)
10	(2,1,1)	2	(1,0,1) , (2,1,0)
11	(1,1,2)	3	(1,0,2) , (2,1,1)
12	(2,1,2)	3	(1,0,2) , (2,1,1)
13	(1,2,0)	2	(1,1,0)
14	(2,2,0)	2	(1,1,0)
15	(1,2,1)	3	(1,1,1) , (2,2,0)
16	(2,2,1)	3	(1,1,1) , (2,2,0)
17	(1,2,2)	4	(1,1,2) , (2,2,1)
18	(2,2,2)	4	(1,1,2) , (2,2,1)
19	(1,3,0)	3	(1,2,0)
20	(2,3,0)	3	(1,2,0)
21	(1,3,1)	4	(1,2,1) , (2,3,0)
22	(2,3,1)	4	(1,2,1) , (2,3,0)
23	(1,3,2)	5	(1,2,2) , (2,3,1)
24	(2,3,2)	5	(1,2,2) , (2,3,1)

The fact that the stage variable n does not increase monotonically with each iteration is of no serious consequence. What is of fundamental importance in all backward recursions and is in fact present here too, is the fact that for each state corresponding to a particular iteration (say state (1,2,2), iteration 17) all the "next" states (here (1,1,2) and (2,2,1)) have been evaluated at prior iterations (in this example, at iterations 11 and 16).

Obviously, one could also have achieved this by examining the states stage-by-stage: First evaluate all states of stage 0, then do the same for stage 1, and so on. This latter scheme has the advantage that we can use less storage space for the array V_Z than by using the lexicographic scheme. On the other hand, there are certain drawbacks associated with this approach: First, the number of states per stage is not constant and second, one also has to provide "coding" and "decoding" algorithms in order to identify them for a given stage n. We have had an opportunity to study similar and more complicated issues in detail in the work done in [PSAR 78] For the ASP we feel that the simplicity associated with the lexicographic scheme makes this scheme worthwhile to keep.

4.2 Computational Effort and Storage Requirements.

The simplest (but not necessarily the most efficient) way to have access to the arrays V_Z and $NEXT_Z$ is to keep them in main storage. In that fashion, we have to store $2N\prod_{j=1}^{N}(k_j^{max}+1) \triangleq 2C$ values and use (4.1) C times. If $k_j^{max}=k$ for every j, then $C=N(k+1)^N$. Note that C is a polynomial function of the number of airplanes k per category. It is an exponential function of the number of categories N, so our algorithm would become inadequate if N were large. However, we mentioned earlier that N is small in the ASP. Typical values of C are given in the following table:

Values of $C=N(k+1)^N$

N	k=5	k=10	k=20	k=50
2	72	242	882	5.2×10^3
3	648	4.0×10^3	2.8×10^4	4.0×10^5
4	5.2×10^3	5.9×10^4	7.8×10^5	2.7×10^7
5	3.9×10^4	8.1×10^5	2.0×10^7	1.7×10^9

It is interesting now to compare the above computational effort with the computational effort associated with applying the classical D.P. approach to the ASP. For k aircraft per category, the graph would have kN nodes, so the equivalent value to C would become $C'=kN.2^{kN}$. Defining as $r = \frac{C'}{C}$, we see that $r = k[\frac{2^k}{k+1}]^N$ which is always ≥ 1 (r=1 when k=1). Typical values of r are:

Values of r

N	k=5	k=10	k=20	k=50
2	142	8.6×10^4	5.0×10^{10}	2.4×10^{28}
3	758	8.1×10^6	2.5×10^{15}	5.4×10^{41}
4	4.0×10^3	7.5×10^8	1.2×10^{20}	1.2×10^{55}
5	2.1×10^4	7.0×10^{10}	6.2×10^{24}	2.6×10^{68}

We can therefore observe the drastic savings in computational effort and in storage requirements arising only from the fact that we somehow have managed to exploit the special structure of the problem.

It should also be noted that the computational effort discussed so far is associated only with the "optimization" part of the algorithm. By comparison, the "identification" part is the least time-consuming, requiring only $T=\sum_{j=1}^{N} k_j^0$ iterations.

4.3 Computer Runs-Discussion of the Results

Let us now see how the algorithm works by presenting a few examples.

We start by examining several cases for a mix of N=3 categories of aircraft. Category 1 consists of B707's with P_1=150 passengers, category 2 consists of B727's with P_2=120 passengers and category 3 consists of DC-9's with P_3=100 passengers. A typical time separation matrix for these three categories is the following (in seconds):

$$[t_{ij}] = \begin{bmatrix} 70 & 100 & 130 \\ 70 & 80 & 110 \\ 70 & 80 & 90 \end{bmatrix}$$

For the next 4 cases we vary the initial composition of the aircraft reservoir (k_1^0, k_2^0, k_3^0), the zeroth landed category, is as well as the problem's objective: Z=1 stands for Last Landing Time minimization and Z=2 for Total Passenger Delay minimization:

Case 1: $(i_0, k_1^0, k_2^0, k_3^0) = (2,5,5,5)$, Z=1

As can be seen from Figure 4.1, the optimal sequence starts with the landing of all planes of category 2, proceeds with all landings of category 3 and finally ends with all landings of category 1. The Last Landing Time incurred was 1,220 seconds (the minimum) while the Total Passenger Delay was 1,299,000 passenger-seconds (not necessarily the minimum). For a total passenger number of 1850 (=5X150 + 5X120 + 5X100), this corresponds to an average per passenger delay of 702 seconds. To find now whether we can do any better than that for the same aircraft mix, we examine the next case.

Case 2: $(i_0, k_1^0, k_2^0, k_3^0) = (2,5,5,5)$, Z=2.

It turns out that we can improve upon the Total Passenger Delay of case 1, by forming a new landing sequence. Figure 4.2 shows that this sequence is entirely different from that of Fig. 4.1: It starts with all landings of category 1, proceeds with all landings of category 2 and ends with all landings of category 3. The Total Passenger Delay incurred is 1,053,500 passenger-seconds (the minimum) which corresponds to an average per passenger delay of 569 seconds and an improvement of 19% over the corresponding measure of performance of case 1. Note however that the new sequence has a longer Last Landing Time (1240 seconds) than that of the old one, a deterioration of 1.6%.

So in general one should expect that the minimization of one of the two measures of performance would be accompanied by a deterioration in the other one. Also, in general, the optimal solutions for these two objectives would be different. There are of course cases where the two objectives yield the same optimal sequence, as illustrated by cases 3 and 4:

Case 3: $(i_0, k_1^0, k_2^0, k_3^0) = (1, 2, 4, 3)$ Z=1

Case 4: $(i_0, k_1^0, k_2^0, k_3^0) = (1, 2, 4, 3)$ Z=2

Both cases yield the same optimal sequence: (Fig. 4.3) It starts with all the landings of category 1, proceeds with all the landings of category 2 and ends with all the landings of category 3. Last Landing Time

1) Category 1

2) Category 2

3) Category 3

EXPLANATION OF SYMBOLS : These symbols will be used in all cases of the rest of Part II.

4) * Zeroth-landed category

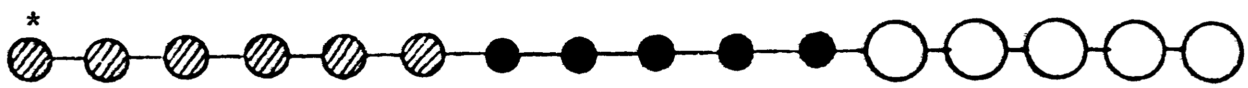


Fig. 4.1 : Case 1.

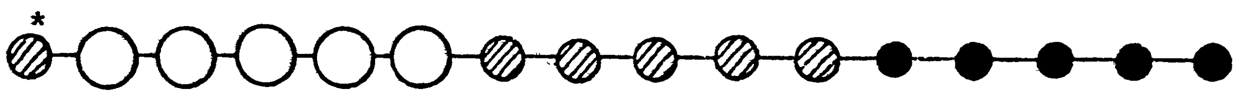


Fig. 4.2 : Case 2.

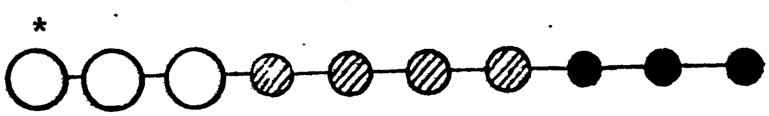


Fig. 4.3 : Cases 3 and 4.

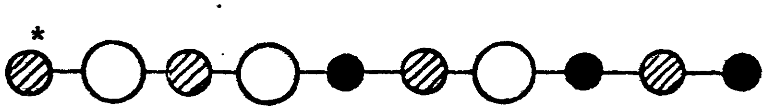


Fig. 4.4 : Case 5.

is 770 seconds (the minimum) and Total Passenger Delay is 408,300 passenger-seconds (also the minimum), corresponding to an average per passenger delay of 378 seconds.

An immediate and obvious observation is the following: In all cases above, the landings tend to be done in "bunches" by category. In other words all aircraft of the same category tend to cluster around each other and land as a group. This observation motivates the following question: <u>Could it be that this problem is so structured, that the optimal sequence of landings always involves category-clustering?</u> For if this is the case, then the problem is so trivial that it can be solved immediately by complete enumeration.

The next case provides a "no" answer to the question above. However, we shall see that our example will eventually lead to more complicated issues.

Suppose that we input a "random" time separation matrix into our problem, the following:

$$[t_{ij}] = \begin{bmatrix} 300 & 25 & 30 \\ 18 & 400 & 35 \\ 25 & 20 & 450 \end{bmatrix}$$

<u>Case 5:</u> $(i_0, k_1^0, k_2^0, k_3^0) = (2, 3, 3, 3) \qquad Z=2$

Not unexpectedly (Fig. 4.4) the optimal sequence involves no category clustering. So the answer to the question asked earlier is "no".

But still, one cannot say that this issue is settled because the last time separation matrix was a contrived one and in practice, for the Aircraft Sequencing Problem, it never arises. Rephrasing therefore our question, we now ask the following: Could it be that for "real-world" input data for aircraft sequencing, this problem is so structured that the optimal sequence of landings is always achieved through category-clustering? This is a harder question to answer than the previous one. Nevertheless, one can find counterexamples to demonstrate that again, the answer to the question is "no":

Assume that category 1 consists of B747's, category 2 of B707's and category 3 of DC-9's. Under certain well defined conditions concerning the landing velocities, the length of the common final approach and other information, the time separation matrix for this problem is the following* (in seconds):

$$[t_{ij}] = \begin{bmatrix} 96 & 181 & 228 \\ 72 & 80 & 117 \\ 72 & 80 & 90 \end{bmatrix}$$

The numbers of passengers are $P_1=300$, $P_2=150$ and $P_3=100$.

Let us now examine two cases, identical in all inputs, except one: The number of wide-body jets, which is equal to 2 in case 6 and 1 in case 7:

<u>Case 6:</u> $(i_0, k_1^0, k_2^0, k_3^0) = (2, 2, 5, 5)$ $Z=2$

<u>Case 7:</u> $(i_0, k_1^0, k_2^0, k_3^0) = (2, 1, 5, 5)$ $Z=2$

Case 6 (fig. 4.5) exhibits the familiar "category-clustering" behavior which we encountered earlier: The optimal landing sequence starts with

*A derivation of this specific time separation matrix is presented in Appendix B.

all the landings of category 1, proceeds with all the landings of category 2 and ends with all the landings of category 3. The Total Passenger Delay is 936,750 passenger-seconds.

Since Case 7 is a slight variation of case 6, one might expect a similar optimal solution as well. Surprisingly enough, however (Fig. 4.6), the optimal solution in Case 7 is entirely different:

First of all, category 1 (which now has one member), loses the "first-to-land" privilege it held in Case 6. But, more interestingly, the optimal sequence shows the single B747 <u>being inserted</u> between two groups of B707's: One group of 4 planes, which leads the queue, and a single B707 which follows the B747! The remaining sequence consists of all the DC-9's. The total Passenger Delay is 758,550 passenger-seconds.

This is certainly a very peculiar behavior. It is not possible to decrease the Total Passenger Delay below that value of 758,550, no matter what rearrangement is tried upon. Two intuitively "obvious," but provably unsuccessful strategies are:

1) to move the B747 <u>at the head</u> of the queue, in front of all the B707's. This would result in a Total Passenger Delay of 766,350 (Fig. 4.7a).
2) to move the B747 even further downstream: between the B707's and the DC-9's. This would result in a Total Passenger Delay of 761,600 (Fig. 4.7b).

Before summarizing our observations, let us see one more instance of peculiar behavior:

<u>Returning to our first time separation matrix,</u> (Cases 1 through 4), we examine the following two cases:

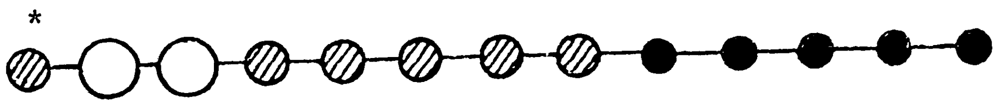


Fig. 4.5 : Case 6.

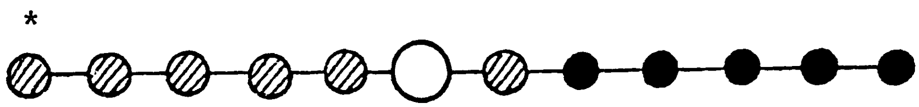
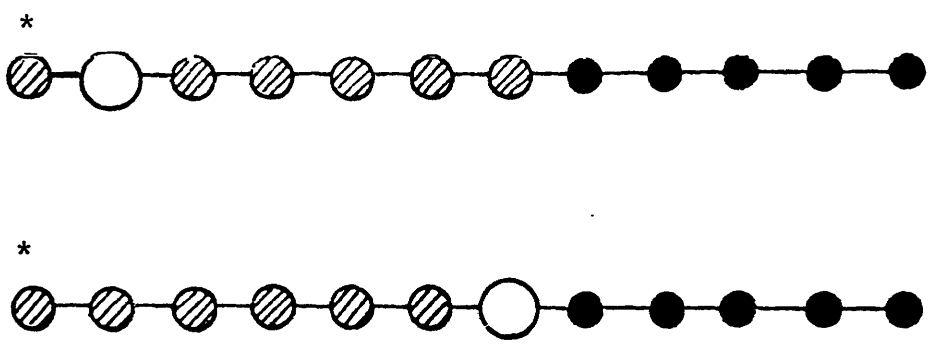


Fig. 4.6 : Case 7.



(a)

(b)

Fig. 4.7.

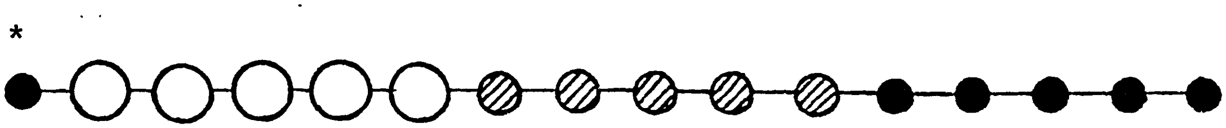


Fig. 4.8 : Case 8.

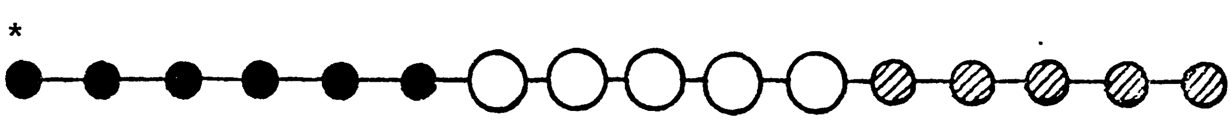


Fig. 4.9 : Case 9.

Case 8: P_1=110, P_2=110, P_3=120, $(i_0, k_1^0, k_2^0, k_3^0)$ = (3, 5, 5, 5), Z=2

Case 9: P_1=110, P_2=110, P_3=130, $(i_0, k_1^0, k_2^0, k_3^0)$ = (3, 5, 5, 5), Z=2

In other words the only difference between these two cases is the number of passengers in category 3: 120 vs. 130.

It turns out that this small difference is sufficient to lead to entirely different optimal sequences: Both solutions exhibit category-clustering, but in Case 8 the order is 1-2-3 while in Case 9 the order is 3-1-2. (Figures 4.8, 4.9.) The Total Passenger Delays incurred are 1,087,000 and 1,121,500 respectively.

4.4 Directions for Subsequent Investigation

The primary purpose for presenting the above examples was to demonstrate that our ASP algorithm indeed works.

Nevertheless, we cannot avoid noticing that under some circumstances, the pattern of the solution is itself specially structured (category-clustering) while under other, as yet unpredictable, circumstances this pattern is upset. Thus, one can ask questions like the following:

1) When do categories cluster?
2) Why does the optimal sequence of case 7 take this particular form?
3) Why did a difference of only 10 passengers per category change entirely the optimal sequence of Case 8 to that of Case 9?

We shall attempt to shed some light on these and related issues in Appendices A, B, C and D of this report.

Our immediate task is different however. We shall extend the capabilities of our algorithm by examining how we can solve the ASP when priority considerations are incorporated. In this respect, the assumption of the current unconstrained case that at any stage of the sequencing procedure we are free to assign the next landing "slot" to <u>any</u> of the remaining aircraft, will be dropped. The next Chapter will introduce, formulate and solve this new version of the problem.

CHAPTER 5

ASP-CONSTRAINED POSITION SHIFTING (CPS): DYNAMIC PROGRAMMING SOLUTION

5.0 Introduction: Meaning of CPS

A basic assumption in the formulation of the ASP presented in the previous Chapter was that we would not take into account any priority considerations. This was like hypothesizing that all planes in the system arrived simultaneously. We started "counting time" from the instant the zeroth airplane landed at the airport and we did not bother with the actual order in which these planes had arrived into the system. We considered ourselves free to form our own landing sequence in any way we wished, namely at any stage of our procedure we could assign the next landing slot to any of the aircraft remaining in the queue.

In practice, of course, the situation is somewhat different: The aircraft in our "reservoir" could not have arrived all at once, but in some order. To neglect that order totally, would most likely result in repeatedly biased decisions in favor of certain category types and against other categories. For example, we could note in most of the cases presented in the previous Chapter, that category 3 (light jets) was assigned to land last. This assignment is certainly a biased decision against category 3, since it is likely that some airplanes of this category may have arrived in the terminal area much earlier than planes of other categories which are likely to be assigned a high landing priority by the algorithm.

The main point of our argument is not "bias" or "fairness" per se but rather concerns the stability characteristics of our sequencing philosophy, particularly in the real-world "dynamic" environment. If we try to apply the algorithm of the previous chapter as it is to a dynamic environment, the following may happen: Each time a new airplane is added to our "reservoir" we shall have to take this airplane into account and run the algorithm again with this new piece of information as part of the input. If we adopt this "updating" scheme, it is entirely conceivable that some airplanes (for example those of category 3) continuously remain assigned to the end of the queue, just because there exist other airplanes with more favorable characteristics.

It is also conceivable that the optimal aircraft sequences resulting from two consecutive updates will bear no relationship at all with one another. For example, an airplane of category 3 which is just about to land, may be shifted back at the end of the sequence if a more "favorable" airplane appears in the meanwhile, and in fact, may be denied permission to land forever as long as some more favorable airplanes keep arriving. As mentioned also by Dear [DEAR 76], any "dynamic" sequencing scheme which is realized by continuous updates based on "global reoptimizations" of the current system, is likely to produce "global shifts" in the landing positions from one update to the next one as well.

We should still keep in mind, of course, that if "fairness", or "stability" is the only important issue, then the easiest thing to do is to return to our First-Come-First-Served discipline and simply assign landing slots according to the order in which the aircraft have arrived at the terminal area*. Clearly, this is also an undesirable prospect. So we should try to find a scheme which is somewhere between the provably unstable unconstrained case and the inefficient FCFS discipline. It turns out that such a scheme has already been suggested, although barely investigated in depth. It is called Constrained Position Shifting and consists of the following:

Suppose that our aircraft "reservoir" is ordered, namely we know which of these aircraft has arrived first, which second, etc. Suppose also that there is a rule which forbids any landing sequence which results in the shifting of the initial position of any particular aircraft by more than a prescribed number of single steps, upstream or downstream in the sequence. We shall call this number Maximum Position Shift (MPS). Thus, if MPS=3, for example, an aircraft occupying the 12^{th} position in the initial string of arriving aircraft, can land potentially anywhere from the 9^{th} to the 15^{th} position in the actual optimal landing sequence, but cannot be assigned a position outside this interval.

Other than for these new priority constraints, our problem remains as it was formulated earlier. It is clear that this new version has two additional inputs:

a) The initial sequence of aircraft.

*As Dear [DEAR 76] points out, two distinct FCFS schemes can appear in the ASP. The first sets priorities according to the order at which each aircraft enter the Terminal Area System. This event takes place at a certain distance from the runway, of the order of 50 n. miles. Due to the fact that the transit time from System Entrance to the runway varies with the aircraft (because of different velocities), a second FCFS scheme emerges: This sets priorities according to the projected time at which each aircraft are expected to arrive at the runway. The former discipline is termed FCFSSE (System Entrance) and the latter FCFSRWY (Runway). Throughout this dissertation we shall use FCFS for FCFSRWY.

b) The value of MPS.

We make the following additional observations:

1) The version of the problem which we shall examine is again "<u>static</u>", namely no intermediate arrivals will be considered. We shall again start counting time at the instant the zeroth plane lands (t=0). All delays that have been sustained prior to that time cannot be changed ("sunk costs") and, in view of our linear objectives, will be ignored. Despite the fact that we shall again examine a "static" problem, the importance of the MPS constraints in the equivalent "dynamic" problem is easy to recognize. With MPS constraints, any aircraft occupying the ith position in the initial queue is guaranteed to land occupying a position in the landing sequence which falls between i-MPS and i+MPS. So any updating scheme in the "dynamic" version which keeps track of the above constraints, automatically takes care of the biases likely to occur in the equivalent unconstrained case and all airplanes are guaranteed to land sooner or later. We shall come back to the issue of "dynamic" sequencing in Chapter 7.

2) Clearly, MPS=0 corresponds to the FCFS discipline, where no "optimization" is really involved.

3) On the other hand, if T is the total number of aircraft in our "reservoir", then the case MPS$\geq$T-1 corresponds essentially to the unconstrained case, because there is no way that an aircraft among a set of T airplanes can be shifted by more than T-1 positions.

4) Letting υ(MPS) be the optimal value of the problem according to some objective (minimization of Last Landing Time or of Total Passenger Delay), and for a specific value of MPS, then, everything else being equal,

we shall have:

$$\upsilon(MPS_1) \geq \upsilon(MPS_2)$$

if and only if

$$MPS_1 \leq MPS_2$$

We can see this by the observation that we cannot possibly do better (i.e. reduce our costs) by reducing MPS, because by doing so we will essentially be reducing the number of feasible final sequences (tightening of the constraints).

5) Letting also $\upsilon(\infty)$ be the optimal value of the problem without any MPS constraints, then the following are true:

$$\upsilon(MPS) \geq \upsilon(\infty) \qquad \text{for } MPS < T-1$$

and $$\upsilon(MPS) = \upsilon(\infty) \qquad \text{for } MPS \geq T-1$$

6) If there exists a value of MPS so that:

$$\upsilon(MPS) = \upsilon(\infty)$$

then we shall also have

$$\upsilon(MPS') = \upsilon(\infty) \text{ for all } MPS' \geq MPS$$

7) Similarly, if there exist two values of MPS, $MPS_1 < MPS_2$ such that:

$$\upsilon(MPS_1) = \upsilon(MPS_2)$$

then we shall also have:

$\upsilon(MPS_1) = \upsilon(MPS) = \upsilon(MPS_2)$ for all values of MPS between MPS_1 and MPS_2.

The CPS problem was tackled by Dear [DEAR 76] in conjunction with the "dynamic" problem, roughly as follows:

Each time a new aircraft enters the system, a "local reoptimization" of the tail of the existing (currently "optimal") queue is performed. For any given value of MPS, this "local reoptimization" concerns only the last MPS airplanes of the queue, which, together with the newly entered airplane are eligible for possible rearrangement. Thus, all (MPS+1)! combinations of possible tail sequences are evaluated by exhaustive enumeration and the best sequence is chosen. No reoptimization of the remainder of the queue is considered.

This procedure is computationally adequate only for small values of MPS (up to 6 for example). In addition, it is clear that the solutions produced by the procedure are suboptimal, because at each iteration, a part of the queue is "frozen" and the optimization which is performed on the remainder is local. Thus, while it is conceivable that the appearance of the new airplane might create rearrangements in the queue beyond MPS positions upstream (in a kind of chain-reaction), this possibility is ruled out by the proposed procedure.

5.1 Outline of Solution Approach Using Dynamic Programming.

Our Approach to the CPS problem will be more sophisticated than the above. We shall see that it will not be limited to small values of MPS and that the solutions will not be suboptimal. Our algorithm will be seen to solve the CPS problem for any value of MPS and still remain within polynomially bounded execution times with respect to the number of planes per category. We again remind the reader that we solve here the "static" case. The following arguments will explain the rationale of the approach.

1) To describe the initial sequence of aircraft we shall use the notation $(i_0, i_1, \ldots, i_T)$ where, again, i_0 is the zeroth -landed category and category i_j holds the j^{th} position in the initial sequence. $i_j \varepsilon \{1, 2, \ldots, N\}$ for $j=0,1,\ldots,T$.

For example, $(i_0, i_1, i_2, i_3, i_4) = (2,1,3,1,2)$ means that the zeroth landed category is 2, and that the 1st and 3rd positions are held by aircraft of category 1, the 2nd position by an aircraft of category 3 and the 4^{th} position by an aircraft of category 2.

2) If we are to keep the notation of the D.P. approach presented in the previous Chapter, we should try to find a way to translate the MPS constraints into a formulation compatible with that of the D.P. approach. The next observation is an indication that this compatibility may be achieved.

3) It is logical to assume that an "internal" FCFS discipline exists among airplanes belonging to the same category. The major consequence of this fact is that if at any stage we know $(L,k_1,\ldots,k_N)$ as they were defined in the previous Chapter, then we know not only <u>how many</u>, but also specifically <u>which</u> airplanes per category have landed so far. This means that the state representation $(L,k_1,\ldots,k_N)$ used in the unconstrained case is <u>sufficient</u> to describe the system in the CPS case too.

4) It should be clear that in general, the effect of the MPS constraints will be to <u>reduce</u> the feasible state space, namely there may be state configurations which cannot be feasible. As an example of an infeasible configuration, we consider $(L,k_1,k_2)\equiv(1,1,2)$ with respect to the

initial sequence $(i_1,i_2,i_3,i_4,i_5,i_6) \equiv (2,1,2,2,1,1)$ with N=2 and MPS=1.

Marking by * the landed airplanes and by $ the last landed airplane we will have:

$$\begin{array}{cccccccc} (& 2, & 1, & 2, & 2, & 1, & 1 &) \\ & * & * & & & * & & \\ & & & & & \$ & & \end{array}$$

We can see that the last landed airplane ($) has landed <u>third</u> while its initial position was <u>fifth</u>. A shift of 2 is not permissible for MPS=1, so (1,1,2) is infeasible.

Before formalizing this observation, let us introduce some new notation:

(i) Let k_j^{max} be the total number of airplanes belonging to category j in the initial sequnce $(j=1,\ldots,N)$.

(ii) Let $s_j \equiv k_j^{max} - k_j$. This is the number of airplanes belonging to category j which <u>have</u> landed, given that k_j airplanes of this category <u>have not</u> landed $(j=1,\ldots,N)$.

(iii) Let $m \equiv \sum_{j=1}^{N} s_j$. This is the total number of airplanes landed, therefore the position held by the last landed airplane.

(iv) Let, finally, $LAST(j,s_j)$ be the position in the initial sequence $(i_1,\ldots,i_T)$ of the s_j^{th} airplane of category j, if we start counting from i_1. This position can be uniquely determined from the initial sequence itself. By convention $LAST(j,0) \equiv 0$ for every j.

Adopting the above notation, we can state the following proposition:

A <u>necessary</u> condition for $(L,k_1,\ldots,k_N)$ to be feasible is that $|m-LAST(L,s_L)| \leq MPS$ (5.1)

We can see this by the fact that m is the final position category L is assigned, while LAST(L,s_L) was its initial position. The absolute value of their difference must be no more than MPS, according to the MPS constraint.

5) It should be clear that condition (5.1) alone is <u>not sufficient</u> for feasibility. As an example consider the combination $(L,k_1,k_2,k_3) \equiv (1,1,0,1)$ in the initial sequence $(i_1,i_2,i_3,i_4,i_5,i_6) \equiv (2,1,3,1,2,1)$, with N=3 and MPS=0. It will turn out that we have infeasibility despite the fact that the combination does satisfy (5.1).

In fact, we will have:

```
(2, 1, 3, 1, 2, 1)
 *  *     *  *
          $
```

Here $s_1=s_2=2$ and $s_3=0$, so $m=s_1+s_2+s_3=4$.
Also, LAST(L,s_L)=LAST(1,2)=4 and since $|4-4| = 0 =$ MPS, it follows that (5.1) holds.

<u>Nevertheless</u>, if we examine what can possibly happen next, in this example, we can see that there are only two possibilities, since $k_2=0$:

a) We can decide to land category 3: But this is forbidden, since we would assign the 5th position to category 3 while its initial position was the 3rd and MPS=0.

b) We can decide to land category 1: But this is also forbidden, since we would assign the 5th position to category 1 while its initial position was the 6th and MPS=0.

We therefore conclude that since all potential "next" combinations of (1,1,0,1) are infeasible, (1,1,0,1) itself is infeasible, despite the fact that it did satisfy (5.1).

6) A special case arises when $k_j=0$ for every j. Then (5.1) is also sufficient for feasibility, since we already have reached the end of the sequence and need not worry about what will happen next.

We are now in a position to state the following theorem for feasibility, the proof of which should be clear from the observations we have made so far. We will assume that we follow the notation introduced in paragraph 4 above;

RECURSIVE FEASIBILITY THEOREM:

A combination $(L,k_1,\ldots,k_N)$ is feasible with respect to an initial sequence and an integer MPS, if and only if both (A) and (B) are true:

(A) $|m\text{-}LAST(L,s_L)| \leq MPS$

(B) Letting $X = \{y: 1 \leq y \leq N,\ k_y > 0\}$, either one of the following is true:

(B1) X is empty. (i.e. all planes have landed.)

(B2) There exists an x in X such, that the combination $(x,k'_1,\ldots,k'_N)$ is feasible, where:

$$k'_j = \begin{cases} k_j-1 & \text{if } j=x \\ k_j & \text{otherwise} \end{cases} \qquad \text{for } j=1,\ldots,N$$

7) As the name of the above Theorem suggests, feasibility in the CPS problem is of a recursive nature. This means that the feasibility

(or infeasibility) of a certain state combination not only depends on whether that particular state itself satisfies a certain relation (here, inequality (5.1) only, but in general more than one relations), but on whether there exists at least one "next" state combination which we know is feasible. This special nature of feasibility will be used in order to create for all states the information on whether they are feasible or not. This will be described later.

8) As far as the form of the optimality recursion is concerned, it is not difficult to see that only minor modifications are necessary. In fact, if $(L,k_1,\ldots,k_N)$ is infeasible, then we do not have to execute the recursion at all. If $(L,k_1,\ldots,k_N)$ is feasible, then the recursion is:

$$V_Z(L,k_1,\ldots k_N) = \underset{x\in X_1}{\mathrm{Min}}\ [W_Z\cdot t_{L,x}+V_Z(x,k_1',\ldots,k_N')]$$

which is the same recursion as (4.1), with the only difference that now we have to search among the elements of X_1, i.e. the set of x's for which $(x,k_1',\ldots,k_N')$ is feasible. If $k_1=\ldots=k_N=0$, there are no "next" states and $V_Z(L,k_1,\ldots,k_N)=0$ (Provided of course $(L,k_1,\ldots,k_N)$ is feasible, for which only (5.1) is necessary to hold). If, on the other hand there is at least one $k_j\neq 0$, then we have to examine all the feasible "next" states. The existence of at least one of these states is guaranteed by our Recursive Feasibility Theorem, if $(L,k_1,\ldots,k_N)$ is feasible.

5.2 The Dynamic Programming Algorithm.

With the above considerations in mind, we can see that the solution algorithm will consists essentially of three parts, the following:

1) The "Screening" part, where we determine and store information concerning whether each $(L,k_1,\ldots,k_N)$ is feasible or not. We do this by

using backward recursion, starting from $(L,0,0,\ldots,0)$ where only (5.1) is needed and proceeding lexicographically up to $(L,k_1^{max},\ldots,k_N^{max})$ for $L=1,\ldots,N$ according to our Theorem.

2) The "Optimization" part, where we apply the optimality recursion (4.1) only for feasible states, using the information created in the "Screening" part. Whenever we apply the recursion we examine only feasible potential "next" states.

A parenthetical note at this point is that these two recursions <u>need not</u> be separate but may as well be <u>merged</u> together and be executed simultaneously. This may have certain computational advantages (as we shall see below) but has the disadvantage of coupling the two recursions together. A decoupling of the two recursions is better for sensitivity analysis: for example if we change objective or time matrix we don't have to execute the "Screening" part again but may move directly to the "Optimization" part.

3) The "Identification" part, which will essentially be the same as the one in Chapter 4. A minor difference is in Step 2 and of $L_m=0$; x now has to be chosen from the set of <u>feasible</u> combinations $(x,k_1,\ldots,k_N)$. The major difference is that by contrast to the unconstrained case, this part of the algorithm cannot be used as many times as one wishes, but from the fact that it is the initial sequence itself, together with MPS, which determine feasibility, and once either of them is changed one has to perform parts 1 and 2 of the solution procedure again.

5.3 Computational Effort and Storage Requirements

Let us examine first the case where "Screening" and "Optimization" are separate precedures. In this case, we must create a new array (in addition to the existing arrays V_Z and $NEXT_Z$) to store feasibility information. This can be a logical array which we may call FEASBL$(L,k_1,\ldots,k_N)$ with values "true" or "false" depending on whether $(L,k_1,\ldots,k_N)$ is feasible or not. The size of this array is $C = N.\prod_{j=1}^{N}(k_j^{max}+1)$ and the tabulation of it will take C iterations.

An interesting question is now the following: How many iterations will the subsequent "Optimization" part take?

We recall that we use the optimality recursion only for feasible states. The number of feasible states is a function of both the composition of the initial sequence of aircraft <u>and</u> MPS. For general values of MPS the exact behavior of this function is far from obvious, being most likely dependent on the quasi-random nature of the initial sequence, in an unknown fashion. We shall not get involved in analyzing this question in an exact way in this **report.** Instead, we shall contend ourselves with the following facts:

a) The function in question has the value T+1 if MPS=0 and the value C (as defined above) if MPS $\geq$ T-1. We can see the former fact by observing that if MPS=0 then <u>only one</u> sequence will be feasible, the initial FCFS sequence, itself: $(i_0, i_1,\ldots,i_T)$. Since at each of the T+1 stages of this sequence only one state combination is feasible, the total number of feasible states is T+1. On the other hand if MPS $\geq$ T-1 then we are essentially back to the unconstrained case and all C states will be characterized as feasible.

b) For intermediate values of MPS, the number of feasible states is a non-decreasing function MPS.

c) In any event, this function is bounded by the worst case performance which corresponds to the unconstrained case. Still, in this worst case the computational effort we have to experience is a polynomial function of the number of planes per category. In other words, the order of magnitude of computational effort due to the introduction of MPS constraints is the same as that of the unconstrained case.

Coming now to the case where "Screening" and "Optimization" are merged, we observe that we can save the storage space reserved for the array FEASBL. In fact we can put $V_Z(L,k_1,\ldots,k_N) = -1$ whenever the corresponding state is infeasible. The computational effort will decrease too (by a proportionality factor only, but not in its order of magnitude) for we shall determine feasibility on the spot, at each iteration of the optimality recursion. The mechanics of the algorithm will, in all other respects, remain unchanged. As we have already indicated earlier, this "merged" version is not particularly suitable for sensitivity analysis with respect to the objective function, time matrix and number of passengers.

5.4 Computer Runs-Discussion of the Results

Several example runs of the algorithm follow:

We shall examine various cases concerning a specific initial sequence of T=15 aircraft, divided into N=3 categories. The sequence, including the zeroth landed airplane i_0 is the following:

$$(i_0,i_1,i_2,\ldots,i_{15}) \equiv (2, 1, 1, 3, 2, 2, 3, 2, 1, 2, 3, 3, 2, 1, 2)$$

The time separation matrix is:

$$[t_{ij}] = \begin{bmatrix} 96 & 181 & 228 \\ 72 & 80 & 117 \\ 72 & 80 & 90 \end{bmatrix}$$

and the numbers of passengers are:

$$(P_1, P_2, P_3) \equiv (300,\ 150,\ 100)$$

We again use the notation Z=1 for Last Landing Time minimization and Z=2 for Total Passenger Delay minimization.

The next cases show the behavior of the optimal solution for various values of MPS and Z, as well as the values of the measures of performance.

Case 1 is trivial but is presented here for comparison purposes. It is for MPS=0 (Fig. 5.1). It is clear that the optimal sequence is identical to the initial one (FCFS) and independent of whether Z=1 or 2. The Last Landing Time, incurred is 1729 seconds and the Total Passenger Delay 2383800 passenger-seconds (an average of 851 seconds per passenger.)

Case 2 and 3 are for MPS=5. Case 2 (Fig. 5.2) is for Z=1. The optimal sequence exhibits a Last Landing Time of 1400 seconds (19% improvement over Case 1) and a Total Passenger Delay of 2033800 passenger seconds, or an average of 726 seconds per passenger (a 14% improvement over Case 1, despite the fact that Total Passenger Delay is not the objective of Case 2).

Case 3 (Fig. 5.3) is for Z=2. The optimal sequence exhibits a Last Landing Time of 1528 seconds (an 11% improvement over Case 1, but as expected not as high as in Case 2) and a Total Passenger Delay of 1883250 passenger-seconds, or an average of 673 seconds per passenger (a 20% improvement over Case 1).

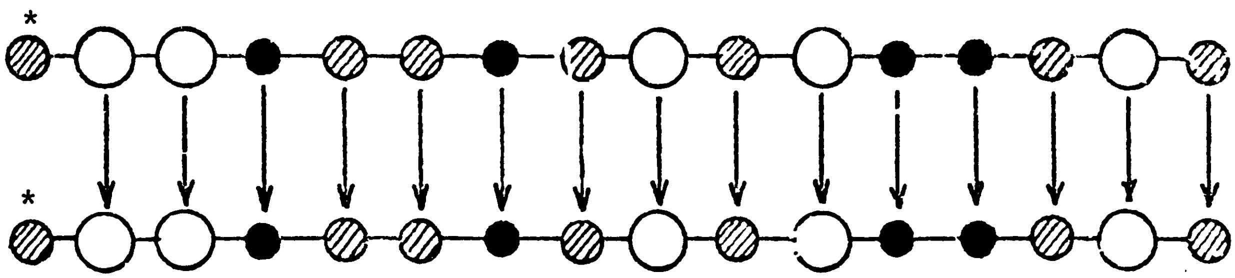


Fig. 5.1 : Case 1. In figures 5.1 through 5.5 the top sequence is the initial FCFS sequence and the bottom one is the optimal sequence. Arrows depict the position shifts of the various airplanes.

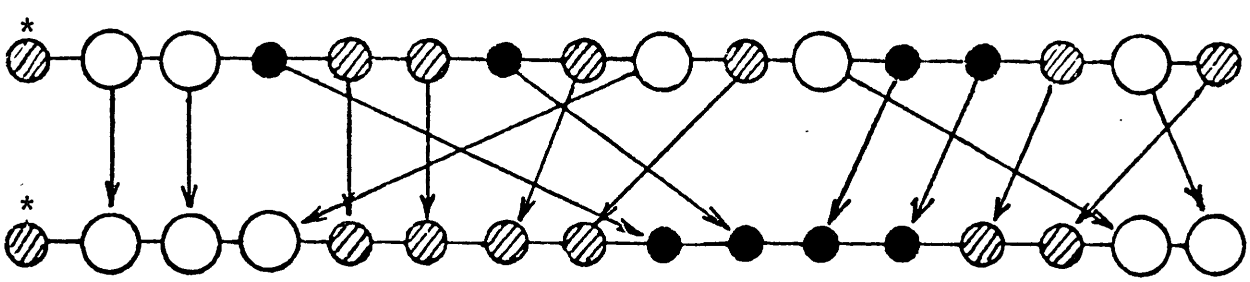


Fig. 5.2 : Case 2.

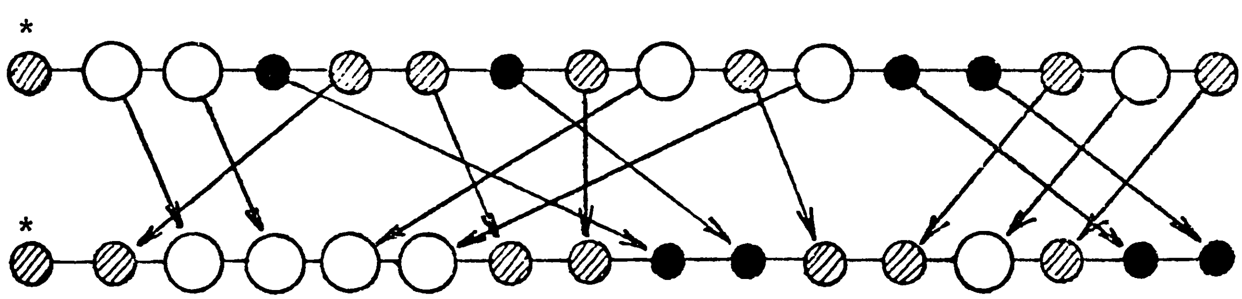


FIG. 5.3 : Case 3.

Note (Figs. 5.2, 5.3) that in accordance with our CPS rule, no position shift is greater in magnitude than MPS=5. Note also the different optimal sequences.

The final two cases 4 and 5 correspond to MPS=14 which reduces the problem to the equivalent unconstrained case. (MPS=∞.)

Case 4 is for Z=1. The Last Landing Time of the optimal sequence (Fig. 5.4) reaches its lowest achievable value of 1323 seconds (a 23% improvement over Case 1). By contrast, the Total Passenger Delay rises to 2241300 passenger seconds, an average of 800 seconds per passenger (only a 5% improvement over Case 1, i.e. worse than Cases 2 and 3. This sudden deterioration in the secondary measure of performance is a further indication that the two alternative objectives of the problem are not always "in harmony" with each other, i.e. that successive improvements with respect to one objective will not necessarily lead to successive improvements with respect to the other).

Case 5 is for Z = 2. The Last Landing Time of the optimal sequence (Fig. 5.5) is now 1424 seconds (a 17% improvement over Case 1) while the Total Passenger Delay reaches its lowest achievable value of 1664900 passenger seconds, or an average of 595 seconds per passenger (a 30% improvement over Case 1).

Note that the two last cases exhibit drastically different optimal sequences. Note also that although MPS is equal to 14, the actual maximum shifts achieved are lower (-10 in Case 4 and 9, -9 in Case 5). This last observation means that it is possible that the actual value of MPS for which the CPS problem reduces to the unconstrained problem is lower than T-1.

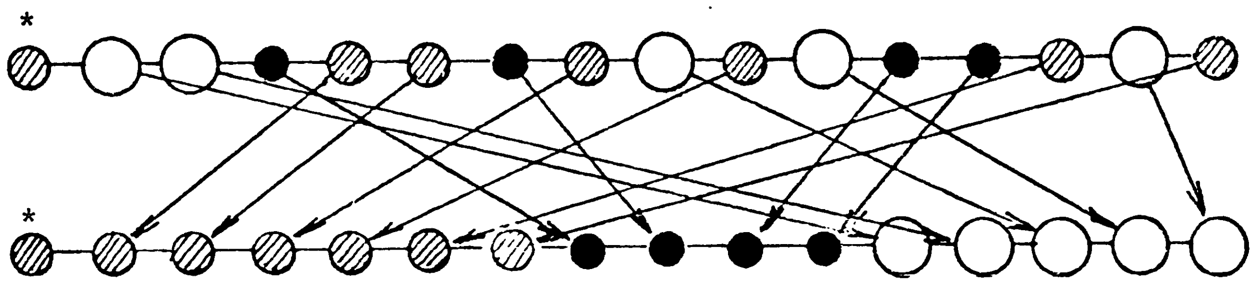


Fig. 5.4 : Case 4.

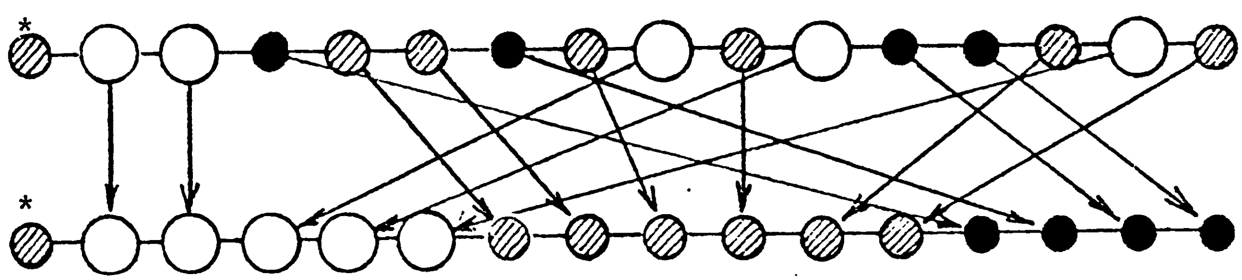


Fig. 5.5 : Case 5.

In fact, a value of MPS=10 would do for Z=1 and a value of MPS = 9 for Z=2 in this example.

In figures 5.6 and 5.7 we show for our example how the percent improvement over the FCFS sequence for each of the two measures of performance changes as a function of MPS. Solid lines indicate that the measure of performance in question is the objective to be minimized. Dotted lines show the behavior of the other measure of performance when the former is minimized. Not unexpectedly, the solid lines are non-decreasing, while we see that this is not generally the case for the dotted lines. Also, as expected, the solid lines are nowhere below the dotted lines, because the improvement of the measure of performance which is the objective of the problem (solid line) is the maximum improvement achievable.

The above examples conclude for the moment our discussion of the CPS problem. It has been seen that the concept of constrained Position Shifting takes care of several disadvantages of the equivalent unconstrained case especially when "dynamic" considerations enter the problem. In this chapter we developed an algorithm for solving the "static" version of the Aircraft Sequencing Problem for any value of MPS, leaving the "dynamic" version for later discussion (Chapter 7). This algorithm exhibits a computational effort which is of the same order of magnitude as that of the equivalent unconstrained case,namely a polynomial function of the number of airplanes per category.

In the next Chapter we shall return to the unconstrained case but this time the problem of sequencing aircraft in two parallel runways will be examined.

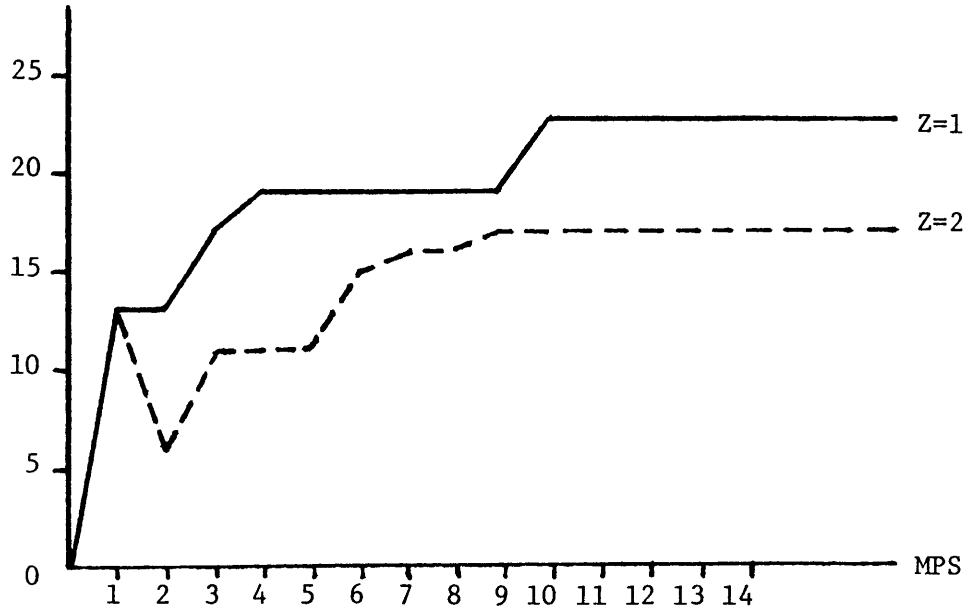


Fig. 5.6 : % improvement in LLT with respect to the FCFS discipline.

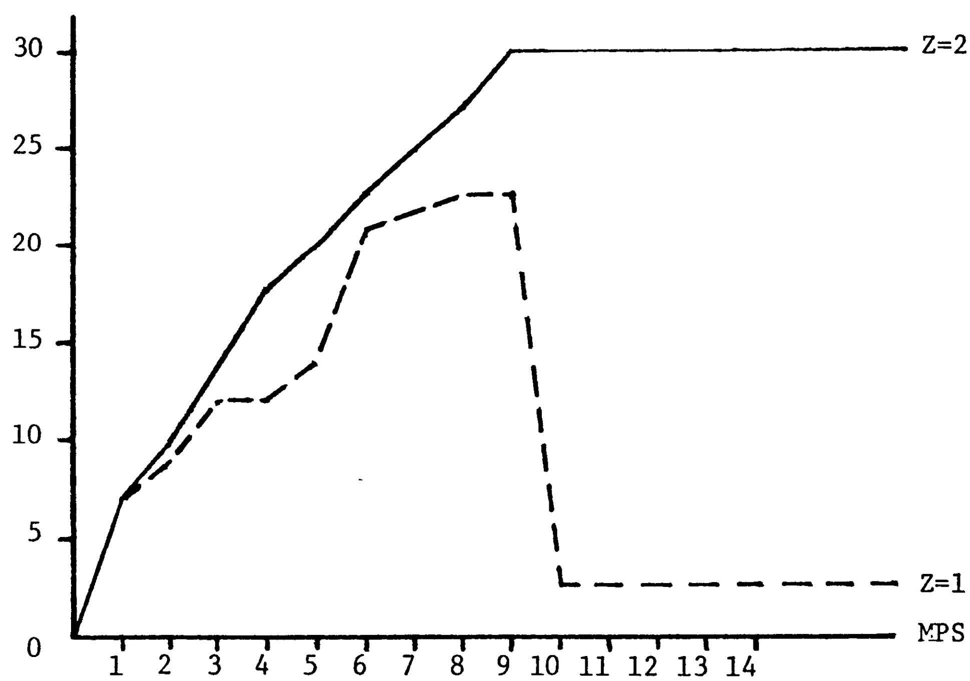


Fig. 5.7 : % improvement in TPD with respect to the FCFS discipline.

CHAPTER 6

THE TWO-RUNWAY ASP

6.0 Introduction: Formulation and Complexity of the Problem

All versions of the ASP examined so far concerned the single runway configuration, that is, the case where all the planes have to use the same runway to land.

In this Chapter we shall introduce and discuss certain issues connected to the case where two identical, parallel and independent runways are available. This configuration appears at many major airports in the world, where it is impossible to accommodate all the traffic in a single runway.

Let us state at the outset that this will not be a complete examination of the two-runway case, neither an attempt to find an "efficient" algorithm for this problem. Actually it is well known that the general problem of sequencing a number of tasks in two parallel processors, belongs also to the class of NP-complete problems [GARE 76]. What this chapter will attempt to do, is to link the two runway problem to the single runway problem and indicate an "elementary" solution procedure based on that connection. The following observations will establish our basic philosophy for looking at this problem:

1) We assume that we are dealing with an unconstrained situation, i.e. once again we neglect priority considerations. Later it will be seen that to include the latter in the two runway case would make the problem extremely difficult.

2) We also assume that this problem is also "static" namely no intermediate aircraft arrivals will be considered and the sequencing procedure ends as soon as all aircraft have landed.

3) The alternative objectives which will concern us here are almost equivalent to the ones which we have examined so far, namely Last Landing Time minimization (Z=1) and Total Passenger Delay minimization (Z=2). The latter measure of performance is rather straightforward to envision. It consists of the sums of the waiting times for all passengers in this system, from t=0 (when our sequencing procedure starts) till the time each passenger lands. Coming to the first objective however, and since we shall actually observe two Last Landing Times, one for each runway, the question is: "What does Z=1 mean in the two-runway case?"

It is not difficult to give an answer to this question. If t_1, t_2 are the last landing times for runways 1, 2 respectively, it is clear that the Last Landing Time for the combined system of both runways should be the largest of t_1, t_2. This will correspond to the time at which the last (for both runways) aircraft lands. Z=1 therefore implies an attempt to minimize the maximum of the two Last Landing Times observed at the two runways. Because of this, our problem can also be called a minimax problem.

4) It is clear that the two-runway problem must be at least as difficult as the equivalent single runway problem. In fact, in the two runway case an additional decision we have to make (besides the sequencing strategy per se) concerns "what-aircraft-goes-to-what-runway." In other words, we have to partition the set of airplanes, into those which will go

to Runway 1 and those which will go to Runway 2 and then we have to sequence these planes. At this moment of course, it is not clear at all that these two distinct decisions can be separated from one another and therefore executed sequentially. Our next observation deals with this issue.

5) Suppose for the moment that we have somehow decided on a particular partition, not necessarily the optimal one. In other words, given that at t=0 the composition of our aircraft "reservoir" is $(k_1^0,\dots,k_N^0)$ planes per category, suppose we have decided that $(x_1,\dots,x_N)$ of them should go to Runway 1 and the remainder $(k_1^0-x_1,\dots,k_N^0-x_N)$ should go to Runway 2. We then ask ourselves the following question: Given the above partition, how should the airplanes be sequenced on the two runways?

The answer to this question is that since we have already separated the airplanes, we have essentially constructed two independent sets, one for each runway. Since no interaction takes place between these two sets, it makes sense to sequence each set individually, by applying the single runway algorithm described in Chapter 4. So, if i_{01}, i_{02} are the zeroth-landed categories on Runways 1 and 2* respectively, then the initial state combination for Runway 1 will be $(i_{01}, x_1,\dots,x_N)$ and for Runway 2 $(i_{02},k_1^0-x_1,\dots,k_N^0-x_N)$.

6) The above argument settles the question of what happens if we know the partition. But as we mentioned earlier, this is what we have to decide upon. So now the problem reduces to finding a partition

*We assume that i_{01}, i_{02} have landed simultaneously at t=0. The relaxation of this assumption will not create major problems in our subsequent reasoning.

$(x_1^*,\ldots,x_N^*)/(k_1^0-x_1^*,\ldots,k_N^0-k_N^*)$ which is <u>optimal</u>.

To <u>evaluate</u> a particular partition $(x_1,\ldots,x_N)/(k_1^0-x_1,\ldots,k_N^0-x_N)$ we simply look into the appropriate entries of the array V_Z which we have previously prepared by a single pass of the "optimization" part of the single runway algorithm. So the measure of performance $M_Z(\vec{x})$ corresponding to the vector $\vec{x} = (x_1,\ldots,x_N)$ and the objective Z, is given by:

$$M_Z(\vec{x}) \equiv \begin{cases} \text{Max}[(U_Z(1),\ U_Z(2)] & \text{if } Z = 1 \\ \\ U_Z(1) + U_Z(2) & \text{if } Z = 2 \end{cases} \tag{6.1}$$

where $U_Z(1) \equiv V_Z(i_{01},x_1,\ldots,x_N)$ (6.2)

and $U_Z(2) \equiv V_Z(i_{02},k_1^0-x_1,\ldots,k_N^0-x_N)$ (6.3)

We recognize $U_Z(1)$ and $U_Z(2)$ as the optimal (according to our objective) values of the individual sequencing problems for each of the two runways, <u>after</u> we have decided on the partition.

<u>To find the optimal partition</u>, we have to find a vector $\vec{x}=(x_1,\ldots,x_N)$, so that $M_Z(\vec{x})$ (given by (6.1) through (6.3) above) is <u>minimized</u>.

7) At this point we do not propose anything other than a complete enumeration procedure for minimizing $M_Z(\vec{x})$. This procedure will examine essentially <u>all possible partitions</u> (i.e. all combinations of $(x_1,\ldots,x_N)$ with each x_j between 0 and k_j^0) corresponding to the initial composition $(k_1^0,\ldots,k_N^0)$ of our aircraft reservoir. This can be done in exactly $\prod_{j=1}^{N}(k_j^0+1)$ steps. If the entire tableau of V_Z is readily available*,

*This will happen if V_Z is in main storage rather than in auxiliary storage.

the computational effort associated with this exhaustive search will be of the same order of magnitude as the one for the "optimization" part of the single runway algorithm, namely a polynomial function of the number of airplanes per category.

6.1 Solution algorithm

With the above arguments in mind, we see that we can construct an algorithm for the two-runway problem which is essentially a post-processing procedure of the information created by a single pass of the "Optimization" part of the single runway algorithm (Chapter 4). It will consist of the following steps:

Step 1: Perform one pass of the "Optimization" part of the single runway problem. Store the values of $V_Z(L, k_1,\ldots k_N)$ and $NEXT_Z(L,k_1,\ldots,K_N)$ for all values of L (from 1 to N) and k_j (from 0 to k_j^{max}).

Step 2: For any particular composition of the aircraft reservoir $(k_1^0,\ldots,k_N^0)$ and initial conditions (i_{01}, i_{02}: zeroth-landed categories), provided each $k_j^0 \leq k_j^{max}$, do the following:

Examine all combinations of vectors $\vec{x} = (x_1\ldots,x_N)$ with $0 \leq x_j \leq k_j^0$ and select the one which minimizes the quantity:

$$M_Z(\vec{x}) \equiv \begin{cases} \text{Max}[V_Z(i_{01},x_1,\ldots,x_N),\ V_Z(i_{02},k_1^0-x_1,\ldots,k_N^0-x_N)]\,, \\ \quad \text{if } Z=1 \\ V_Z(i_{01},x_1,\ldots,X_N)+V_Z(i_{02},k_1^0-x_1,\ldots,k_N^0-x_N)\,, \\ \quad \text{if } Z=2 \end{cases}$$

Let $\vec{x}* = (x_1^*,\ldots,x_N^*)$ be the optimal vector.

Step 3: Using the information available from the already tabulated array $NEXT_Z$, perform two separate "identification" procedures (as described in the single runway problem in Chapter 4): One for Runway 1 with initial state $(i_{01}, x_1^*, \ldots, x_N^*)$ and one for Runway 2 with initial state $(i_{02}, k_1^0 - x_1^*, \ldots, k_N^0 - x_N^*)$.

Whenever a new aircraft reservoir and/or new initial conditions are given go to Step 2. For a new time separation matrix and/or new numbers of passengers per aircraft category, go to Step 1. Otherwise END

To understand now this algorithm, we give a small illustrative example:

Step 1: Suppose that N=2, Z=1, and $k_1^{max} = k_2^{max} = 2$. Suppose also that the single pass of the "Optimization" part of the single runway problem creates and stores the following values:

(L, k_1, k_2)	$V_1(L, k_1, k_2)$	$NEXT_1(L, k_1, k_2)$
1,0,0	0	-
2,0,0	0	-
1,0,1	120	2
2,0,1	90	2
1,0,2	210	2
2,0,2	180	2
1,1,0	80	1
2,1,0	80	1
1,1,1	200	1

(L, k_1, k_2) cont'd	$V_1(L,k_1,k_2)$ cont'd	$NEXT_1(L,k_1,k_2)$ cont'd
2,1,1	170	2
1,1,2	290	2
2,1,2	290	1
1,2,0	160	1
2,2,0	160	1
1,2,1	280	2
2,2,1	250	2
1,2,2	370	1
2,2,2	340	2

Step 2: Suppose now that we have a composition of our reservoir $(k_1^0,k_2^0)=(2,1)$ and initial conditions $(i_{01},i_{02})=(1,2)$. Then we do the following:

We examine all vectors $\vec{x}=(x_1,x_2)$ with $0 \leq x_1 \leq 2$, $0 \leq x_2 \leq 1$ and for each of them evaluate $M_1(\vec{x})$ as given above. Then we choose the combination which minimizes $M_1(\vec{x})$. The vectors we examine are:

$\vec{x}=(x_1,x_2)$	$V_1(i_{01},x_1,x_2)$	$V_1(i_{02},k_1^0-x_1,\ k_2^0-x_2)$	$M_1(\vec{x})$
(0,0)	0	250	250
(0,1)	120	160	160
(1,0)	80	170	170
(1,1)	200	80	200
(2,0)	160	90	160
(2,1)	280	0	280

So we see that each of the vectors $(x_1,x_2) = (0,1)$ and $(2,0)$ minimizes $M_1(x)$. (Multiple optimal solution.) Breaking this tie arbitrarily we select $(x_1^*,x_2^*)=(0,1)$ as the optimal partition.

Step 3: We now perform two identification procedures.

Runway 1: Start with $(i_{01}, x_1^*,x_2^*)=(1,0,1)$, next state is $(2,0,0)$.

Runway 2: Start with $(i_{02},k_1^0-x_1^*,k_2^0-x_2^*) = (2,2,0)$ next state is $(1,1,0)$ and final state is $(1,0,0)$.

The partition and sequencing of this example is depicted in Fig. 6.1

6.2 Computer Runs-Discussion of the results

Several example runs follow. We use a category mix we have used before. N=3 and the time separation matrix is (in seconds):

$$[t_{ij}] = \begin{bmatrix} 96 & 181 & 228 \\ 72 & 80 & 117 \\ 72 & 80 & 90 \end{bmatrix}$$

The numbers of passengers per category are $(P_1,P_2,P_3)=(300, 150, 100)$ The input for each run is the vector $(i_{01},i_{02},k_1^0,k_2^0,k_3^0)$ as defined earlier. A parameter for each run is our objective Z, as also defined earlier. The outputs for each run are:

a) The optimal partition of the initial set of aircraft: (x_1^*,x_2^*,x_3^*) aircraft go to Runway 1, the rest $(k_1^0-x_1^*,k_2^0-x_2^*,k_3^0-x_3^*)$ go to Runway 2.

b) The optimal sequences for each of the two runways. These will be depicted in accompanying **figures**.

c) Certain measures of performance:

For each individual Runway i ($i=1,2$) we have:

t_i = Last Landing Time for the set of aircraft landing there.

C_i = Total Passenger Delay for the same set.

$\bar{t}_i$ = Average per passenger delay for the same set.

Combined performance for both Runways: We shall have :

t = Last Landing Time.

C = Total Passenger Delay.

$\bar{t}$ = Average per passenger delay.

It is clear that the following relations are true:

$t = \text{Max}\,[t_1, t_2]$

$C = C_1 + C_2$

and $\dfrac{C}{\bar{t}} = \dfrac{C_1}{\bar{t}_1} + \dfrac{C_2}{\bar{t}_2}$

The cases we examine are the following :

Case 1: $(i_{01}, i_{02}, k_1^0, k_2^0, k_3^0) = (1,1,4,4,4)$, $Z=1$ (Fig. 6.2)

Optimal partition: Runway 1: (2,2,2) Runway 2: (2,2,2)

Individual Performance :

Runway 1: $t_1=636$, $C_1=418{,}500$, $\bar{t}_1= 380$

Runway 2: $t_2=636$, $C_2=418{,}500$, $\bar{t}_2= 380$

Combined Performance :

$t=636$

$C=837{,}000$

$\bar{t}=380$

Case 2: $(i_{01}, i_{02}, k_1^0, k_2^0, k_3^0) = (1,1,4,4,4)$, $Z=2$ (Fig. 6.3)

(Same as Case 1 except for the objective.)

Optimal Partition: Runway 1: (2,2,2) Runway 2: (2,2,2)

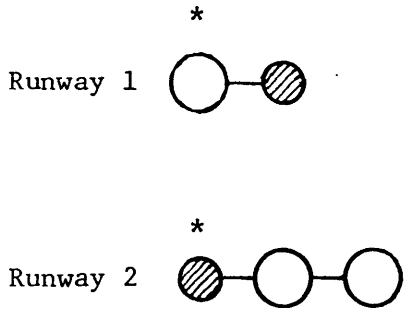


Fig. 6.1.

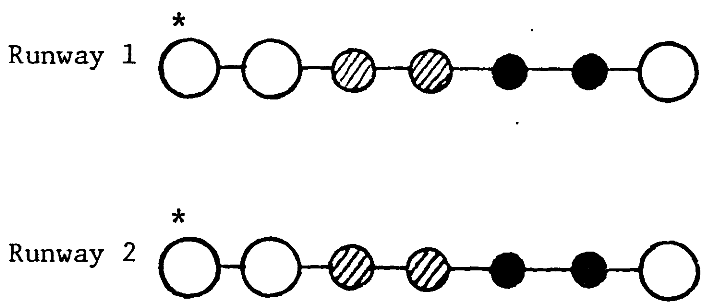


Fig. 6.2 : Case 1.

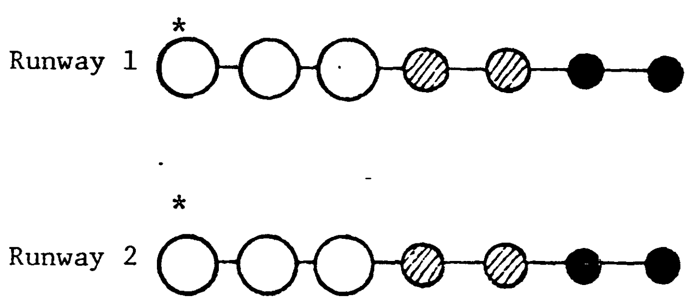


Fig. 6.3 : Case 2.

Individual Performance:

Runway 1: t_1=660 C_1=333,300 $\bar{t}_1$= 303

Runway 2: t_2=660 C_2=333,300 $\bar{t}_2$= 303

Combined Performance:

t = 660

C = 666,600

$\bar{t}$ = 303

We note several facts (see also Figs. 6.2 and 6.3).

1) In each case the sequences on the two runways are identical to one another.

2) The two cases exhibit the same optimal partition.

3) The two cases differ in the optimal sequencing and in their measures of performance.

A question which arises after eximining these cases is whether the two sequences on each runway always look "similar" (In Cases 1 and 2 they are identical). The next two cases will show that this is not always true:

Case 3: $(i_{01}, i_{02}, k_1^0, k_2^0, k_3^0)$ = (2,2,5,5,5), Z=1 (Fig. 6.4)

Optimal partition: Runway 1: (2,0,5), Runway 2: (3,5,0)

Individual Performance:

Runway 1: t_1=645 C_1 = 506,700 $\bar{t}_1$=460

Runway 2: t_2=664 C_2 = 691,200 $\bar{t}_2$=418

Combined Performance:

t = 664

C = 1,197,900

$\bar{t}$ = 436

Case 4: $(i_{01}, i_{02}, k_1^0, k_2^0, k_3^0) = (2,2,5,5,5)$, Z=2 (Fig. 6.5)

(Same as case 3 except for the objective)

Optimal partition: Runway 1: (0,5,3), Runway 2: (5,0,2)

Individual performance:

Runway 1: $t_1 = 697$ $C_1=362,100$ $\bar{t}_1=344$

Runway 2: $t_2 = 774$ $C_2=541,800$ $\bar{t}_2=318$

Combined performance :

$t = 774$

$C = 903,900$

$\bar{t} = 329$

We can note several facts (see also Figs. 6.4, 6.5):

1) In both cases, the partition between the two runways is totally asymmetric (e.g., Case 3: All planes of category 2 go to Runway 2, all planes of category 3 go to Runway 1, Category 1 is split).

2) The partitions of the two cases are different.

3) The same holds for the orders in the sequences.

We shall examine two more cases:

Case 5: $(i_{01}, i_{02}, k_1^0, k_2^0, k_3^0) = (3, 3, 1, 3, 5)$, Z=1 (Fig. 6.6)

Optimal partition: Runway 1 (0,0,4), Runway 2: (1,3,1)

Individual performance:

Runway 1: $t_1 = 360$ $C_1=90,000$ $\bar{t}_1=225$

Runway 2: $t_2 = 402$ $C_2=242,100$ $t_2=284$

Combined performance:

$t = 402$

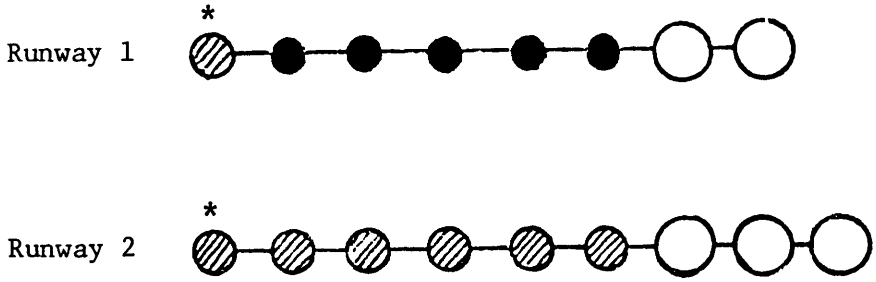


Fig. 6.4 : Case 3.

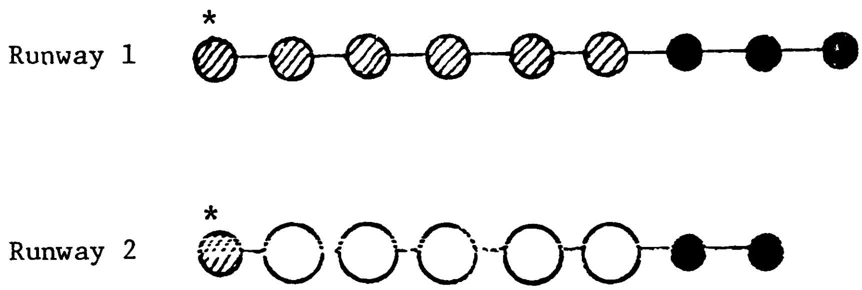


Fig. 6.5 : Case 4.

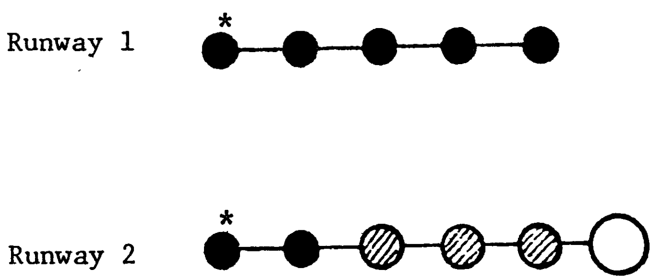


Fig. 6.6 : Case 5.

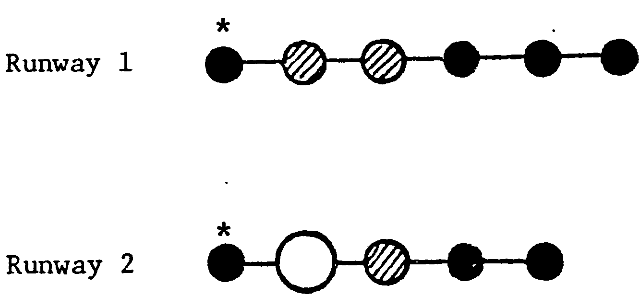


Fig. 6.7 : Case 6.

$C = 332{,}100$

$\bar{t} = 266$

Case 6 $(i_{01}, i_{02}, k_1^0, k_2^0, k_3^0) = (3, 3, 1, 3, 5)$, Z=2 (Fig. 6.7)

(Same as case 5 except for the objective)

Optimal partition: Runway 1: (0,2,3), Runway 2: (1,1,2)

Individual performance:

Runway 1: $t_1=457$ $C_1 = 146{,}100$ $\bar{t}_1=243$

Runway 2: $t_2=460$ $C_2 = 142{,}550$ $\bar{t}_2=219$

Combined performance:

$t = 460$

$C = 288{,}650$

$\bar{t} = 231$

We can note the following facts: (See also Figs. 6.6, 6.7)

1) Runway 1 in Case 5 is "dedicated" solely to landings of category 3, but not all of them land there. One is assigned to Runway 2.

2) All 4 sequencings (2 "similar" cases, 2 runways) bear no relationship to one another.

3) The Total Passenger Delays on each of the two runways in Case 5 are drastically different (C_2 is 169% larger than C_1!). However surprising this may be, it is indeed this partitioning and sequencing that actually minimizes the Last Landing Time (402 seconds).

4) A more general remark, motivated by Cases 5 and 6 is the following: It is often customary to consider the minimization of the largest of two quantities as being roughly equivalent to approximately

equalizing those quantities. The reasoning of course is that if the difference between the two quantities gets large enough, then it is likely that the largest of them is not minimal. Surprisingly enough, cases 5 and 6 provide a good counterexample to the above reasoning. In fact, the two Last Landing Times in Case 6 are 457 and 460, i.e. approximately equal. However, the strategy that minimizes the largest of the two is in Case 5, where these quantities are 360 and 402, i.e. significantly different from one another!

Counterexamples like the above should be a warning for exercising caution when trying to "guess" the properties of solutions of such minimax problems. In particular, one should be careful when a solution procedure is based on heuristics that try to take advantage of, supposedly "intuitively obvious" properties - properties that may conceivably not exist.

For instance, one could have based a solution algorithm for the partitioning problem of the two runway case on the "intuitively obvious", yet non-existent property that the optimal partition of the initial aircraft mix is, more or less, symmetric between the two runways. Cases 3 and 4 show that this is not in general true.

6.3 Further Remarks on the Two-Runway ASP

The examples which we presented exhibit a sufficient number of interesting characteristics to further stimulate one's curiosity on the two runway Aircraft Sequencing Problem. The "elementary" solution procedure we presented provided a scheme through which the information created by the single runway algorithm is utilized to obtain the optimal partitioning of the set of airplanes between the two runways. Nevertheless, this procedure provides no help for answering questions arising from the examination of the computer runs, such as the following:

1) When are the sequences on the two runways "similar" and when are they completely different?
2) What causes a particular category of aircraft to be assigned entirely to one runway while other categories are "split"?
3) Is there an underlying pattern in the partitioning and sequencing schemes that we can use to improve the solution efficiency?

In addition to the above, some additional issues can be addressed:

a) What happens if we introduce priority considerations (i.e. Constrained Position Shifting) for two runways?
b) Can we extend the proposed procedure to <u>three</u> (or more) independent runways?
c) What happens if the problem becomes "dynamic"?

We shall discuss such issues in PartIII of this report. It will be seen there that the degree of difficulty associated with most of these questions is considerably higher than any we have encountered so far.

6.4 Summary of the D.P. Approach to the ASP

This Chapter concludes our considerations on the Dynamic Programming Approach to various problems connected with the optimal sequencing of aircraft landings. In this respect, the following problems were considered:

(1) The unconstrained ASP - single runway

(2) the Constrained Position Shifting ASP - single runway

(3) The unconstrained ASP - two runways.

All the problems above were considered "static". Their "dynamic" versions will be discussed in Chapter 11. Algorithms were developed specifically for (1) and (2). For (3), an "elementary" algorithm, based on that of (1) was presented. All three algorithms exhibit computational efforts and storage requirements which are bounded by polynomial functions of the number of aircraft per category.

Appendices A,B,C, and D will examine, as mentioned in Chapter 4, several issues addressed there. These issues deal with "category clustering," the time separation matrix and other related problems. Among other things, we shall extensively investigate under what conditions certain well defined patterns are certain to occur in the optimal sequence and how the specific structure of the ASP affects these patterns. Certain other, less predictable patterns (like that of Case 7 in Chapter 4 (Fig. 4.6)) will be explained as well.

It will be seen that the approach used in these appendices constitutes a new way of looking at the ASP and as such, is essentially self-contained. Thus, no loss of continuity will occur should the reader decide not to examine this material. On the other hand, the extensive investigation of the same problem from a point of view different from the one used so far, may provide additional insight into the problem.

A description of the computer programs used for the ASP will be given in Appendix E.

PART III

FINAL REMARKS

CHAPTER 7

CONCLUSIONS AND DIRECTIONS FOR FURTHER RESEARCH

7.0 Introduction

In the single Chapter of this part of the report, we shall attempt to do the following:

1) Review the results of this work
2) Suggest several directions towards which this work can be extended.
3) Discuss the problems associated with these extensions.
4) Address several issues concerning the real-world implementation of the algorithms.

7.1 ASP: Review of the Results of this work

This work has developed algorithms for three versions of the general problem of sequencing aircraft landings at an airport. These were the following:

(1) Single runway airport - unconstrained case.

(2) Single runway airport - Constrained Position Shifting (CPS) case.

(3) Two-runway airport-unconstrained case.

In all three versions, the fact that the airplanes waiting to land, however numerous, could be classified into a relatively small number of categories, was exploited. Category classifications resulted in the development of dynamic programming algorithms, which for all three versions exhibit computational efforts and storage requirements which are polynomially bounded functions of the number of aircraft per category. These functions are exponential with respect to the number of categories, but this number is usually of the order of 3 and can be at most 5. All three versions of the problem were assumed "static", namely no new aircraft arrivals were assumed to occur during the time the existing aircraft land.

In all three versions, two alternative objective functions were considered. The first was the minimization of the Last Landing Time (LLT), namely now to sequence a given set of aircraft so that they can land as soon as possible. The second was the minimization of Total Passenger Delay (TPD), namely how to sequence the landing of the aircraft so that the sum of the waiting-to-land times for all passengers in the system is as low as possible. The two alternative objectives are handled by the dynamic programming algorithms with equal ease and produce, in general, different optimal solutions.

The CPS rules were seen to fit the dynamic programming procedure in a particularly adequate way, with no increase in the order of magnitude of the computational effort, for any value of the Maximum Position Shift (MPS). This has been a substantial improvement over the complete enumeration procedure through which it was initially proposed that the CPS problem be tackled [DEAR 76].

The two-runway problem was solved as a post-processing of the information created by a single pass of the single runway unconstrained algorithm. The complete enumeration approach to finding the optimal partition of the set of aircraft between the two runways was seen not to increase the order of magnitude of the computational effort of the problem.

In addition to the dynamic programming approach to the problem, an extensive investigation of the underlying structure of the problem from a different point of view was presented in Appendices A through D. Specifically, issues like group clustering, the influence on the solution of the structure of the time separation matrix and of the number of passengers per aircraft etc., were considered.

We shall now present what we feel can be done to extend the work on the ASP in this report and what we think are the main problems associated with that.

7.2 ASP: "Dynamic" Case

The arrival of the particular aircraft in the vicinity of the near terminal area is signalled to the air traffic controller at a point in time substantially prior to the instant when the aircraft actually starts its final landing manoeuvre. This point in time is when the aircraft enters the region which is under ground control (this is usually a cylindrically shaped region with a radius of approximately 50 nautical miles centered around the airport and of altitude approximately 10,000 feet). It is also known as System Entrance Time (SET).

Assuming no other aircraft are in the system, the aircraft will traverse the terminal area, gradually losing speed and altitude, execute its landing manoeuvre and finally land, at a time which we shall call Preferred Landing Time (PLT). This time depends on such parameters as the characteristics of the terminal area, the aircraft speed, pilot preferences etc. Upon entrance into the terminal area (t=SET), the PLT of a particular aircraft can be calculated.

The PLT of a particular aircraft can be taken to be the earliest time when this aircraft can land. Its Actual Landing Time (ALT) may, in case of conflict with other aircraft, occur later. The simplest case of such a conflict is shown in Figure 7.1.

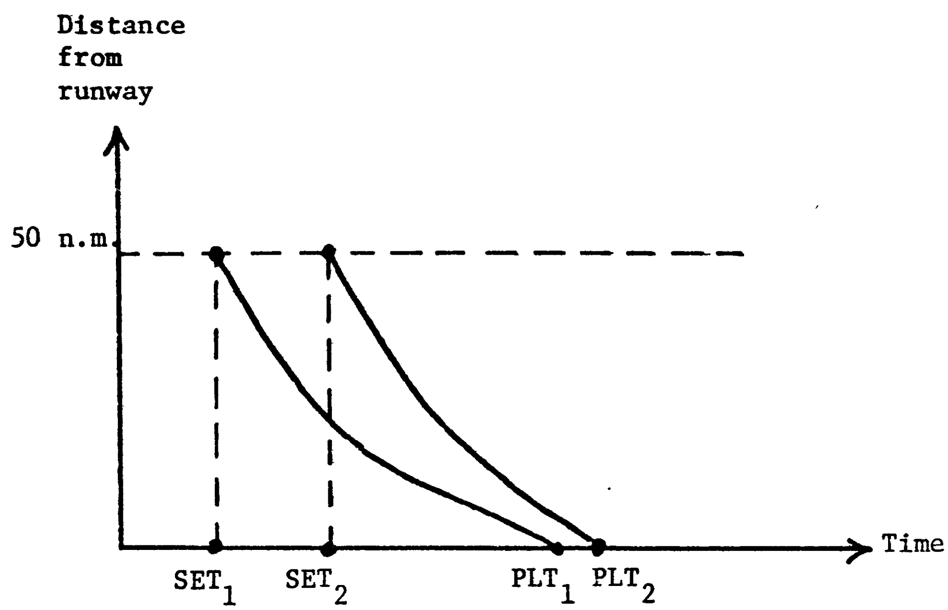


Fig. 7.1.

At $t=SET_1$ and $t=SET_2$, aircraft 1 and 2 enter our terminal area (i.e. enter the (say) 50 n.m.-radius area that is under ground control). Let PLT_1 and PLT_2 be the preferred landing times of these two aircraft. Suppose now that $PLT_1 < PLT_2$ but $PLT_2-PLT_1 < t_{12}$ where t_{12} is the minimum permissible time interval between the landing of aircraft 1, followed by the landing of aircraft 2 (t_{12} is the element of the time separation matrix that corresponds to our case). It is apparent in our example above that our two aircraft <u>cannot</u> land both at PLT_1 and PLT_2. One of them has to wait and its actual landing time will occur later than its preferred landing time.

Clearly, this is a different situation from our "static" case, where all of our aircraft were waiting to land and were supposed able to do so at any time given sufficient prior notice.

The main difficulty associated with the extension of the "static" algorithm into its "dynamic" version is the implicit presence of the PLT constraints: Clearly, any solution which produces a landing time for an aircraft which is <u>earlier</u> than its corresponding PLT, is <u>infeasible</u>. It is conceivable that our "dynamic" algorithm will produce such infeasible solutions, if we make an update each time an aircraft enters our system (i.e. at its corresponding SET). We suggest below several approaches to overcome this difficulty:

1) <u>Defer</u> the inclusion of an arriving aircraft into the set of problem inputs, until the solution to be produced by the "dynamic" algorithm is guaranteed (or very likely) to be feasible. This approach, of course, creates the problem of determining <u>when</u> this inclusion should be made. It should be noted at this point that an easy way to guarantee feasible

solutions is to put every arriving aircraft into a holding stack and include each aircraft as part of the problem's input only at its corresponding PLT. In this way, we transform our "dynamic" problem into a "variable reservoir" problem that can be handled easily by a modification of our "static" algorithm. It is clear that this approach will only work in situations of extreme congestion, when no other alternative exists than putting arriving aircraft into holding stacks. For other situations however, this approach will create unnecessary delays and sub-optimal solutions. Clearly, if we are to defer the inclusion of an aircraft into our input without sacrificing the efficiency of our landing operation, we should try intermediate points in time (as long as these can guarantee feasible solutions). For example, we may include each aircraft as part of our input at a prescribed (or, dependent on the system load) time instant before its PLT. Or, do the same when the aircraft reaches a similarly defined distance from the runway.

2) Another approach might be to try to solve the problem by trial and error. This would involve updates at fixed points in time (say, every 30 seconds or 1 minute). Each update will include all present aircraft into our system. If a landing time produced by the algorithm is less than the corresponding PLT, the corresponding aircraft is termporarily removed from our input and the problem is solved again, until a feasible solution is obtained.

3) An approach which would also be sensible, but for which there is no indication at this moment on whether it can be successful, is to try to directly incorporate time constraints into our D.P. formulation, in a way similar to how we had incorporated the CPS rules earlier. If this can be

done, then the question on when to include an aircraft into our input is eliminated, for we shall include it as soon as it appears and the algorithm itself will take care of the PLT constraint. It should be pointed out however, that the explicit inclusion of the time dimension into our feasibility investigation is substantially different in concept from the position-and-shift (non-dimensional) formulation of our priority constraints and is thus likely to create problems.

We now summarize our thoughts concerning the extension of our "static" ASP algorithm to its equivalent "dynamic" version. We argued earlier that in periods of extreme congestion, where the arrangement of aircraft in holding patterns cannot be avoided, our "static" algorithm will have no problem being extended to its "dynamic" version. This extension will involve updates each time an aircraft enters the holding stack. In each update, we shall keep track of the position shifts of all aircraft, so that the CPS rules are not violated.

Problems with our algorithm do appear at other than heavy traffic situations. From a philosophical point of view, these problems should perhaps be expected and it would not be fair to attribute the difficulties created by these problems to the algorithm itself. It should be kept in mind that all of our "static" algorithms were developed to solve specific sequencing problems in heavily congested situations, where all of a substantial number of airplanes would like to use the runway facilities at the same time, and no gaps in the utilization of these facilities exist. It would therefore be reasonable to anticipate that these specialized algorithms and their "dynamic" extensions would work well for situations similar to those they were created for (heavy traffic) but would most likely have some problems handling other, totally different, situations with equal efficiency (light traffic, gaps in runway utilization, "occasional" arrivals only, etc.).

Still, we feel that the results obtained so far are sufficiently interesting and promising to encourage further research on the subject along the lines suggested above, rather than tackling the problem by means of less sophisticated (enumerative or heuristic) approaches.

7.3 ASP-Multiple Runways

Despite the fact that the method we used to solve the two-runway ASP, produced an algorithm with computational efficiency of the same order of magnitude with that of the single runway algorithm, that algorithm used a "brute force" approach to solve the problem of how to partition the set of aircraft between the two runways. A possible refinement of the two-runway algorithm would therefore be to take advantage of the problem's special structure in order to develop a more sophisticated partitioning strategy. This refinement would become a necessity if more than two runways are involved, because then it would be extremely inefficient to examine all possible partitions of aircraft among all runways.

Another issue which can be examined in the two-runway case, is what happens if we include MPS constraints into our problem. We can see that this problem is substantially more complicated than the equivalent single runway case, by the fact that the order that an aircraft has among the landings of all the other aircraft (i.e 5^{th}, 10^{th}, etc.), not only depends on the relative position of this aircraft among other planes landing on the same runway, but on the landings that have taken place on the other runway as well, this latter dependence being of a rather complex nature. On the other hand, we might decide to define our MPS constraints in a different way, so that we can solve our problem by first partitioning our set of aircraft (applying the two-runway unconstrained algorithm) and subsequently solving two independent CPS problems for each of the runways. Our CPS rules will therefore be applied to each of the two sequences independently from one another. It is understood of course that this method considers the concept of priority in a rather distorted way, since it is always conceivable that an airplane which lands, say, 10^{th} on the first runway, lands actually later than an airplane which lands, say 12^{th} on the second runway.

We should mention that any attempt to tackle the multiple runway problem by direct application of dynamic programming would most likely result in an explosion in the size of the state space and should now include as state variables not only the numbers of aircraft per category for all runways, but also the time intervals until the next landings. The inclusion of continuous variables in the state space will make the problem much larger than we can reasonably handle.

Finally, it should not be forgotten that runways are used also for departures. How will departures affect our landing strategy? Clearly a runway on which both landings and departures are performed presents a different problem than the one we examined.

7.4 ASP - Usefulness of the "Steepest Descent" Heuristic for TPD Minization.

As mentioned in Appendix A, the "steepest descent" approach consists of landing waiting aircraft by descending order of the ratio P_i/t_{ii} (P_i is the number of passengers per aircraft of category i and t_{ii} is the corresponding element on the diagonal of the time separation matrix). The above heuristic seems particularly appropriate for TPD minimization (Z=2) in the single runway/unconstrained case ASP and has been seen (Appendix A) to produce the exact optimal sequence in most of the cases. Specifically, a prerequisite for the application of this heuristic is "group clustering," namely when all the aircraft of each specific category land in a single unbroken string.

While the strict application of the above heuristic without prior verification that "group clustering" will indeed occur, will in general result in sub-optimal solutions, the following facts may suggest that the usefulness of the heuristic can be substantially broader than it would appear at first glance:

1) It was observed (Chapter 4) and subsequently verified mathematically (Appendices A through D) that "group clustering" constitutes the rule rather than the exception in the cases examined.

2) In the rare cases where "group clustering" did not occur (Case 7 of Chapter 4 for example), the deterioration in TPD resulting from an arbitrary assumption of "group clustering" was not substantial: the sub-ptimal landing sequence resulting from an assumption of "group clustering" and an application of "steepest descent" in the above case exhibits a TPD

of 766,350 passenger-seconds (Fig. 4.7a), as compared to a TPD of 758,550 passenger-seconds of the exact optimal sequence (Fig. 4.6). In other words, our measure of performance in this case is seen to be rather <u>insensitive</u> to the exact optimal sequence near its optimal value.

These observations suggest that "steepest descent" may be a good heuristic to work with for TPD minimization in the single runway/unconstrained case ASP, and thus, it may conceivably be suitable for real-world computer-assisted implementation. It may also be possible to extend this heuristic to other cases as well, in particular to the CPS case and to the two-runway case. It should be realized of course that the nature of these latter problems (take for example the presence of MPS constraints in the CPS problem) may require substantial modifications in the application of this heuristic sequencing technique.

Summarizing, we express our feeling that substantial investigation should be further pursued in order to establish the usefulness of this heuristic beyond the cases examined in this report.

Also, the development of a similar heuristic for LLT minimization (Z=1) will probably require an analysis similar to the one presented in Appendices A through D for the TPD minimization objective.

7.5 ASP - Implementation Issues

It is beyond the scope of this report to get involved in detail into problems connected with the implementation of the algorithms developed here. We shall nevertheless attempt to give a flavor for the complexity of these problems.

The actual implementation of an Aircraft Sequencing algorithm is perhaps more difficult than the theoretical development of the algorithm itself. Aviation Authorities and pilots can perhaps be convinced that a sequencing strategy, <u>if implemented,</u> will result in the reduction of delays, but they must also be convinced that this strategy <u>can in fact be implemented at all</u>. This may be a very difficult thing to accomplish. Safety will be the main consideration in this respect. Clearly, a "dynamic" algorithm which drastically shifts the position of each aircraft at each update, or dictates frequent and difficult rearrangement manoeuvres in the terminal area will most likely be rejected.

Aircraft Sequencing should, of course, be computer-<u>assisted</u>, but under no circumstances should the computer (at least at its current level of evolution) be allowed full control of the whole process. The process should be flexible enough so that the participation of the human controller in decision-making is possible. The controller should, for example, be able to override the computer in cases of emergency or other unpredictable situations. Interaction with the pilots should also be allowed if necessary.

We conclude by expressing our feeling that substantial research has to be accomplished on these issues before any theoretical developments can reach the implementation stage.

REFERENCES

[AHO 74] Aho, A.V., J.E. Hopcroft, J.D. Ullman, "The design and Analysis of Computer Algorithms," Addison-Wesley, Reading Mass. (1974).

[BELL 60] Bellman, R., "Combinatorial Processes and Dynamic Programming," Proceedings of the 10th Symposium in Applied Mathematics of the American Mathematical Society (1960).

[BLUM 60] Blumstein, A., "An Analytic Investigation of Airport Capacity," Ph.D. Thesis, Cornell University (1960).

[BONN 75] Bonny, J.M., "Computer Assisted Approach Sequencing," AGARD Conference Preceedings No. 188, Cambirdge Mass. (1975).

[CHRI 75] Christofides, N., "Graph Theory: An Algorithmic Approach," Academic Press, London (1975).

[CHRI 76] Christofides, N., "Worst-Case Analysis of a New Heuristic of the Travelling Salesman Problem," appearing in Algorithms and complexity: Recent Results and New Directions (1976).

[DANT 51] Dantzig, G.B., "Maximization of a Linear Function of Variables Subject to Linear Inequalities," Chap XXI of "Activity Analysis of Production and Allocation," Cowles Commission Monograph 13, T.C. Koopmans (editor) John Wiley, New York (1951).

[DANT 53] Dantzig, G.B., "Computational Algorithm of the Revised Simplex Method," RAND Report Rm-1271, The RAND Corporation Santa Monica, Calif. (1953).

[DEAR 76] Dear, R.G., "The Dynamic Scheduling of Aircraft in the Near Terminal Area," Ph.D. Thesis, Flight Transportation Laboratory Report R76-9, M.I.T., Cambridge, Mass. (1976).

[DIJK 59] Dijkstra, E.W., "A note on two Problems in Connection with Graphs," Numerische Mathematik, 1 (1959).

[DINI 70] Dinic, E.A., "Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation," Soviet Math. Dokl., 11 (1970).

[EDMO 72] Edmonds, J., and R.M. Karp, "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems," JACM, 19 (1972).

[FLOY 62] Floyd, R.W., "Algorithm 97 - Shortest Path," CACM, 5 (1962).

[FRED 77] Frederickson, U.N., M.S. Hecht, C.E. Kim "Approximation Algorithms for some Routing Problems," Department of Computer Science, University of Maryland. (1977)

[GARE 76] Garey, M.R., D.S. Johnson, R. Sethi, "Complexity of Flow-Shop and Job-Shop Scheduling," Mathematics of Operations Research, 1 (1976).

[HELD 62] Held, M., R.M. Karp, "A Dynamic Programming Approach to Sequencing Problems," Journal of SIAM, Vol. 10, No. 1, (1962)

[HELD 70] Held, M., R.M. Karp, "The Travelling Salesman Problem and Minimum Spanning Trees," Operations Research Vol 18, No. 6 (1970).

[HELD 71] Held, M., R.M. Karp, "The Travelling Salesman Problem and Minimum Spanning Trees: Part II," Mathematical Programming 1 (1971).

[IGNA 65] Ignall, E.,L. Schrage, "Application of the Branch and Bound Technique to Some Flow-Shop Scheduling Problems," Operations Research Vol. 13, No. 3 (1965).

[KARP 75] Karp, R.M., "On the Computational Complexity of Combinatorial Optimization Problems," in Networks, 5 John Wiley & Sons, Inc. (1975).

[KARZ 74] Karzanov, A.V., "Determining the Maximal Flow in a Network by the Method of Preflows," Soviet Math. Dokl., 15 (1974).

[KLEE 70] Klee, V., G.J. Minty, "How Good is the Simplex Algorithm?" Mathematical Note No. 643, Boeing Scientific Research Laboratories (1970).

REFERENCES (CONT'D)

[KLEI 75] Kleinrock, L., "Queuing Systems, Vol. 1: Theory," Wiley Interscience (1975)

[KROL 71] Krolak, P., W. Felts, A. Marble, "A Man-Machine Approach Toward Solving the Travelling Salesman Problem," CACM, 14 (1971).

[LEWI 78] Lewis, H.R., and C.H. Papadimitriou, "Efficient Computability," Scientific American, Vol. 238, 1 (Jan. 1978).

[LIN 65] Lin, S., "Computer Solutions of the Travelling Salesman Problem," Bell System Technical Journal, 44 (1965).

[LIN 73] Lin, S., B.W. Kernighan, "An Effective Heuristic for the Travelling Salesman Problem," Operations Research Vol. 21, No. 2 (1973).

[MOON 67] Moon, J.W., Various Proofs of Cayley's Formula for Counting Trees in "A Seminar in Graph Theory," Harary, Ed., Holt, Rinehart & Winston, New York (1967).

[MURC 65] Murchland, J.D., "A New Method for Finding All Elementary Paths in a Complete Directed Graph," London School of Economics, Report LSE-TNT-22 (1965).

[PAPA 77] Papadimitriou, C.H., "The Euclidean Travelling Salesman Problem is NP-Complete," Theoretical Computer Science, 4 (1977).

[PARD XX] Pardee, R.S., "An Application of Dynamic Programming to Optimal Scheduling in a Terminal Area Air Traffic Control System," TRW Computers Company.

[PRIM 57] Prim, R.C., "Shortest Connection Networks and some Generalizations," Bell System Technical Journal, 36 (1957).

[PSAR 78] Psaraftis, H. N., "A Dynamic Programming Approach to the Dial-A-Ride Problem", Department of Civil Engineering Report R78-34 (1978)

[SHAM 75] Shamos, M.I., D. Hoey, "Closest Point Problems," Proceedings of the 6th Annual Symposium on Foundations of Computer Science (1975).

[TURI 37] Turing, A.M., "On Computable Numbers with an Application to the Entscheidungsproblem," Proceedings of London Mathematical Society, Ser. 2, Vol. 42 (1937).

[VAND 75] Van Deman, J.M., K.R. Baker, "Minimizing Mean Flowtime in the Flow Shop with No Intermediate Queues," AIIE Transactions, Vol. 6, No. 1 (1975).

[ZADE 73] Zadeh, N., "A Bad Network Problem for the Simplex Method and Other Minimum Cost Flow Algorithms", Mathematical Programming, 5, (1973).

PART IV

APPENDICES

APPENDIX A

ASP - INVESTIGATION OF GROUP "CLUSTERING"

The solution methodology developed in Chapter 4 for the ASP can be used for any values of the input, namely for any values of the elements t_{ij} of the time separation matrix and for any values P_i of the number of passengers. Nevertheless, in the examples we presented, we could not help but notice the fact that if the inputs were "reasonable" enough, the output exhibited certain very simple and well defined patterns: A first observation has been that all planes of the same category tend to be clustered together. A second observation has been that the order of preference among various categories depended, in a way which has still not been clarified, on the values of the inputs. In Chapter 4 we saw a particular example where this dependence was very delicate: Thus, while there was only one minimal difference in the inputs of Cases 8 and 9 (P_3 was 120 passengers in Case 8 versus 130 in Case 9) the optimal sequences of these two cases were strikingly different: In case 8, all planes of category 1 landed first, then all planes of category 2 and finally all planes of category 3. In case 9 we had again grouping by categories but the order now was 3, 1, 2.

A third observation has been that for other sets of "unreasonable" inputs (Case 7 versus Case 6) the optimal patterns may be entirely unpredictable.

We have put the words "reasonable" and "unreasonable" in quotes, because so far we do not know *a priori* if a set of inputs belongs to the first or to the second class and what behavior we can expect in the corresponding output. To decide upon these, we must first define what we mean by "reasonable" and second, if possible, develop a set of criteria, through which we can predict the behavior of the optimal sequence by just verifying whether the inputs satisfy these criteria or not.

Our task in this Appendix will therefore be to try to find relations among the problem inputs, so that if those are satisfied, the optimal solution can be predicted, without even having to run the dynamic program. The driving force behind this approach is the strong clues that we have perceived on the existence of a fundamental underlying structure, hidden for the time being, which controls the whole problem. The hope is that this structure, if it exists, will be simple enough, to be of practical use in the solution of the problem.

Clustering in Groups

We start by investigating the issue of the clustering of the airplanes of the same category in a single, unbroken string.

Let (A) be a segment of a given landing sequence and (B) a rearrangement of the elements of (A). Let us furthermore introduce the notation (A) > (B) if (A) is preferable to (B). This would happen if and only if the contribution of (A) to the "cost" of our objective is less than the corresponding contribution of (B).

The results which we shall present subsequently hold if our objective is to minimize Total Passenger Delay (TPD), (Z=2) and no priority constraints exist.

RESULT 1 (Fig. A.1): Interchange between two airplanes of categories i and j between two airplanes of category i:

$$(A) > (B) \iff \frac{t_{ii}}{P_i} < \frac{t_{ij} + t_{ji} - t_{ii}}{P_j} \tag{A.1}$$

Proof: Let $\Delta \equiv$ Cost (A) - Cost (B) and Q the number of passengers still waiting to land after the first aircraft of category i of (A) has landed. The subsequent numbers of passengers still waiting to land are shown in in Fig. A.1.

$$\text{Then} \quad \Delta = Q.t_{ii} + (Q-P_i)t_{ij} + (Q-P_i-P_j)t_{ji}$$

$$-Q.t_{ij} - (Q-P_j)t_{ji} - (Q-P_i-P_j)t_{ii}$$

$$\text{or} \quad \Delta = P_j.t_{ii}-P_i(t_{ij}+t_{ji}-t_{ii})$$

Obviously $(A) > (B) \iff \Delta < 0$, hence (A.1) #

We observe that (A.1) is independent of the value of Q.

RESULT 2 (Fig. A.1):

If $[t_{ij}]$ satisfies

$$t_{ii} + t_{jj} \leq t_{ij} + t_{ji} \tag{A.2}$$

and if $$\frac{P_i}{t_{ii}} > \frac{P_j}{t_{jj}}, \tag{A.3}$$

then (A) > (B).

Proof: From (A.3) and (A.2) we have:

$$\frac{t_{ii}}{P_i} < \frac{t_{jj}}{P_j} \leq \frac{t_{ij} + t_{ji} - t_{ii}}{P_j},$$ so by (A.1) it follows that (A) > (B). #

Some notes on notation:

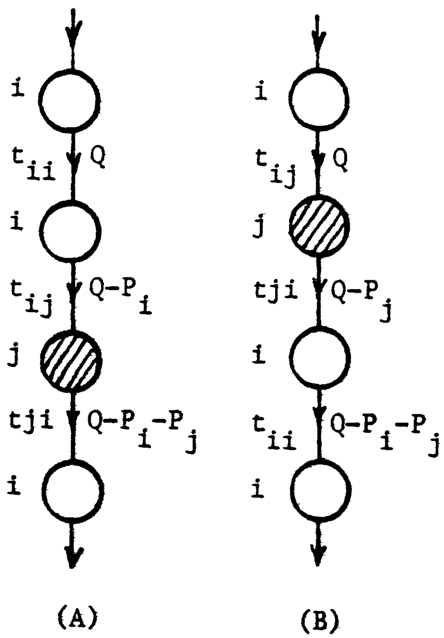


Fig. A.1.

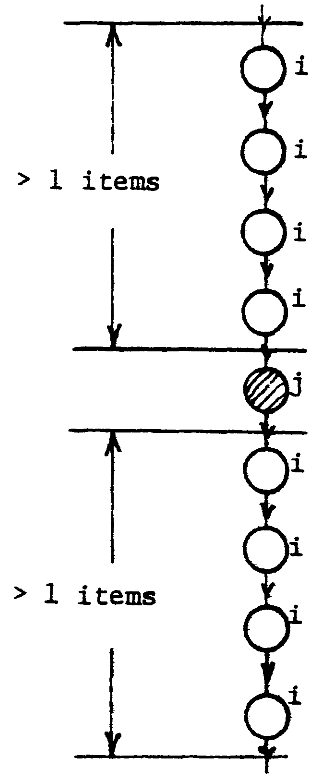


Fig. A.2.

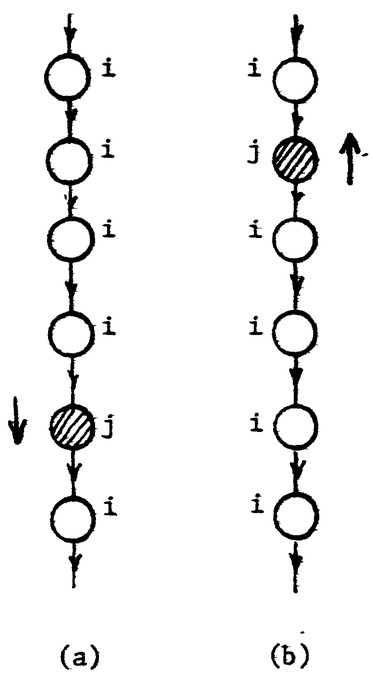


Fig. A.3.

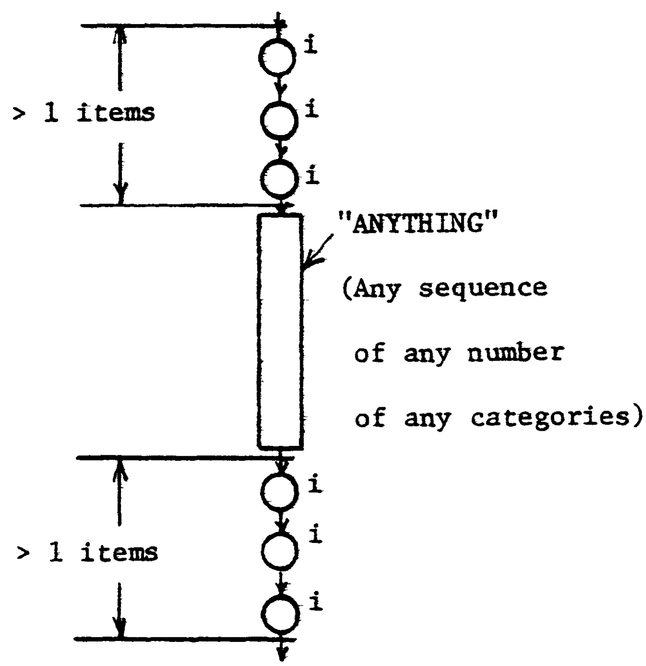


Fig. A.4.

a) From now on, we shall characterize a the time separation matrix $[t_{ij}]$ that satisfies (A.2) as "(i,j)-reasonable." In Appendix B we show how the elements of $[t_{ij}]$ are derived for the ASP and in Appendix C we prove mathematically that for real-world data and for all pairs (i,j), a time separation matrix $[t_{ij}]$ for the ASP is always "(i,j)-reasonable."

b) We shall furthermore define that category i is "denser" than category j if and only if (A.3) holds. Note that the ratios in (A.3) represent the rate of passengers landed per unit time if each category is followed by itself. This concept is not to be confused with the one of the landing velocity of a category. The notion of denseness, as we have defined it, will turn out to be very important in our subsequent investigation.

RESULT 3 (Fig. A.2):

A sequence of the form of Fig. A.2 cannot be optimal, unless:

$$\frac{t_{ii}}{P_i} = \frac{t_{ij} + t_{ji} - t_{ii}}{P_j} \tag{A.4}$$

Proof a) suppose first that $\frac{t_{ii}}{P_i} < \frac{t_{ij} + t_{ji} - t_{ii}}{P_j}$. Then from (A.1) we see that we can gain by moving category j downstream (in the sequence) by one step. In fact by repeatedly applying (A.1) we can move j as far down as one position before the last item of category i.

b) If now $\frac{t_{ii}}{P_i} > \frac{t_{ij} + t_{ji} - t_{ii}}{P_j}$ then by similar arguments we conclude that we can gain by moving j upstream as far as we can, namely as far up as one position after the first item of category i.

These two cases are depicted in Figure A.3

c) If $\frac{t_{ii}}{p_i} = \frac{t_{ij} + t_{ji} - t_{ii}}{p_j}$ then we have equilibrium but we can see that this equilibrium is neutral, namely we can move j up or down without any effect on the value of the solution #.

Observe that at this point we cannot say anything about whether j will actually stop at one of the positions shown in (a) or (b) or whether it will cross the border of the single time of category i and move further up or down. This will depend on "what is beyond" this point. We will investigate this matter later.

A generalization of Result 3 is the following:

RESULT 4 (Fig. A.4):

A sequence of the form of Fig. A.4 is either non-optimal or neutral.

Proof: The argument here is that "ANYTHING" (see Fig. A.4) can be regarded as a new single category (with uniquely defined time and passenger characteristics), so that our result is a natural consequence of (A.4). We shall examine how we can transform any sequence to an "equivalent" single category in Appendix D. #

RESULT 5

If the time separation matrix $[t_{ij}]$ is (i,j)-reasonable then a unique item of j in the middle of a string of items of category i should go downstream if i is "denser" than j.

Proof: Immediate from Results 2 and 3 #.

Important observation: Result 5 does not imply that j should necessarily go upstream if j is "denser" than i, because Result 2 is only a sufficient and not a necessary condition. Observe also that Result

5 is a clue that the "steepest-descent" criterion* applies at least in several cases. We will encounter the same criterion many times later. We look now at a slightly different configuration:

RESULT 6 (Fig. A.5)

Concerns whether two categories i and j should be seperated (A) or overlap (B):

$$(A) > (B) \iff Q(t_{ii}+t_{jj}-t_{ij}-t_{ji}) + P_j(t_{ji}+t_{ij}-t_{jj})-P_i t_{jj} < 0 \tag{A.5}$$

Proof: Similar to the proof of (A.1) # . Observe that (A.5) depends on the value of Q and this makes it not as handy as (A.1), for Q will in general depend on the upstream sequence.

RESULT 7 (Fig. A.5):

If $[t_{ij}]$ is "(i,j)-reasonable" and i is "denser" than j then (A) > (B)

Proof: First of all, since $[t_{ij}]$ is "(i,j)-reasonable", $t_{ii}+t_{jj}-t_{ij}-t_{ji} \leq 0$. Second, noticing that the passenger outflow from the last node j is at least P_j, we conclude that $Q \geq P_i + 2P_j$. Taking into account these two inequalities and (A.5) we conclude that to have (A) > (B), it is sufficient to have $(P_i+2P_j)(t_{ii}+t_{jj}-t_{ij}-t_{ji})+P_j(t_{ji}+t_{ij}-t_{jj})-P_i \cdot t_{jj} < 0$ which, after some manipulations, is equivalent to:

*By "steepest-descent" we mean the landing discipline that lands "denser" categories first. It should be noted that this strategy is similar to a well-established result in queuing theory [KLEI 75] where it is shown that to minimize the average cost to users in a queuing system, one should assign priorities by descending order of cost-per-unit-time ratios.

$$\frac{t_{ii}+t_{ji}}{P_i+P_j} < \frac{t_{ij}+t_{ji}-t_{ii}}{P_j} \tag{A.6}$$

We will show that the above inequality <u>always</u> holds if i is "denser" than j and $[t_{ij}]$ is "(i,j)-reasonable":

First, we show that $\frac{t_{ii}+t_{jj}}{P_i+P_j} < \frac{t_{jj}}{P_j}$ (A.7)

In fact, we check this, since (A.7) is equivalent to:

$$P_j.t_{ii} + P_jt_{jj} < P_it_{jj} + P_jt_{jj} \quad \text{or to } \frac{P_j}{t_{jj}} < \frac{P_i}{t_{ii}}$$

which is true since i is "<u>denser</u>" than j.

Second we show that:

$$\frac{t_{ji}}{P_j} < \frac{t_{ij}+t_{ji}-t_{ii}}{P_j} \tag{A.8}$$

which is obvious if $[t_{ij}]$ is <u>(i,j)-reasonable"</u>.

Then (A.6) follows directly from (A.7) and (A.8). But (A.6) is a <u>sufficient</u> condition for (A) > (B) and this concludes our proof. #.

This is an important result for it may help resolve an issue mentioned earlier (see observation following the proof of Result 3), namely: Given that i is "denser" than j, can we say anything about the sequence of Fig. A.6?

At this point, we can say that while Result 3 states that the first of the j's is <u>correctly</u> at its "downmost" position, Result 3 cannot resolve the question of whether it should go <u>further</u> <u>downstream</u>, getting in fact interchanged with the last of i's so that finally the two categories are fully separated and no overlap exists. Result 7 provides a "yes" answer to this latter question.

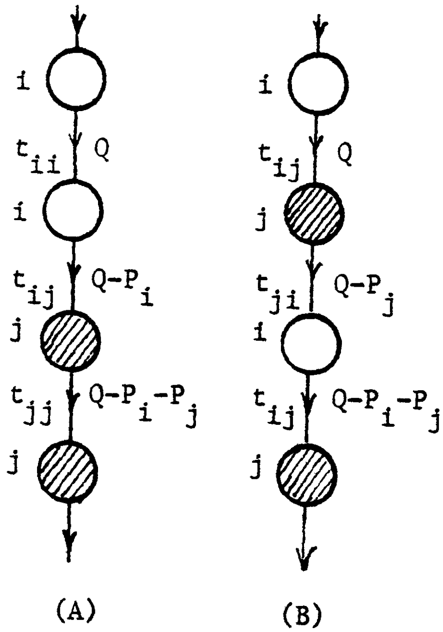


Fig. A.5.

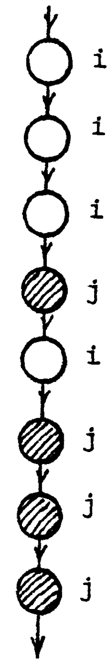


Fig. A.6.

Fig. A.7.

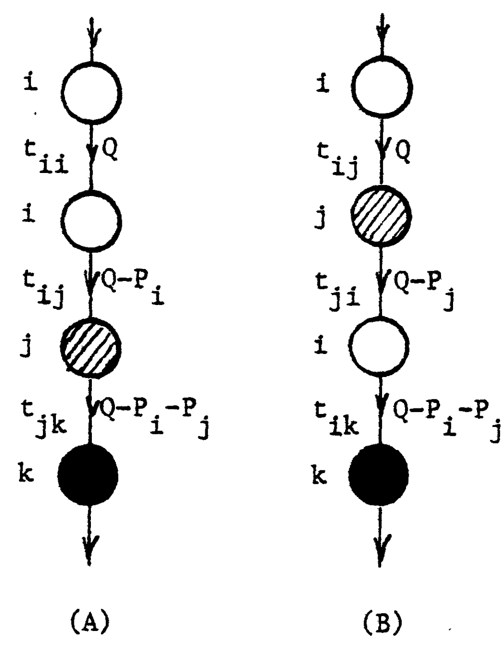


Fig. A.8.

Still, Result 7 is not powerful enough to help us decide what happens with the sequence of Fig. A.7 (even if it is given that i is "denser" than j).

This is the issue we shall investigate next.

RESULT 8 (Fig. A.8)

i versus j given the presence of a third category k at the bottom:

$$(A) > (B) \Leftrightarrow Q[t_{ii}+t_{jk}-t_{ji}-t_{ik}] + P_i[-t_{ij}-t_{jk}+t_{ik}] + P_j[-t_{jk}+t_{ji}+t_{ik}] < 0 \tag{A.9}$$

Proof: Similar to the proof of (A.1) #. Before moving to the next result, we give the following definition:

> A time separation matrix $[t_{ij}]$ is "(i,j,k)-reasonable" if and only if $t_{ii}+t_{jk} \leq t_{ji}+t_{ik}$

Observe that this is a generalization of the concept of "(i,j)-reasonableness," in the sense that if a matrix is "(i,j,j)-reasonable," then this is exactly the same thing as being "(i,j)-reasonable." We repeat that we study thoroughly "(i,j,k)-reasonableness" in Appendix C where we show that for the ASP, $[t_{ij}]$ is "(i,j,k)-reasonable under all (i,j,k) configurations, except when only j is a "heavy" jet and i has a higher landing velocity than k.

RESULT 9 (Fig. A.8):

If $[t_{ij}]$ is "(i,j,L)-reasonable" for both L=k and j and if i is "denser" than j then (A) > (B).

Proof: The proof is similar to the proof of Result 7: We take into account both $t_{ii}+t_{jk}-t_{ji}-t_{ik} \leq 0$ ((i,j,k)-reasonableness) and the fact that $Q > P_i+P_j$. So we obtain a sufficient condition for (A) > (B), the following:

$$(P_i+P_j)(t_{ii}-t_{jk}-t_{ji}-t_{ik})+P_i(-t_{ij}-t_{jk}+t_{ik})+P_j(-t_{jk}+t_{ji}+t_{ik}) < 0$$

or, equivalently, after some manipulations:

$$\frac{t_{ii}}{P_i} < \frac{t_{ij}+t_{ji}-t_{ii}}{P_j}$$

It is easy now to see that if $[t_{ij}]$ is "(i,j)-reasonable" and if i is "denser" than j then the above is true #

Before commenting on the consequences of the above results, we state below two more equivalent results which hold if category k is at the top.

RESULT 10 (Fig. A.9):

i versus j given the presence of a third category k at the top:

$$(A) > (B) \iff Q[t_{kj}-t_{ki}+t_{ii}-t_{ij}]+P_i[-t_{ii}+t_{ij}+t_{ji}]-P_jt_{ii} < 0 \qquad (A.10)$$

Proof: Similar to the proof of (A.1) #.

RESULT 11: (Fig. A.9)

If: a) $[t_{ij}]$ is not (i,k,j)-reasonable

b) $[t_{ij}]$ is (i,j,k)-reasonable

c) i is denser than j, then (B) > (A)

Proof: Let Δ= Cost(A) - Cost(B). Then, since $t_{kj}-t_{ki}+t_{ii}-t_{ij} > 0$ because of (a) and since $Q > P_i$, we have:

$$\Delta > P_i[t_{kj}-t_{ki}+t_{ji}] - P_j t_{ii}$$

But because of (b) $t_{kj}-t_{ki}+t_{ji} > t_{jj}$, so

$$\Delta > P_i t_{jj} - P_j t_{ii}$$

But from (c), this last quantity is > 0 so $\Delta>0$#.

<u>RESULT 12 (Fig. A.10)</u>

i versus j at the end of the queue given the presence of category k:

$$(A) > (B) <=> \frac{t_{ij}+t_{ki}-t_{kj}}{P_i} < \frac{t_{kj}+t_{ji}-t_{ki}}{P_j} \qquad (A.11)$$

<u>Proof:</u> Similar to the proof of (A.1) #.

<u>RESULT 13</u> (End of the queue as above but with k≡i)

Then $$(A) > (B) <=> \frac{t_{ii}}{P_i} < \frac{t_{ij}+t_{ji}-t_{ii}}{P_j} \qquad (A.12)$$

<u>Proof</u>: From (A.11), setting k≡i #.

<u>RESULT 14</u>

If $[t_{ij}]$ is (i,j)-reasonable and i is denser than j then (A) > (B) in (A.12).

<u>Proof</u>: Because of (i,j)-reasonableness we have $\frac{t_{ij}+t_{ji}-t_{ii}}{P_j} > \frac{t_{jj}}{P_j}$.

Also $\frac{t_{jj}}{P_j} > \frac{t_{ii}}{P_i}$ then (A.12) holds #.

We next present three results concerning the comparison of <u>groups</u> of categories.

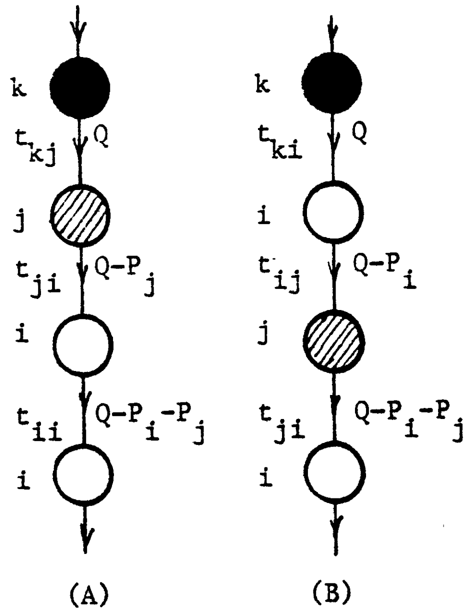


Fig. A.9.

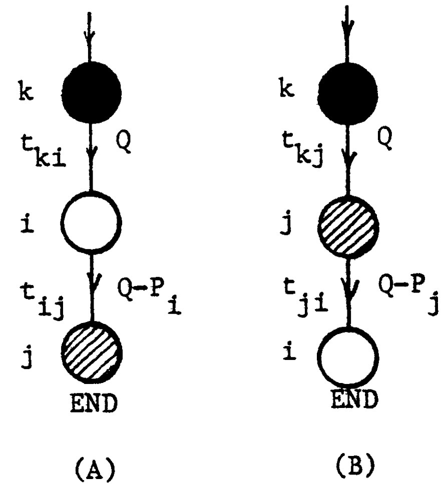


Fig. A.10.

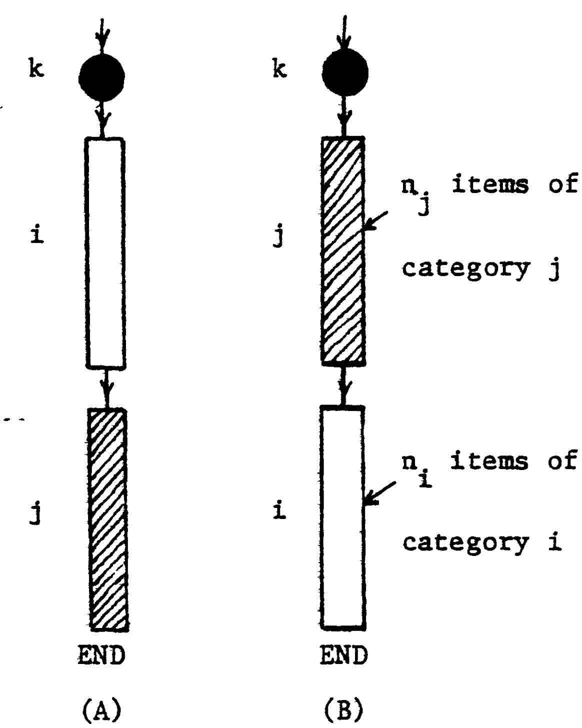


Fig. A.11.

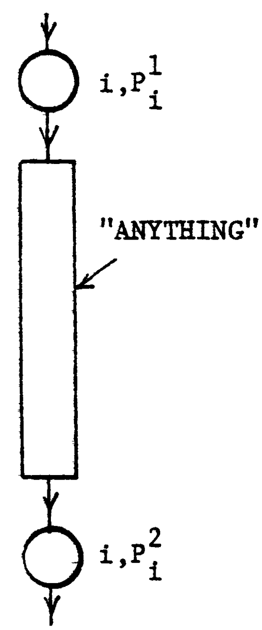


Fig. A.12.

RESULT 15 (Fig. A.11)

$$(A) > (B) \iff n_i n_j (P_j t_{ii} - P_i t_{jj}) + (n_i P_i + n_j P_j)(t_{ki} - t_{kj}) +$$

$$+ n_i P_i (t_{jj} - t_{ji}) - n_j P_j (t_{ii} - t_{ij}) < 0 \qquad (A.13)$$

Proof: Straightforward, taking into account the equivalence transformations of Appendix D. #

RESULT 16: (Fig. A.11)

If n_i, n_j are large, then (A) > (B) if and only if category i is denser than category j.

Proof: If n_i, $n_j \to +\infty$, then dominant term in (A.13) is the first one #.

RESULT 17:

If $[t_{ij}]$ is "(i,j,k)-reasonable", and if the relation [(A) > (B) for (n_i,n_j)] is true, the relation [(A) > (B) for (n_i+1, n_j)] is also true.

Proof: The first relation yields that

$$n_i [n_j P_j t_{ii} + P_i (t_{ki} - t_{kj} - t_{ji} + t_{jj}) - n_j P_i t_{jj}] < n_j P_j [t_{ii} + t_{kj} - t_{ki} - t_{ij}]$$

Since $[t_{ij}]$ is "(i,j,k)-reasonable", the right-hand side is < 0. Therefore the left-hand side is also < 0. A fortiori, the value of the left-hand side will decrease if instead of n_i we multiply the bracket by (n_i+1). But this is equivalent to saying that (A) > (B) for (n_i+1, n_j) #.

One can generalize the previous result by stating that if we know that (A) > (B) for some values of (n_i,n_j) then we also know that (A)>(B) for (n_i',n_j) where $n_i'>n_i$. This result may prove itself useful when we know how groups of a given size behave and we want to know what happens to groups of different size

Our last result concerns what happens if we have a variable number of passengers for aircraft of the same category. We shall examine other issues connected to this case in Appendix D. For the moment our fundamental result is the following:

RESULT 18 (Fig. A.12):

Variable number of passengers per category:

If for any two aircraft of category i, $P_i^1 < P_i^2$, then we can always obtain an improvement by interchanging them.

Proof: An interchange argument will suffice. #.

An important consequence of the above result is that as far as the internal ordering of the aircraft of a certain category is concerned, one cannot do better than by ordering them by descending order of number of passengers. A formal proof of this result will be presented in Appendix D, in the case where all aircraft of this category are clustered in a single group. Nevertheless, our observation holds also for any positions that these aircraft may have in the sequence. It should however be pointed out that this observation does not give an answer to the question of how various categories interact with one another. To answer that question in all its various aspects, one must get involved in an investigation similar to the one that we performed so far, while also taking into account the variability of the numbers of passengers. It is felt that such a task is beyond the scope of this dissertation. However, we contend ourselves to state without proof that Result 4 (Fig. A.4) which we developed for a constant number of passengers, holds also for the variable number of passengers case, provided we have arranged the aircraft of category i by descending order of number of passengers.

Summary - Conclusions

This Appendix examined the ASP from a different point of view than the one used in Part II. The main goal was to derive quantitative relations between the problem inputs so that the optimal sequence exhibits category clustering. In this respect, the following concepts were seen to be important:

1) The concept of "reasonableness" of the time separation matrix was seen to appear in several configurations as a condition for category clustering. In Appendix C it is shown that the time separation matrix derived especially for the ASP (Appendix B) is always "(i,j)-reasonable" and in all but one cases, "(i,j,k)-reasonable" as well. In the case where the matrix is not "(i,j,k)-reasonable" one would expect the possibility that category clustering is not an optimal configuration. In Appendix C we show that Case 7 of Chapter 4 (where the categories do not cluster entirely in groups) can be explained by "reasonableness" considerations.

2) The concept of category "density" was seen to govern which category receives priority in sequencing. In this respect it was seen that "denser" categories (i.e. categories which deliver more passengers per unit time if clustered in a group) tend to be preferred and be assigned a higher priority than other categories. This is in accordance with the "steepest descent" criterion (land "denser" categories first) that one also encounters as a similar priority rule in queueing theory. This criterion was shown to be asymptotically valid, but holds in many other configurations as well.

3) We have also looked at, although not examined in detail, the case of a variable number of passengers per category. In this respect, we saw that it always pays to order the corresponding airplanes by descending order of passengers. In Appendix D we show that we can substitute any such group of airplanes by an "equivalent" group with constant number of passengers.

APPENDIX B

ASP - DERIVATION OF THE TIME SEPARATION MATRIX

The purpose of this Appendix is to present how the elements of the time separation matrix $[t_{ij}]$ of the Aircraft Sequencing Problem are derived from the characteristics of the aircraft of our problem.

Without loss of generality, we decide at this point that we shall index the various categories by descending order of landing velocity. Namely, for two categories i and j, $i < j$ implies $v_i > v_j$ and vice versa (v_i, v_j are the landing velocities of i,j). The landing velocity of a particular category is a very important parameter in itself, because:

1) It is mainly this parameter which determines how soon a particular airplane will land after the landing of another one.

2) The landing velocity is far from being the same for all airplanes. Typical values of the landing velocities of various categories were given in Chapter 4.

The landing velocity is not the only parameter which is important. Another is h_{ij}, the minimum horizontal separation of an aircraft of category j following an aircraft of category i. Wake vortex considerations dictate that h_{ij} be different for different combinations of i and j. The main distinction between aircraft in this case concerns whether we have a "heavy" jet or not. Thus, recent FAA regulations state that h_{ij} must have the following values:

1) 3 nautical miles if the preceding aircraft i is not a heavy jet.

2) 6 nautical miles if the preceding aircraft i is a heavy jet and the following aircraft j is not.

3) 4 nautical miles if both i and j are heavy jets.

Before deriving a formula for t_{ij}, let us define as F the <u>horizontal length of the common final approach</u>, in nautical miles. F is of the order of 6-8 nautical miles and is constant for all airplanes in the system.

We are now in a position to develop a formula for t_{ij}. In [BLUM 60] Blumstein suggested the following formula for t_{ij}.

1) "Overtaking" case: The "following" airplane j has a landing velocity greater than the landing velocity of the "leading" airplane i. ($v_i < v_j \iff i > j$.)

Then it is clear that the distance gap between i and j becomes smaller and smaller with time, reaching its minimum permissible value h_{ij} at the instant T_o, at which airplane i lands. The time interval between that specific instant and the instant airplane j lands is therefore $t_{ij} = \frac{h_{ij}}{v_j}$, as can be seen from Fig. B.1.

2) "Opening" case: In that case the "following" airplane j has a landing velocity smaller than the landing velocity of the "leading" airplane i ($v_i > v_j \iff i < j$).

Then it is clear that the distance gap between i and j gets larger and larger with time, having reached its minimum permissible value h_{ij} at the instant T_o at which the "leading" airplane i begins its final approach of length F. The time interval t_{ij} between the two successive landings is therefore the difference between $\frac{h_{ij}+F}{v_j}$ (Time interval from T_o to the landing of airplane j) and $\frac{F}{v_i}$ (Time interval from T_o to the landing of airplane i). Thus $t_{ij} = \frac{h_{ij} + F}{v_j} - \frac{F}{v_i}$ as can be seen from

Fig. B.2.

Remark: If i=j the two cases coincide so $t_{ii}=\frac{h_{ii}}{v_i}$

Summary: $i \geq j$ $(v_i \leq v_j)$: $t_{ij} = \frac{h_{ij}}{v_j}$ (B.1)

$i < j$ $(v_i > v_j)$: $t_{ij} = \frac{h_{ij}+F}{v_j} - \frac{F}{v_i}$ (B.2)

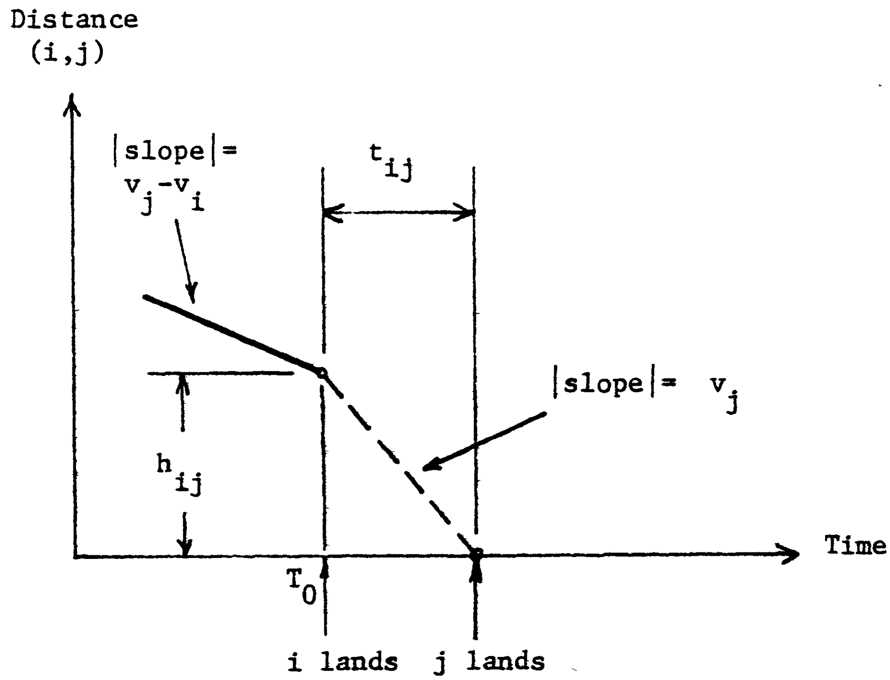


Fig. B.1 : "Overtaking" case ($v_j > v_i$).

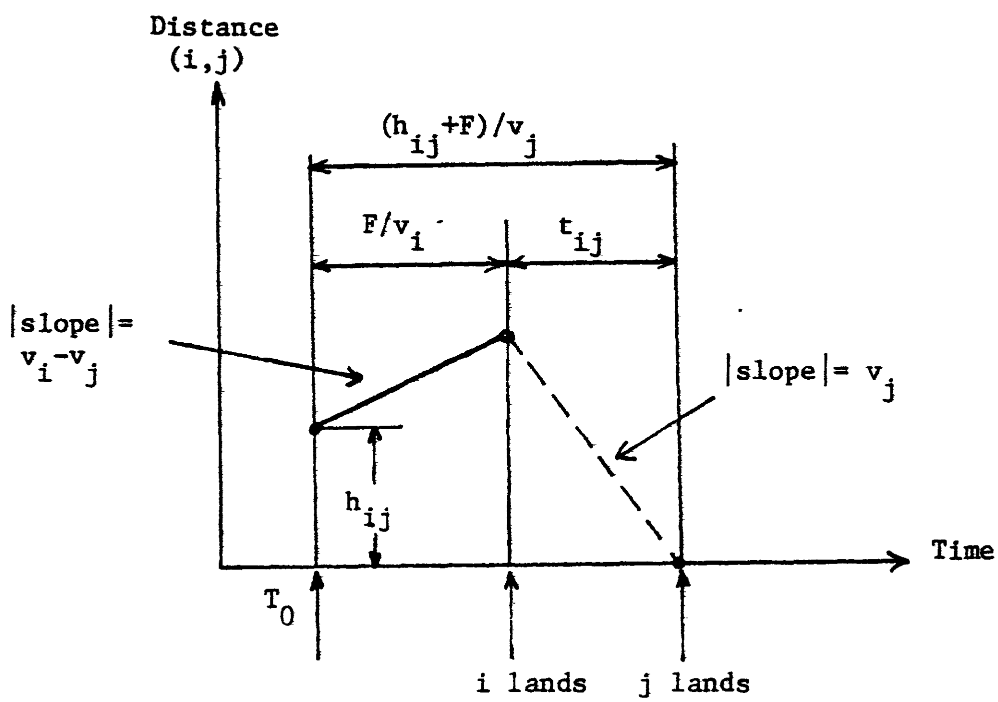


Fig. B.2 : "Opening" case ($v_i > v_j$).

APPENDIX C

ASP - ON THE "REASONABLENESS" OF THE TIME SEPARATION MATRIX

We saw in Appendix A that in certain cases where the time separation matrix $[t_{ij}]$ had some special characteristics, we were able to take advantage of these and reach some conclusions about the optimal pattern of the Aircraft Sequencing Problem which facilitated the solution. These special characteristics are the "(i,j)-reasonableness" and, more generally, the "(i,j,k)-reasonableness" of the matrix, as they were defined there. This Appendix will investigate under what conditions this matrix is (i,j) or (i,j,k)-reasonable.

We repeat the definition of <u>(i,j,k)-reasonableness</u>:

> A time separation matrix $[t_{ij}]$ is <u>(i,j,k)-reasonable</u> if and only if for the specific values of (i,j,k), we have
>
> $E \equiv t_{ii}-t_{ji}+t_{jk}-t_{ik} \leq 0.$

Also:

> A time separation matrix $[t_{ij}]$ is <u>(i,j)-reasonable</u> if and only if it is (i,j,j)-reasonable.

Since the concept of (i,j,k)-reasonableness is more general, we shall examine it directly.

In Appendix B we saw that the minimum horizontal separation h_{ij} is by rule a constant of 3 nautical miles, <u>except</u> when i is a heavy jet, when it is equal to 4 nautical miles if j is also a heavy jet and 6 nautical miles otherwise.

Being careful about this fact and indexing the heavy jet category with index = 1 and the other categories by descending order of landing velocity, we form the following 15 mutually exclusive and collectively exhaustive cases:

1) $i=j$
2) $k=i\neq j$
3) $i>j>k=1$
4) $i>j\geq k$ 1
5) $i>j=k=1$
6) $k>i>j>1$
7) $k>i>j=1$
8) $i>k>j>1$
9) $i>k>j=1$
10) $j>i>k>1$
11) $j>i>k=1$
12) $k>j>i>1$
13) $k>j>i=1$
14) $k\geq j>i>1$
15) $j\geq k>i=1$

We shall present here <u>only one</u> of the 15 cases, for demonstration purposes, Case 5:

Since $i>j=k=1$, this means that:

a) $v_i<v_j=v_k$,

b) $h_{ii} = h_{ik}=3$,

c) $h_{ji} = 6$

d) $h_{jk} = 4$ nautical miles.

A straightforward application of (B.1), (B.2) (Appendix B) yields.

$$t_{ii} = \frac{3}{v_i}$$

$$t_{ji} = \frac{6+F}{v_i} - \frac{F}{v_j}$$

$$t_{jk} = \frac{4}{v_k}$$

$$t_{ik} = \frac{3}{v_k}$$

Then $$E= \frac{3}{v_i} - \frac{6+F}{v_i} + \frac{F}{v_j} + \frac{4}{v_k} - \frac{3}{v_k} = \frac{F+1}{v_j} - \frac{F+3}{v_i} < \frac{F+1}{v_i} - \frac{F+3}{v_i} = -\frac{2}{v_i} < 0.$$

So $E<0$ in this case, which means that our matrix is (i,j,k)-reasonable.

After tedious but straightforward similar calculations for the remaining 14 cases, we have come to the following conclusions:

1) The time separation matrix $[t_{ij}]$ of the ASP is always (i,j)-reasonable.

2) The time separation matrix $[t_{ij}]$ of the ASP is (i,j,k)-reasonable in all cases except the case where $k > i > j = 1$ (Case 7). This corresponds to the case where j is a heavy jet and the landing velocity of i is higher than that of k.

What complications can Case 7 create?

To answer that question we examine a real-world example. Suppose we have 3 categories of aircraft:

1) B747's with a landing velocity of v_1=150 knots and with number of passengers P_1=300.

2) B707's with v_2=135 knots and P_2 = 150.

3) DC-9's with v_3=120 knots and P_3=100.

We furthermore assume that the length of the common final approach is F = 8 nautical miles.

A straightforward application of (B.1), (B.2), incorporating also the FAA regulations on minimum horizontal separation, yields the following time separation matrix (in seconds):

$$[t_{ij}] = \begin{bmatrix} 96 & 181 & 228 \\ 72 & 80 & 117 \\ 72 & 80 & 90 \end{bmatrix}$$

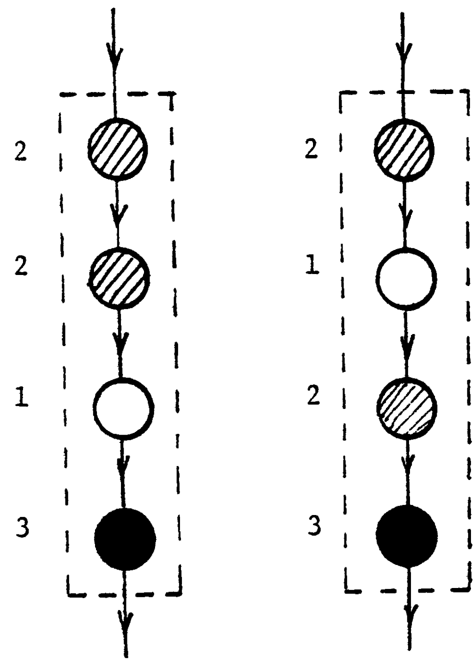
One can recognize this matrix as one of those we already have used in several of our examples.

As it was suggested earlier, this matrix is (i,j,k)-reasonable for all (i,j,k) except for the combination (2,1,3). In fact, for this combination, E=80-181+228-117 = +10 > 0.

A consequence of this fact is that we cannot apply Result 9 of Chapter 7, which states that segment (A) is preferable to segment (B) (Fig. C.1) if $[t_{ij}]$ is (2,1,3)-reasonable.

So segment (B) may, under certain circumstances, be preferable to segment (A). And we may at this point recall a case we already have presented where exactly this happens. It is Case 7 in Chapter 4 where this segment appears. (Fig. 4.6). Referring to this Case, we have shown in Chapter 4 that it is impossible to improve upon the corresponding sequence. (Fig. 4.7a, 4.7b).

Our investigation on the "reasonableness" of $[t_{ij}]$ has shed some light on this whole issue.



(A) (B)

Fig. C.1.

APPENDIX D

EQUIVALENCE TRANSFORMATIONS IN GROUP "CLUSTERING"

In Appendix A we saw under what conditions we should expect airplanes belonging to a certain category to be clustered together in a separate group. We also hinted that it can be shown that any sequence of any airplanes can be considered as a single item, provided certain "equivalence" criteria are met. (Result 4 of Appendix A.)

It may be possible that we can take advantage of this "grouping" in our D.P. formulation: In fact, if we can show that the optimal sequence corresponding to $(k_1,k_2,\ldots,k_N)$ airplanes consists solely of N groups, each containing the k_i items belonging to each particular category i, then we essentially have only N items (instead of $\sum_{i=1}^{N} k_i$) which we have to sequence, namely the N "blocks" of all categories clustered together. So we may be able to use the same D.P. approach for the "clustered" sets and bring the computational effort from $\prod_{i=1}^{N} (k_i+1)$ down to 2^N iterations*.

At this moment, however, there are some points which are still obscure:

(1) What will be the "equivalent" time separation matrix?

(2) What will be the "equivalent" numbers of passengers?

(3) How shall we take into account the zeroth landing category?

The purpose of this Appendix is to answer the above questions and

*Of course this problem may be so simple, so that we can solve it by complete enumeration.

to describe the implementation of the "grouping" procedure for the D.P. formulation. The simplest way to state our problem is the following (Fig. D.1):

Given a group of n items of category i, with constant number of passengers P_i, and two (not necessarily distinct from each other and from i) categories k and j, is there a way to replace this group by a single category I, so that the two segments of Fig. D.1 are completely equivalent?

Before attempting to analyze the problem, we have to state explicitly what we mean by complete equivalence. By this we mean that 3 conditions should hold simultaneously:

(1) Both segments deliver the same number of passengers

(2) Both segments have the same contribution in the Last Landing Time of our sequence.

(3) Both segments have the same contribution in the Total Passenger Delay of our sequence.

First of all it is not obvious at all whether actually there exists a combination of parameters that satisfies (1), (2) and (3) simultaneously. Referring to Fig. D.1 however, we can see that the unknowns of the single category I are also three: t_{kI}, P_I and t_{Ij}. This is a clue that perhaps we can find a combination of them that satisfies (1), (2), and (3) simultaneously. We proceed as follows:

For (1) to be satisfied, it is clear that category I should have a number of passengers equal to the total number of passengers within the group. So

$$P_I = nP_i \tag{D.1}$$

For (2) to be satisfied, we have to have

$$t_{ki} + (n-1)t_{ii} + t_{ij} = t_{kI} + t_{Ij} \tag{D.2}$$

For (3) to be satisfied, we have to have:

$$Qt_{ki}+[(Q-P_i)+(Q-2P_i)+\ldots+(Q-(n-1)P_i)]t_{ii}+(Q-nP_i)t_{ij}=$$
$$= Qt_{kI}+(Q-P_I)t_{ij}$$

Rewriting we have:

$$Q[t_{ki}+(n-1)t_{ii}+t_{ij}] - \frac{n(n-1)}{2} t_{ii}P_i - nP_i t_{ij} =$$
$$= Q[t_{kI}+t_{Ij}]-P_I t_{Ij}$$

We observe now that the terms containing Q cancel out because of (D.2). Also we take (D.1) into account so that:

$$\frac{n(n-1)}{2} t_{ii}P_i + nP_i t_{ij} = nP_i t_{Ij}$$

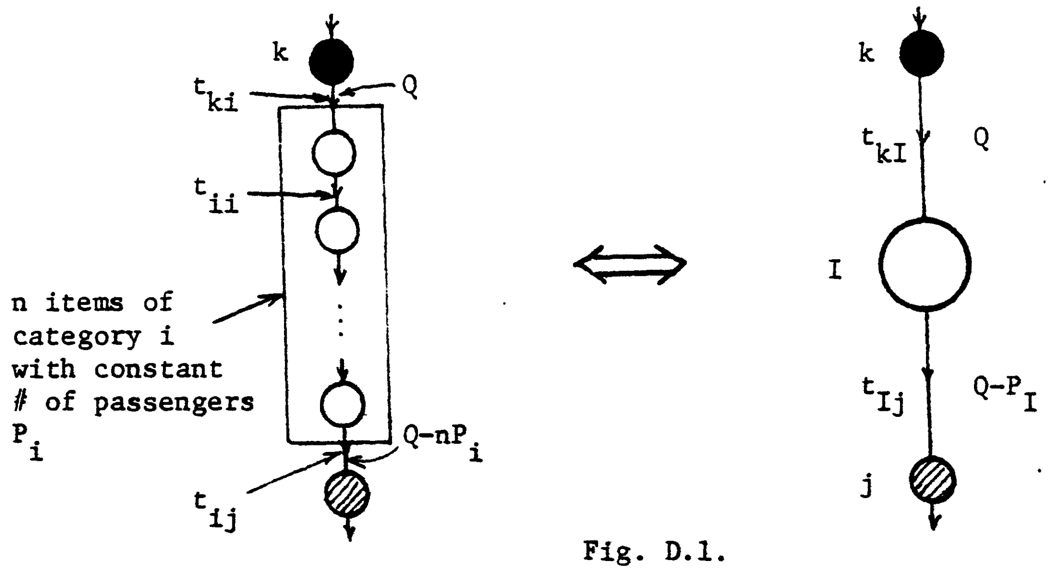
or, finally

$$t_{Ij} = t_{ij} + \frac{(n-1)}{2} t_{ii} \tag{D.3}$$

From (D.2) we find also that:

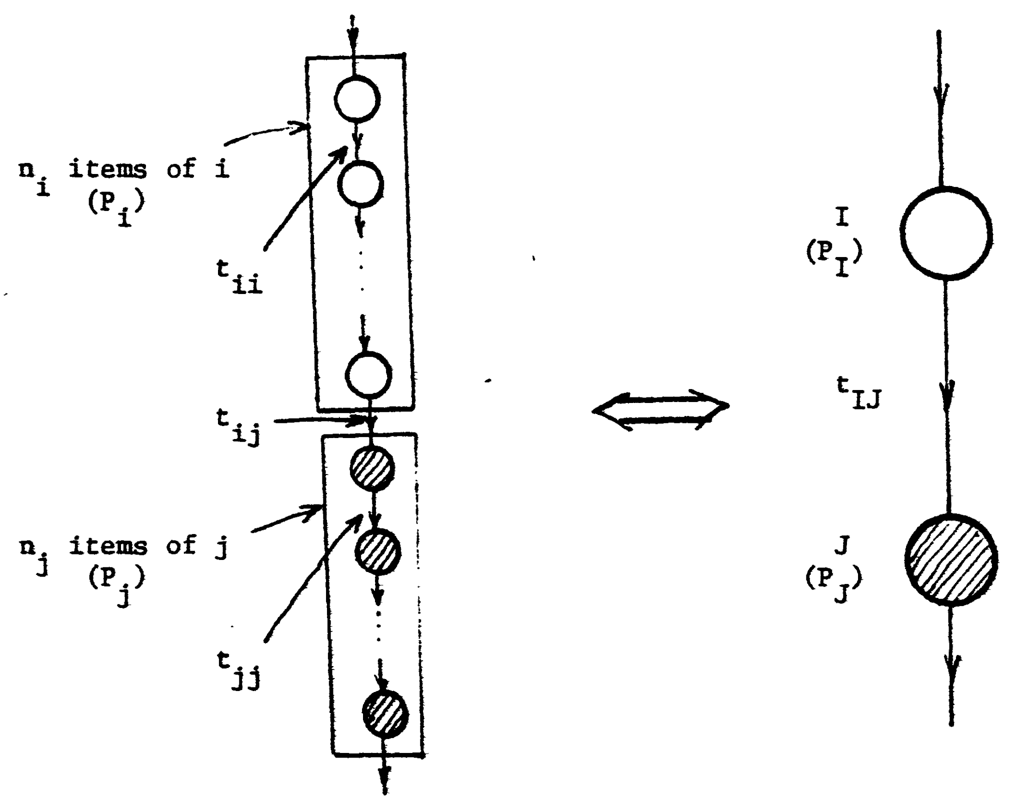
$$t_{kI} = t_{ki} + \frac{(n-1)}{2} t_{ii} \tag{D.4}$$

So (D.1), (D.3), (D.4) constitute the solution to our problem #



k
t_{ki}
Q
t_{ii}
n items of category i with constant # of passengers P_i
Q-nP_i
t_{ij}
k
t_{kI}
Q
I
t_{Ij}
Q-P_I
j

Fig. D.1.



n_i items of i (P_i)
t_{ii}
t_{ij}
n_j items of j (P_j)
t_{jj}
I (P_I)
t_{IJ}
J (P_J)

Fig. D.2.

It is easy to see now the truth of the equivalence of Fig. D.2, where:

$$P_I = n_i P_i$$

$$P_J = n_j P_j$$

$$t_{IJ} = t_{ij} + \frac{(n_i-1)}{2} t_{ii} + \frac{(n_j-1)}{2} t_{jj}$$

The equivalence of Fig. D.3 is also true, with:

$$P_I = nP_i$$

$$t_{I,END} = \frac{(n-1)}{2} t_{ii}$$

Note the dummy node "END" after category I. Note also that there is no passenger flow from I to "END" since they all have left the system in node I. Node "END" is present, only to account for the second half of the total delay $(n-1)t_{ii}$ of the group, since the first half was accounted for before node I.

We shall state without proof several generalizations of the previous results:

1) We show how to transform a string which may include any number of various different categories into a single category I. (Fig. D.4.) It is "straightforward" to show that:

$$P_I = \sum_{i=1}^{n} P_i$$

$$t_{kI} = t_{k1} + \frac{1}{\sum_{i=1}^{n} P_1} \sum_{i=1}^{n-1} t_{i,i+1} \left(\sum_{m=i+1}^{n} P_m \right)$$

$$t_{Ij} = t_{nj} + \frac{1}{\sum_{i=1}^{n} P_1} \sum_{i=1}^{n-1} t_{i,i+1} \left(\sum_{m=1}^{i} P_m \right)$$

2) We show how to transform a string which includes n planes of category i, but where the number of passengers <u>is not constant</u>, but varies, $P_i^1, P_i^2, \ldots, P_i^n$, (Fig. D.5) into a <u>single</u> category I, or, equivalently, into a string of n planes of category i with a <u>constant</u> number of passengers Π_i.

Before stating the equivalence formulae, it will be useful to futher define:

$$\delta_i^m \equiv P_i^m - \Pi_i \quad (m=1,\ldots,n)$$

The equivalence formulae are

$$P_I = \sum_{m=1}^{n} P_i^m$$

$$\Pi_i = \frac{1}{n} P_I$$

$$t_{kI} = t_{ki} + t_{ii} \left[\frac{\sum_{m=1}^{n} m\delta_i^m}{n\Pi_i} + \frac{n-1}{2} \right]$$

$$t_{Ij} = t_{ij} - t_{ii} \left[\frac{\sum_{m=1}^{n} m\delta_i^m}{n\Pi_i} - \frac{n-1}{2} \right]$$

and

$$t'_{ii} = t_{ii}$$

$$t'_{ki} = t_{ki} + t_{ii} \frac{\sum_{m=1}^{n} m\delta_i^m}{n\Pi_i}$$

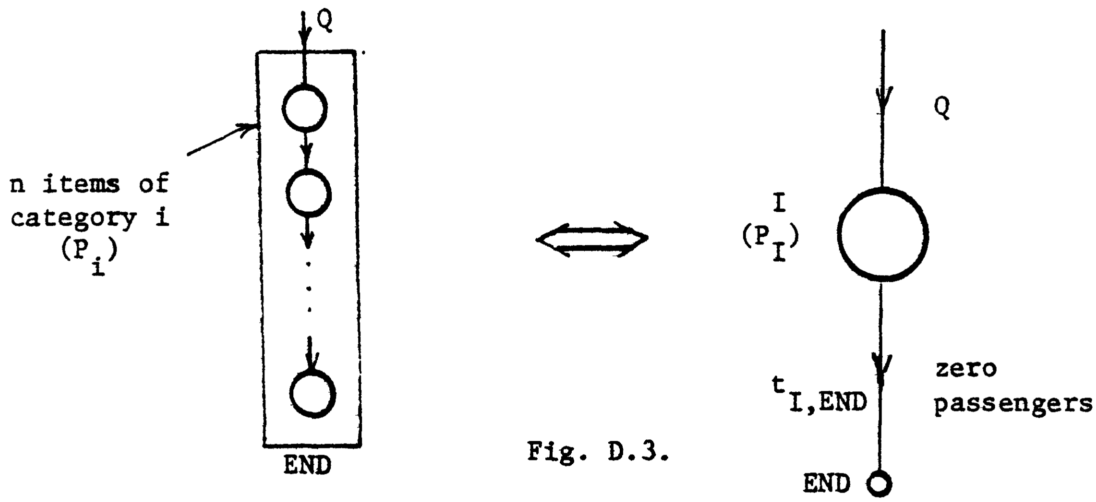


Fig. D.3.

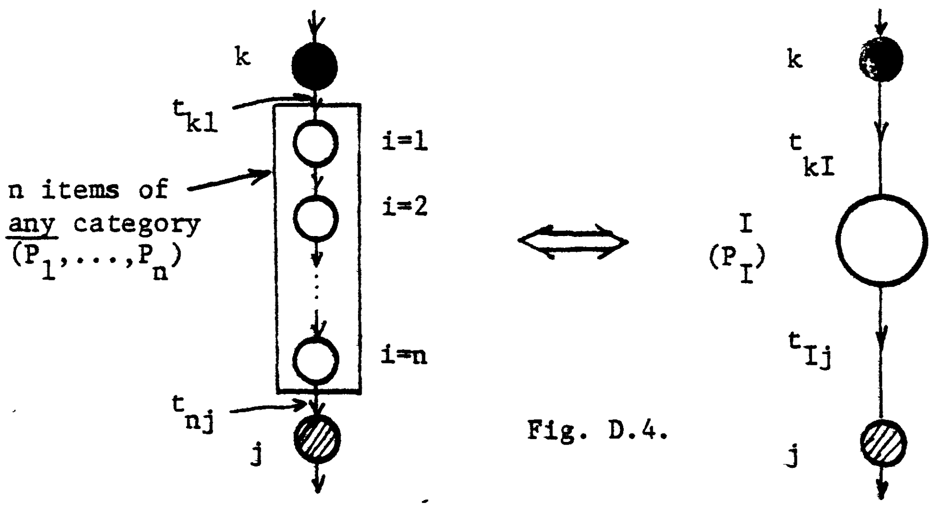


Fig. D.4.

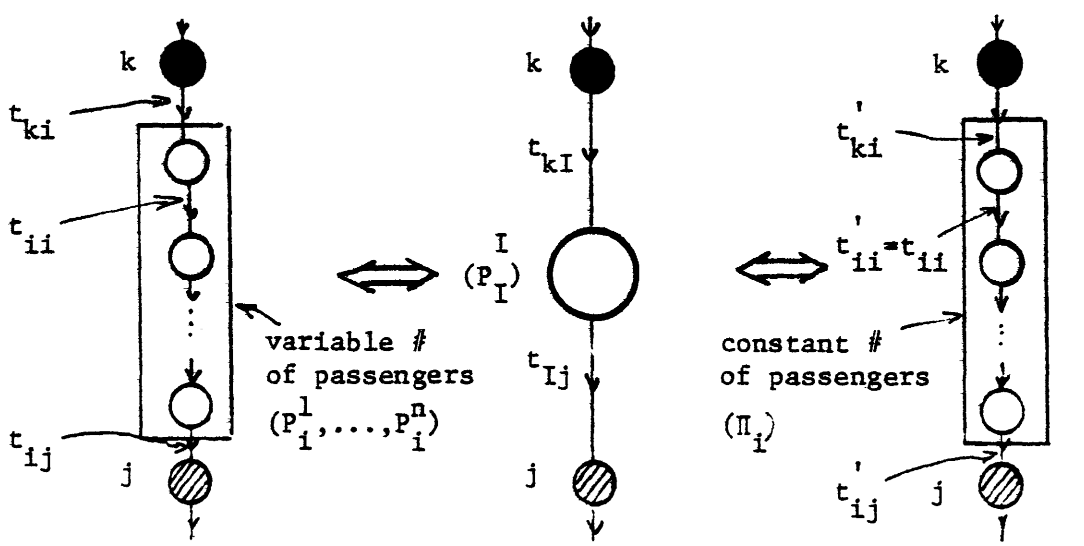


Fig. D.5.

$$t'_{ij} = t_{ij} - t_{ii} \frac{\sum_{m=1}^{n} m\delta_i^m}{n\Pi_i}$$

3) An important consequence of the above transformations emerges when we consider the value of the Total Passenger Delay associated with the segment of Fig. D.5. This value is equal to $C=Q.t_{kI} + (Q-P_I)t_{Ij}$ where Q is the total number of passengers from k to I.

Substituting we finally get

$$C = Q(t_{ki} + t_{ij}) - n\Pi_i t_{ij} + t_{ii} \sum_{m=1}^{n} m\delta_i^m$$

The last term in the expression is the only term we can affect by rearranging the airplanes in the segment. So, in order to minimize C we have to minimize $\sum_{m=1}^{n} m\delta_i^m$. The only way to do this is to rearrange the airplanes by <u>descending</u> order of number of passengers, so that finally $\delta_i^1 \leq \delta_1^2 \leq \ldots \leq \delta_i^n$.

This confirms the observation we have made in Appendix A when examinining variable numbers of passengers per category, namely that we can always improve on Total Passenger Delay by rearranging airplanes of the same category by descending order of number of passengers. (Result 18 of Appendix A.)

4) We show how to transform <u>two strings</u> of n_i planes of category i and n_j planes of category j (as shown in Fig.D.6) where the number of passengers is not constant, into two strings of n_i planes of category i and n_j planes of category j with constant numbers of passengers, Π_i, Π_j:

Defining:

$$\delta_i^m \equiv P_i^m - \Pi_i \qquad (m=1,\ldots,n_i)$$

$$\delta_j^m \equiv P_j^m - \Pi_j \qquad (m=1,\ldots,n_j)$$

It is not diffult to see that the equivalence formulae are:

$$\Pi_i = \frac{1}{n_i} \sum_{m=1}^{n_i} P_i^m$$

$$\Pi_j = \frac{1}{n_j} \sum_{m=1}^{n_j} P_j^m$$

$$t'_{ii} = t_{ii}$$

$$t'_{jj} = t_{jj}$$

$$t'_{ij} = t_{ij} + t_{jj} \frac{\sum_{m=1}^{n_j} \delta_j^m}{n_j\,\Pi_j} - t_{ii} \frac{\sum_{m=1}^{n_i} \delta_i^m}{n_i\,\Pi_i}$$

$$t'_{ji} = t_{ji} - t_{jj} \frac{\sum_{m=1}^{n_j} \delta_j^m}{n_j\,\Pi_j} + t_{ii} \frac{\sum_{m=1}^{n_j} \delta_i^m}{n_i\,\Pi_i}$$

5) A very important consequence of the last two relations is the following:

$$t'_{ij} + t'_{ji} = t_{ij} + t_{ji}$$

Since also $t'_{ii} = t_{ii}$ and $t'_{jj} = t_{jj}$ this means that:

$$t'_{ii} + t'_{jj} - t'_{ij} - t'_{ji} = t_{ii} + t_{jj} - t_{ij} - t_{ji}$$

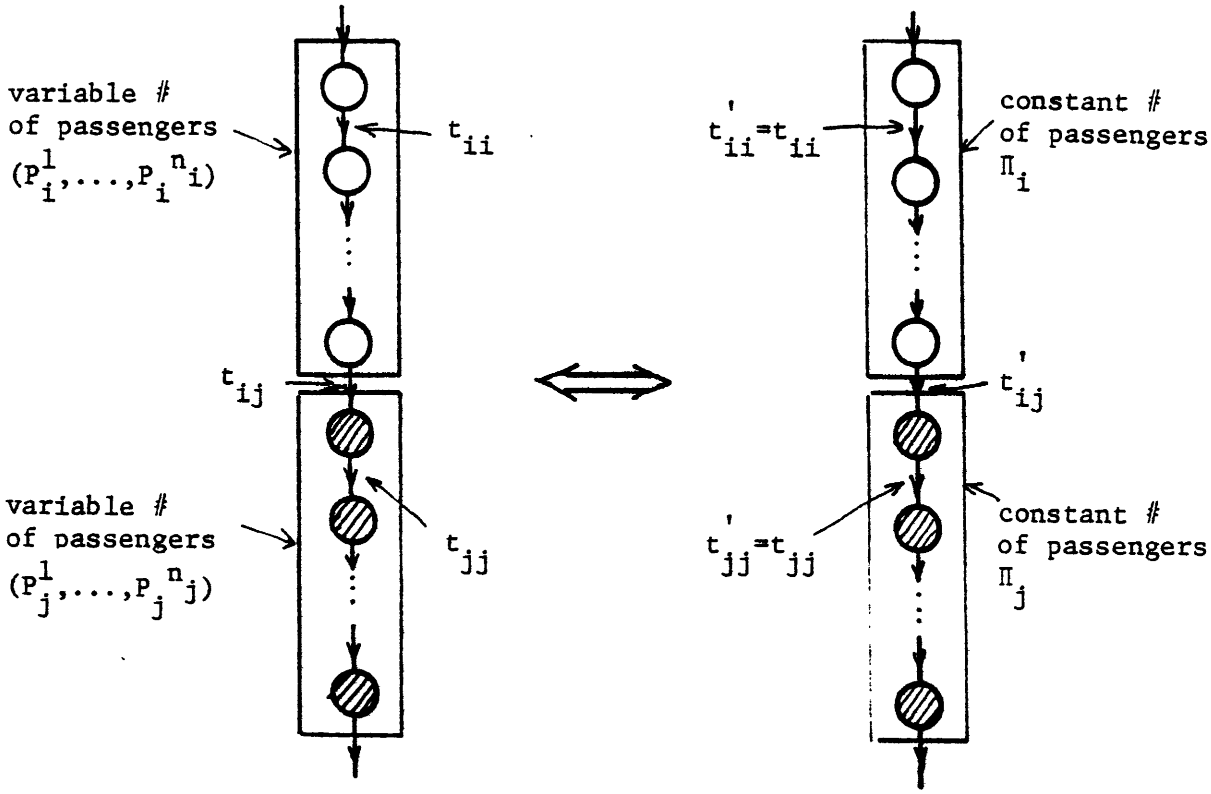

variable #
of passengers
$(P_i^1,\ldots,P_i^{n_i})$
t_{ii}
t_{ij}
variable #
of passengers
$(P_j^1,\ldots,P_j^{n_j})$
t_{jj}
$t'_{ii}=t_{ii}$
constant #
of passengers
Π_i
t'_{ij}
$t'_{jj}=t_{jj}$
constant #
of passengers
Π_j

Fig. D.6.

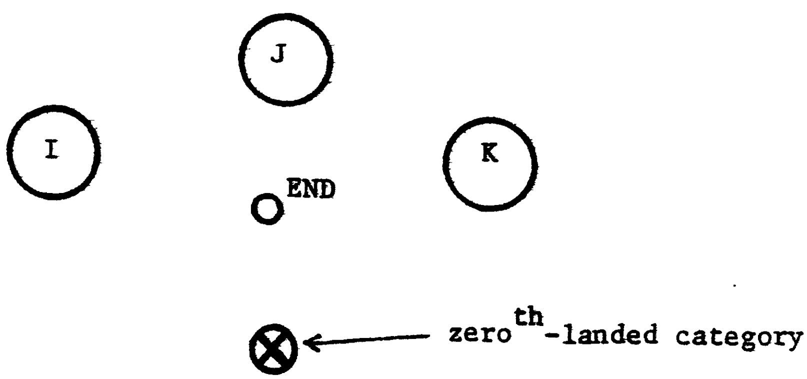

J
I
K
END
zeroth-landed category

Fig. D.7.

The importance of this equality is obvious: If the matrix $[t_{ij}]$ is (i,j)-reasonable (see Appendix C) this will mean that the right-hand-side is < 0. But this will also mean that the derived matrix $[t'_{ij}]$ is also (i,j)-reasonable. So we see that the (i,j)-reasonableness of a time separation matrix $[t_{ij}]$ is preserved when making transformations like those above.

Let us now see how we can modify our D.P. formulation to account for group clustering.

D.P. FORMULATION IN GROUP CLUSTERING

It is necessary to state at this point that this formulation should not be used if one is not sure that there will be group clustering. If we use it blindly, we will obtain suboptimal solutions. So suppose for the moment that we are somehow convinced that such a clustering will indeed occur.

Then our graph will consist of group nodes I,J,K, etc., corresponding to group clusterings of categories i,j,k, etc. (respectively), of an individual category node representing the zeroth landed category, and of the END node, as defined earlier. (In Figure D.7.)

The time separation matrix linking the group nodes is given by:

$$T_{IJ} = t_{ij} + \frac{1}{2}(n_i-1)t_{ii} + \frac{1}{2}(n_j-1)t_{jj}$$

On the other hand, we should also know how individual categories are linked with the groups. This is given by the following formula:

$$T'_{iJ} = t_{ij} + \frac{1}{2}(n_j-1)t_{jj}$$

The numbers of passengers of the groups are given by:

$$P_I = n_i P_i \qquad (i=1,\ldots,N)$$

The fundamental D.P. recursion is almost the same as before:

$$V_Z(L,k_1,\ldots,k_N) = \underset{x\varepsilon X}{\mathrm{Min}}[W_Z.\tau + V_Z(x,k_1',\ldots,k_N')]$$

where L is the currently landed <u>item</u> (which may be either <u>a whole group</u>, if we are at stage $n \leqslant N$, or <u>a single plane</u>, if we are at stage $n=N+1$). Note that by construction the k_i's are either zero or one, since we have at most one group per category. X is the set of i's with $k_i > 0$ and:

$$k_j' = \begin{cases} k_j - 1 & \text{if } j=x \text{ \underline{and} } n \leqslant N \\ k_j & \text{otherwise} \end{cases}$$

Also:

$$\tau = \begin{cases} T_{Lx} & \text{if } n \leqslant N \\ T'_{Lx} & \text{if } n=N+1 \end{cases}$$

where T and T' were defined in (a), (b) above. And finally:

$$W_Z = \begin{cases} 1 & \text{if } Z=1 \\ \sum_{j=1}^{N} k_j P_J & \text{if } Z=2 \end{cases}$$

As far as our boundary conditions are concerned, we have a minor change from our original formulation which stated that $V_Z(L,0,0...)=0$ for any $L=1,...N$ and for $Z=1,2$.

<u>This condition applies only for Z=2 here</u>.

For Z=1 we have to account for the elapsed time between the last group and the dummy node "END" as we saw earlier. So:

$$V_1(L,0,\dots,0) = \frac{(n_L-1)}{2}\, t_{LL}$$

With the above modifications we can execute our D.P. algorithm as usual. Note that we will have to rerun the recursions each time we change the n_i's because the time separation matrices [T] and [T'] will also change.

APPENDIX E

All computer programs of this work were written in FORTRAN IV and run using the Time-Sharing Option (TSO) of the IBM 370/168 system at M.I.T. All programs are interactive, prompting the user to enter the input, select the options, modify the data, etc. Specifically, the following programs were developed.

1) ASP-single runway-unconstrained case: This program implements the algorithm described in Chapter 4. A typical output of this program is shown in Table E.1. Referring to that table, it can be seen that the program prompts the user to enter the time separation matrix, the number of passengers and the problem's objective Z (as defined in Chapter 3). Subsequently, and after one pass of the "optimization" part has been executed, the program prompts the user to enter the initial state vector $(i_0, k_1^0, \ldots k_N^0)$ for the subsequent "identification" procedure, as described in Chapter 4. Referring again to Table E.1, we can translate the symbols appearing there as follows:

OPT.COST = V : Optimal value of the problem

STAGE n: Current stage

FROM a TO b --- c d e; a is the category of the previous landing aircraft, b is the category of the current landing. c. d. and e are the values of the components of the k-vector at

```
ENTER TIME  MATRIX
 96181228
 72 80117
 72 80 90
ENTER NOS. OF PASSENGERS
300150100
OBJECTIVE=? 1 FOR PURE TSP, 2 FOR WEIGHTED TSP
2
TIME MATRIX
 96 181 228
 72  80 117
 72  80  90
PASSENGERS  : 300 150 100
OBJECTIVE=  2
ENTER 0TH LAND. CAT. & 3 NOS. OF PLANES FROM 0 TO 5,     FORMAT(4I3) TO TERMINATE TYPE -1 OR -2 OR -3
  3  1  5  5
INIT. COND:
START FROM CAT.            3   ---  1  5  5   OBJECTIVE=  2                                              OPT.COST=     758550
STAGE 11   FROM  3   TO  2   ---  1  4  5   PASS= 1550   TIME= 80   SUBT1=   80  COST= 124000    SUBT2=   124000  REST=   634550
STAGE 10   FROM  2   TO  2   ---  1  3  5   PASS= 1400   TIME= 80   SUBT1=  160  COST= 112000    SUBT2=   236000  REST=   522550
STAGE  9   FROM  2   TO  2   ---  1  2  5   PASS= 1250   TIME= 80   SUBT1=  240  COST= 100000    SUBT2=   336000  REST=   422550
STAGE  8   FROM  2   TO  2   ---  1  1  5   PASS= 1100   TIME= 80   SUBT1=  320  COST=  88000    SUBT2=   424000  REST=   334550
STAGE  7   FROM  2   TO  1   ---  0  1  5   PASS=  950   TIME= 72   SUBT1=  392  COST=  68400    SUBT2=   492400  REST=   266150
STAGE  6   FROM  1   TO  2   ---  0  0  5   PASS=  650   TIME=181   SUBT1=  573  COST= 117650    SUBT2=   610050  REST=   148500
STAGE  5   FROM  2   TO  3   ---  0  0  4   PASS=  500   TIME=117   SUBT1=  690  COST=  58500    SUBT2=   668550  REST=    90000
STAGE  4   FROM  3   TO  3   ---  0  0  3   PASS=  400   TIME= 90   SUBT1=  780  COST=  36000    SUBT2=   704550  REST=    54000
STAGE  3   FROM  3   TO  3   ---  0  0  2   PASS=  300   TIME= 90   SUBT1=  870  COST=  27000    SUBT2=   731550  REST=    27000
STAGE  2   FROM  3   TO  3   ---  0  0  1   PASS=  200   TIME= 90   SUBT1=  960  COST=  18000    SUBT2=   749550  REST=     9000
STAGE  1   FROM  3   TO  3   ---  0  0  0   PASS=  100   TIME= 90   SUBT1= 1050  COST=   9000    SUBT2=   758550  REST=        0
ENTER 0TH LAND. CAT. & 3 NOS. OF PLANES FROM 0 TO 5,     FORMAT(4I3) TO TERMINATE TYPE -1 OR -2 OR -3
  3  2  5  5
INIT. COND:
START FROM CAT.            3   ---  2  5  5   OBJECTIVE=  2                                              OPT.COST=     936750
STAGE 12   FROM  3   TO  1   ---  1  5  5   PASS= 1850   TIME= 72   SUBT1=   72  COST= 133200    SUBT2=   133200  REST=   803550
STAGE 11   FROM  1   TO  1   ---  0  5  5   PASS= 1550   TIME= 96   SUBT1=  168  COST= 148800    SUBT2=   282000  REST=   654750
STAGE 10   FROM  1   TO  2   ---  0  4  5   PASS= 1250   TIME=181   SUBT1=  349  COST= 226250    SUBT2=   508250  REST=   428500
STAGE  9   FROM  2   TO  2   ---  0  3  5   PASS= 1100   TIME= 80   SUBT1=  429  COST=  88000    SUBT2=   596250  REST=   340500
STAGE  8   FROM  2   TO  2   ---  0  2  5   PASS=  950   TIME= 80   SUBT1=  509  COST=  76000    SUBT2=   672250  REST=   264500
STAGE  7   FROM  2   TO  2   ---  0  1  5   PASS=  800   TIME= 80   SUBT1=  589  COST=  64000    SUBT2=   736250  REST=   200500
STAGE  6   FROM  2   TO  2   ---  0  0  5   PASS=  650   TIME= 80   SUBT1=  669  COST=  52000    SUBT2=   788250  REST=   148500
STAGE  5   FROM  2   TO  3   ---  0  0  4   PASS=  500   TIME=117   SUBT1=  786  COST=  58500    SUBT2=   846750  REST=    90000
STAGE  4   FROM  3   TO  3   ---  0  0  3   PASS=  400   TIME= 90   SUBT1=  876  COST=  36000    SUBT2=   882750  REST=    54000
STAGE  3   FROM  3   TO  3   ---  0  0  2   PASS=  300   TIME= 90   SUBT1=  966  COST=  27000    SUBT2=   909750  REST=    27000
STAGE  2   FROM  3   TO  3   ---  0  0  1   PASS=  200   TIME= 90   SUBT1= 1056  COST=  18000    SUBT2=   927750  REST=     9000
STAGE  1   FROM  3   TO  3   ---  0  0  0   PASS=  100   TIME= 90   SUBT1= 1146  COST=   9000    SUBT2=   936750  REST=        0
ENTER 0TH LAND. CAT. & 3 NOS. OF PLANES FROM 0 TO 5,     FORMAT(4I3) TO TERMINATE TYPE -1 OR -2 OR -3
```

Table E.1: Typical output of computer program for the single runway-unconstrained case. This particular Example corresponds to Cases 7 and 6 of Chapter 4.

that stage. (The particular case of Table E.1 has N=3.)

PASS=x: Number of passengers still waiting to land at the current stage.

TIME = y: Time interval between the landing of a and the landing of b.

SUBT = Z_1: Incremental time counter.

COST = w: Passenger delay incurred between the landing of a and the landing of b (w=x.y)

SUBT2 = Z_2: Incremental passenger delay counter

REST = v: Value of optimal value function at that particular state.

2) ASP-single runway-CPS case: This program implements the algorithm developed in Chapter 5. A typical output of this program is shown in Table E.2. Two additional inputs for the program are the initial FCFS sequence and the value of MPS. In addition to the output produced by the previous program, this program also displays the position shifts of the various aircraft, as well as the percent improvement in LLT and TPD over the ones of the FCFS discipline. These are displayed in the following way: (See Table E.2.)

"TIME"	"COST"
LLT_o (FCFS)	TPD_o (FCFS)
LLT	TPD
LLT_o-LLT	TPD_o-TPD
$100(LLT_o\text{-}LLT)/LLT_o$	$100(TPD_o\text{-}TPD)/TPD_o$

3) ASP-Two runways-unconstrained case: This program implements the

```
ENTER MAXIMUM POSITION SHIFTING (MPS)  FORMAT(I2)
 5
TIME MATRIX
96181228
72 80117
72 80 90
PASSENGERS  : 300 150 100
OBJECTIVE=  2
CAT= 1 1 3 2 2 3 2 1 2 1 3 3 2 1 2   NUM=  5  6  4     MPS=  5
ENTER L0 OR -1,-2,-3,-4,-5,-6.FOR HELP HIT RETURN
 2
INIT. COND:
START FROM CAT.            2   ---  5  6  4   OBJECTIVE=  2                                                        OPT.COST=    1883250
STAGE 15   FROM  2   TO  2   ---  5  5  4   PASS= 2800   TIME= 80   SUBT1=   80  COST= 224000    SUBT2=   224000  REST=  1659250
STAGE 14   FROM  2   TO  1   ---  4  6  4   PASS= 2650   TIME= 72   SUBT1=  152  COST= 190800    SUBT2=   414800  REST=  1468450
STAGE 13   FROM  1   TO  1   ---  3  5  4   PASS= 2350   TIME= 96   SUBT1=  248  COST= 225600    SUBT2=   640400  REST=  1242850
STAGE 12   FROM  1   TO  1   ---  2  5  4   PASS= 2050   TIME= 96   SUBT1=  344  COST= 196800    SUBT2=   837200  REST=  1046050
STAGE 11   FROM  1   TO  1   ---  1  5  4   PASS= 1750   TIME= 96   SUBT1=  440  COST= 168000    SUBT2=  1005200  REST=   878050
STAGE 10   FROM  1   TO  2   ---  1  4  4   PASS= 1450   TIME=181   SUBT1=  621  COST= 262450    SUBT2=  1267650  REST=   615600
STAGE  9   FROM  2   TO  2   ---  1  3  4   PASS= 1300   TIME= 80   SUBT1=  701  COST= 104000    SUBT2=  1371650  REST=   511600
STAGE  8   FROM  2   TO  3   ---  1  3  3   PASS= 1150   TIME=117   SUBT1=  818  COST= 134550    SUBT2=  1506200  REST=   377050
STAGE  7   FROM  3   TO  3   ---  1  3  2   PASS= 1050   TIME= 90   SUBT1=  908  COST=  94500    SUBT2=  1600700  REST=   282550
STAGE  6   FROM  3   TO  2   ---  1  2  2   PASS=  950   TIME= 80   SUBT1=  988  COST=  76000    SUBT2=  1676700  REST=   206550
STAGE  5   FROM  2   TO  2   ---  1  1  2   PASS=  800   TIME= 80   SUBT1= 1068  COST=  64000    SUBT2=  1740700  REST=   142550
STAGE  4   FROM  2   TO  1   ---  0  1  2   PASS=  650   TIME= 72   SUBT1= 1140  COST=  46800    SUBT2=  1787500  REST=    95750
STAGE  3   FROM  1   TO  2   ---  0  0  2   PASS=  350   TIME=181   SUBT1= 1321  COST=  63350    SUBT2=  1850850  REST=    32400
STAGE  2   FROM  2   TO  3   ---  0  0  1   PASS=  200   TIME=117   SUBT1= 1438  COST=  23400    SUBT2=  1874250  REST=     9000
STAGE  1   FROM  3   TO  3   ---  0  0  0   PASS=  100   TIME= 90   SUBT1= 1528  COST=   9000    SUBT2=  1883250  REST=        0

MPS=  5
LANDING ORDER      :  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15      TIME         COST
INITIAL SEQUENCE   :  2  1  1  3  2  2  3  2  1  2  1  3  3  2  1  2      1729      2383800
OPTIMAL SEQUENCE   :  2  2  1  1  1  1  2  2  3  3  2  2  1  2  3  3      1528      1883250
POSITION SHIFTING  :  0  3 -1 -1  4  5 -1  0 -5 -3 -1  2  2  2 -3 -3       201       500550
PERCENT GAINS      :                                                        11           20

ENTER L0 OR -1,-2,-3,-4,-5,-6.FOR HELP HIT RETURN
-5
```

Table E.2: Typical output of the computer program for the single runway - CPS case. This particular example corresponds to Case 3 of Chapter 5.

```
  1  1  4  4  4
OPTIMAL PARTITIONING : RUNWAY 1 :   2  2  2    RUNWAY 2 :  2  2  2    MINMAX=       636

RUNWAY 1
START FROM CAT.            1    ---  2  2  2   OBJECTIVE=  1                                                          OPT.COST=          636
STAGE  6    FROM  1   TO   1    ---  1  2  2   PASS= 1100   TIME= 96    SUBT1=   96  COST= 105600     SUBT2=     105600  REST=          540
STAGE  5    FROM  1   TO   2    ---  1  1  2   PASS=  800   TIME=181    SUBT1=  277  COST= 144800     SUBT2=     250400  REST=          359
STAGE  4    FROM  2   TO   2    ---  1  0  2   PASS=  650   TIME= 80    SUBT1=  357  COST=  52000     SUBT2=     302400  REST=          279
STAGE  3    FROM  2   TO   3    ---  1  0  1   PASS=  500   TIME=117    SUBT1=  474  COST=  58500     SUBT2=     360900  REST=          162
STAGE  2    FROM  3   TO   3    ---  1  0  0   PASS=  400   TIME= 90    SUBT1=  564  COST=  36000     SUBT2=     396900  REST=           72
STAGE  1    FROM  3   TO   1    ---  0  0  0   PASS=  300   TIME= 72    SUBT1=  636  COST=  21600     SUBT2=     418500  REST=            0

RUNWAY 2
START FROM CAT.            1    ---  2  2  2   OBJECTIVE=  1                                                          OPT.COST=          636
STAGE  6    FROM  1   TO   1    ---  1  2  2   PASS= 1100   TIME= 96    SUBT1=   96  COST= 105600     SUBT2=     105600  REST=          540
STAGE  5    FROM  1   TO   2    ---  1  1  2   PASS=  800   TIME=181    SUBT1=  277  COST= 144800     SUBT2=     250400  REST=          359
STAGE  4    FROM  2   TO   2    ---  1  0  2   PASS=  650   TIME= 80    SUBT1=  357  COST=  52000     SUBT2=     302400  REST=          279
STAGE  3    FROM  2   TO   3    ---  1  0  1   PASS=  500   TIME=117    SUBT1=  474  COST=  58500     SUBT2=     360900  REST=          162
STAGE  2    FROM  3   TO   3    ---  1  0  0   PASS=  400   TIME= 90    SUBT1=  564  COST=  36000     SUBT2=     396900  REST=           72
STAGE  1    FROM  3   TO   1    ---  0  0  0   PASS=  300   TIME= 72    SUBT1=  636  COST=  21600     SUBT2=     418500  REST=            0

RUNWAY 1:   TIME=  636   COST=       418500

RUNWAY 2:   TIME=  636   COST=       418500
ENTER 0TH LAND. CAT. FOR RUNWAYS 1 AND 2 & 3 NOS. OF PLANES (MAX=5) FORMAT(5I3) TO TERMINATE TYPE -1 OR -2 OR -3
  2  2  5  5  5
OPTIMAL PARTITIONING : RUNWAY 1 :   2  0  5    RUNWAY 2 :  3  5  0    MINMAX=       664

RUNWAY 1
START FROM CAT.            2    ---  2  0  5   OBJECTIVE=  1                                                          OPT.COST=          645
STAGE  7    FROM  2   TO   3    ---  2  0  4   PASS= 1100   TIME=117    SUBT1=  117  COST= 128700     SUBT2=     128700  REST=          528
STAGE  6    FROM  3   TO   3    ---  2  0  3   PASS= 1000   TIME= 90    SUBT1=  207  COST=  90000     SUBT2=     218700  REST=          438
STAGE  5    FROM  3   TO   3    ---  2  0  2   PASS=  900   TIME= 90    SUBT1=  297  COST=  81000     SUBT2=     299700  REST=          348
STAGE  4    FROM  3   TO   3    ---  2  0  1   PASS=  800   TIME= 90    SUBT1=  387  COST=  72000     SUBT2=     371700  REST=          258
STAGE  3    FROM  3   TO   3    ---  2  0  0   PASS=  700   TIME= 90    SUBT1=  477  COST=  63000     SUBT2=     434700  REST=          168
STAGE  2    FROM  3   TO   1    ---  1  0  0   PASS=  600   TIME= 72    SUBT1=  549  COST=  43200     SUBT2=     477900  REST=           96
STAGE  1    FROM  1   TO   1    ---  0  0  0   PASS=  300   TIME= 96    SUBT1=  645  COST=  28800     SUBT2=     506700  REST=            0

RUNWAY 2
START FROM CAT.            2    ---  3  5  0   OBJECTIVE=  1                                                          OPT.COST=          664
STAGE  8    FROM  2   TO   2    ---  3  4  0   PASS= 1650   TIME= 80    SUBT1=   80  COST= 132000     SUBT2=     132000  REST=          584
STAGE  7    FROM  2   TO   2    ---  3  3  0   PASS= 1500   TIME= 80    SUBT1=  160  COST= 120000     SUBT2=     252000  REST=          504
STAGE  6    FROM  2   TO   2    ---  3  2  0   PASS= 1350   TIME= 80    SUBT1=  240  COST= 108000     SUBT2=     360000  REST=          424
STAGE  5    FROM  2   TO   2    ---  3  1  0   PASS= 1200   TIME= 80    SUBT1=  320  COST=  96000     SUBT2=     456000  REST=          344
STAGE  4    FROM  2   TO   2    ---  3  0  0   PASS= 1050   TIME= 80    SUBT1=  400  COST=  84000     SUBT2=     540000  REST=          264
STAGE  3    FROM  2   TO   1    ---  2  0  0   PASS=  900   TIME= 72    SUBT1=  472  COST=  64800     SUBT2=     604800  REST=          192
STAGE  2    FROM  1   TO   1    ---  1  0  0   PASS=  600   TIME= 96    SUBT1=  568  COST=  57600     SUBT2=     662400  REST=           96
STAGE  1    FROM  1   TO   1    ---  0  0  0   PASS=  300   TIME= 96    SUBT1=  664  COST=  28800     SUBT2=     691200  REST=            0

RUNWAY 1:   TIME=  645   COST=       506700

RUNWAY 2:   TIME=  664   COST=       691200
ENTER 0TH LAND. CAT. FOR RUNWAYS 1 AND 2 & 3 NOS. OF PLANES (MAX=5) FORMAT(5I3) TO TERMINATE TYPE -1 OR -2 OR -3
```

Table E.3: Typical output of the computer program for the two-runway-unconstrained case. This particular example corresponds to Cases 1 and 3 of Chapter 6.

```
ENTER NOS. OF PASSENGERS
110110130
OBJECTIVE=? 1 FOR PURE TSP, 2 FOR WEIGHTED TSP
2
TYPE 1 FOR A PRIORI GROUPING, ELSE RETURN
1
TIME MATRIX
 70 100 130
 70  80 110
 70  80  90
PASSENGERS  : 110 110 130
OBJECTIVE=  2
GROUPING=  1
ENTER NOS. OF PLANES FOR 3 CATEGORIES, MAX=5 EACH
  5  5  5
ENTER 0TH LAND.CAT.FORMAT(4I3) OR TYPE -1 OR -2 OR -3 OR -4
  3
INIT. COND:
START FROM CAT.          3   ---  1  1  1   OBJECTIVE=  2                                                   OPT.COST=   1121500
STAGE  3   FROM  3   TO  3   ---  1  1  0   PASS= 1750   TIME=270   SUBT1=  270  COST= 472500   SUBT2=   472500  REST=    649000
STAGE  2   FROM  3   TO  1   ---  0  1  0   PASS= 1100   TIME=390   SUBT1=  660  COST= 429000   SUBT2=   901500  REST=    220000
STAGE  1   FROM  1   TO  2   ---  0  0  0   PASS=  550   TIME=400   SUBT1= 1060  COST= 220000   SUBT2=  1121500  REST=         0
STAGE  0   FROM  2   TO  END ---  0  0  0   PASS=    0   TIME=160   SUBT1= 1220  COST=      0   SUBT2=  1121500  REST=         0
ENTER 0TH LAND.CAT.FORMAT(4I3) OR TYPE -1 OR -2 OR -3 OR -4
-2
ENTER NOS. OF PASSENGERS
110110120
OBJECTIVE=? 1 FOR PURE TSP, 2 FOR WEIGHTED TSP
2
TYPE 1 FOR A PRIORI GROUPING, ELSE RETURN
1
TIME MATRIX
 70 100 130
 70  80 110
 70  80  90
PASSENGERS  : 110 110 120
OBJECTIVE=  2
GROUPING=  1
ENTER NOS. OF PLANES FOR 3 CATEGORIES, MAX=5 EACH
  5  5  5
ENTER 0TH LAND.CAT.FORMAT(4I3) OR TYPE -1 OR -2 OR -3 OR -4
  3
INIT. COND:
START FROM CAT.          3   ---  1  1  1   OBJECTIVE=  2                                                   OPT.COST=   1087000
STAGE  3   FROM  3   TO  1   ---  0  1  1   PASS= 1700   TIME=210   SUBT1=  210  COST= 357000   SUBT2=   357000  REST=    730000
STAGE  2   FROM  1   TO  2   ---  0  0  1   PASS= 1150   TIME=400   SUBT1=  610  COST= 460000   SUBT2=   817000  REST=    270000
STAGE  1   FROM  2   TO  3   ---  0  0  0   PASS=  600   TIME=450   SUBT1= 1060  COST= 270000   SUBT2=  1087000  REST=         0
STAGE  0   FROM  3   TO  END ---  0  0  0   PASS=    0   TIME=180   SUBT1= 1240  COST=      0   SUBT2=  1087000  REST=         0
ENTER 0TH LAND.CAT.FORMAT(4I3) OR TYPE -1 OR -2 OR -3 OR -4
```

Table E.4: Typical output of the computer program for the single runway-unconstrained case with a priori clustering (grouping) as developed in Appendix D. This particular example corresponds to Cases 9 and 8 of Chapter 4.

algorithm of Chapter 6. A typical output of this program is shown in Table E.3.

As in the single runway unconstrained case, the main input concerns the time separation matrix, the number of passengers and the objective. Subsequently, and after one pass of the "optimization" part of the equivalent single runway program has been executed, the program prompts the user to enter the zeroth landed categories at the two runways and the initial composition of our aircraft reservoir (in Table E.3 we have, for example, (1,1,4,4,4) and (2,2,5,5,5)). The output of the program consists of the optimal partitioning as well as the optimal sequences for each of the two runways. Referring to Table E.3, we can explain the various symbols as follows:

"MINMAX": Optimal value of the problem (in terms of our objective)

"TIME": Last Landing Time

"COST": Total Passenger Delay

The meaning of the rest of the symbols is identical to that of Table E.1.

4) ASP-single runway-unconstrained case with a priori group clustering. This program implements the modified D.P. algorithm presented at the end of Appendix D when we are a priori sure that our airplanes will be clustered in groups. A typical session is shown in Table E.4. This program can also be run as the regular single runway-unconstrained case program (Table E.1) if desired. The option parameter is called "GROUPING" and is equal to 1 if a priori group clustering is desired, 0 otherwise (Table E.4). As shown in Appendix D, our sequence terminates with a dummy node called "END."