

68

DYNAMIC PROGRAMMING ALGORITHMS
FOR SPECIALLY STRUCTURED SEQUENCING AND ROUTING
PROBLEMS IN TRANSPORTATION

by

HARILAOS NICHOLAS PSARAFTIS

Diploma, National Technical University of Athens
(1974)

S.M., Massachusetts Institute of Technology
(1977)

S.M., Massachusetts Institute of Technology
(1977)

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE
DEGREE OF
DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September, 1978
G.E. June 1979

Signature of Author Signature redacted

..... Department of Ocean Engineering

Certified by Signature redacted

Thesis Supervisor

Accepted by Signature redacted

Chairman, Departmental Committee on Graduate Students

© Massachusetts Institute of Technology, 1978

ARCHIVES
MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

DEC 14 1978

LIBRARIES

I devote this thesis to my parents, Nicholas and Irene. I am greatly indebted to them for the affection, encouragement and support they have given me throughout the duration of my studies.

DYNAMIC PROGRAMMING ALGORITHMS FOR SPECIALLY STRUCTURED
SEQUENCING AND ROUTING PROBLEMS IN TRANSPORTATION

by

Harilaos N. Psaraftis

Submitted to the Department of Ocean Engineering in partial fulfillment
of the requirements of the degree of Doctor of Philosophy

ABSTRACT

In this thesis, a number of dynamic programming algorithms are developed for several sequencing and routing problems in Transportation that exhibit special structures. First, three versions of the problem of sequencing aircraft landings at an airport are examined. 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.

We then proceed to examine two versions of the dial-a-ride problem. These versions involve the dispatching of a vehicle to carry certain customers from distinct origins to distinct destinations usually in an urban environment. All customers request immediate service by telephone. Vehicle capacity constraints and priority rules similar to CPS are part of our problem. We study a generalized objective to minimize a weighted combination of the time to service the customers and their corresponding total disutility. In the "static" version, no intermediate customer requests are considered until the service of all initial customers is accomplished. In the "dynamic" version, an update is made each time a new customer requests service. The algorithms for both cases are exponential

but perform asymptotically better than the classical D.P. algorithm applied to a Travelling Salesman Problem of the same size.

All algorithms of the thesis 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.

THESIS SUPERVISOR:

Amedeo R. Odoni

TITLE:

Associate Professor of
Aeronautics and Astronautics

TABLE OF CONTENTS

	<u>Page</u>
ABSTRACT	3
TABLE OF CONTENTS	5
ACKNOWLEDGEMENTS	9
LIST OF FIGURES	11
LIST OF TABLES	15
INTRODUCTION AND OUTLINE	16
<u>PART I: GENERAL BACKGROUND</u>	26
<u>CHAPTER 1: Algorithms and Computational Efficiency</u>	27
1.0 Introduction: Performance Aspects of Algorithms	27
1.1 The Concept of NP-Completeness in Combinatorial Optimization Problems	32
1.2 Remedies for NP-Completeness	36
<u>CHAPTER 2: The Dynamic Programming Solution to the Travelling Salesman Problem.</u>	42
2.0 Introduction	42
2.1 The Dynamic Programming Algorithm	45
2.2 Comparison with Other Approaches	47
<u>PART II: THE AIRCRAFT SEQUENCING PROBLEM (ASP)</u>	52
<u>CHAPTER 3: Aircraft Sequencing: Problem Description</u>	53
3.0 Introduction	53
3.1 Problem Formulation	56
3.2 Graph Representation of the ASP	58
<u>CHAPTER 4: ASP-Unconstrained Case: Dynamic Programming Solution</u>	66

<u>TABLE OF CONTENTS (CONT'D)</u>	<u>Page</u>
4.0 Introduction: Dynamic Programming Formulation	66
4.1 Solution Algorithm	68
4.2 Computational Effort and Storage Requirements	72
4.3 Computer Runs - Discussion of the Results	74
4.4 Directions for Subsequent Investigation	81
<u>CHAPTER 5: ASP-Constrained Position Shifting (CPS)-Dynamic Programming Solution</u>	83
5.0 Introduction: Meaning of CPS	83
5.1 Outline of Solution Approach Using Dynamic Programming	88
5.2 The D.P. Algorithm	93
5.3 Computational Effort and Storage Requirements	94
5.4 Computer Runs - Discussion of the Results	96
<u>CHAPTER 6: The Two-Runway ASP</u>	103
6.0 Introduction: Formulation and Complexity of the Problem	103
6.1 Solution Algorithm	107
6.2 Computer Runs - Discussion of the Results	110
6.3 Further Remarks on the Two-Runway ASP	117
6.4 Summary of the DP Approach to the ASP	118
<u>PART III: THE DIAL-A-RIDE PROBLEM</u>	120
<u>CHAPTER 7: Dial-A-Ride: Problem Description</u>	121
7.0 Introduction	121
7.1 Measures of Performance and Constraints	124
7.2 Existing Solution Procedures	126
<u>CHAPTER 8: Dial-A-Ride- "Static" Case: Dynamic Programming Solution</u>	132

TABLE OF CONTENTS (CONT'D)	Page
8.0 Introduction	132
8.1 Discussion of Alternative Objectives	135
8.2 Constraints	137
8.3 Preliminary Considerations for the D.P. Approach	144
8.4 Feasibility Considerations	146
8.5 Optimality Considerations	153
8.6 The D.P. Algorithm	155
8.7 Computational Effort and Storage Requirements	156
8.8 Computer Runs-Discussion of the Results	157
<u>CHAPTER 9:</u> Dial-A-Ride-"Dynamic" Case: Dynamic Programming Solution	169
9.0 Introduction: Problem Formulation and Approach	169
9.1 Solution Algorithm	176
9.2 Computer Runs: Discussion of the Results	178
<u>CHAPTER 10:</u> Improving the Computational Efficiency of the Dial-A-Ride Algorithm	197
10.0 Introduction	197
10.1 Reducing Storage Requirements by a Factor of Three.	197
10.2 Preliminary Observations Concerning the Structure of the Problem	200
10.3 Stage-by-Stage Solution Procedure	203
10.4 Combinatorics of the Dial-A-Ride Problem	206
10.5 Indexing Considerations in Stage-by-Stage Solution	218
10.6 Summary of the DP Approach to the Dial-A-Ride Problem.	225

TABLE OF CONTENTS (CONT'D)

	<u>Page</u>
<u>PART IV: FINAL REMARKS</u>	228
<u>CHAPTER 11: Conclusions and Directions for Further Research</u>	229
11.0 Introduction	229
11.1 ASP: Review of the results of this work	229
11.2 ASP: "Dynamic" Case	231
11.3 ASP: Multiple Runways	236
11.4 ASP: Usefulness of the "Steepest Descent" Heuristic for TPD Minimization	238
11.5 ASP: Implementation Issues	239
11.6 Dial-A-Ride: Review of the results of this work	240
11.7 Dial-A-Ride: The Multi-Vehicle Problem	242
11.8 Dial-A-Ride: Incorporating Time Constraints	244
11.9 Dial-A-Ride: Probabilistic Extention of the "Dynamic" Case	245
11.10 Dial-A-Ride: Incorporating Heuristics into the D.P. Algorithm	246
11.11 Dial-A-Ride: Implementation Issues	246
REFERENCES	249
<u>PART V: APPENDICES</u>	253
<u>APPENDIX A: ASP-Investigation of Group "Clustering"</u>	254
<u>APPENDIX B: ASP-Derivation of the Time Separation Matrix</u>	271
<u>APPENDIX C: ASP-On the "Reasonableness" of the Time Separation Matrix</u>	275
<u>APPENDIX D: ASP-Equivalence Transformations in Group "Clustering"</u>	280
<u>APPENDIX E: The Computer Programs.</u>	293

ACKNOWLEDGEMENTS

I would first like to thank the members of my Doctoral Thesis Committee, Professors Odoni, Papadimitriou, Devanney, Wilson and Marcus for their assistance.

My deepest gratitude goes to Professor Amedeo Odoni of the Department of Aeronautics and Astronautics who acted as Chairman of the Committee and Supervisor throughout my Doctoral Program. His enthusiasm, support and continuous encouragement have been the main forces behind my interest in Operations Research and Transportation Systems Analysis and helped strengthen my confidence in getting actively involved in such broad disciplines not closely related to my former purely engineering background. His comments and constant accessibility have contributed strongly to the successful completion of this dissertation. Finally his true concern for me as a friend has set an example which I shall not forget.

I also had the luck to benefit from the experience of my friend Professor Christos Papadimitriou of the Department of Computer Science at Harvard University, on issues related to algorithmic complexity in this work. I would like to thank him for his suggestions as well as for the considerable time he spent discussing the thesis with me.

I am particularly indebted to Professor John Devanney of the Department of Ocean Engineering for his encouragement. His suggestions on how to tackle specific algorithmic and computer implementation problems and especially the "dynamic" version of the dial-a-ride problem have been extremely helpful.

Professor Nigel Wilson of the Department of Civil Engineering provided an important stimulus at the beginning of this work by suggesting an alternative objective function to that of the classical Travelling Salesman Problem. His interest in the dial-a-ride part of the thesis and the financial support that he provided through his project on this subject are greatly appreciated.

Professor Henry Marcus of the Department of Ocean Engineering and of Civil Engineering helped me by discussing applications of this work in the Marine Transportation environment (Seaport Scheduling).

In addition to the above, I would like to thank Professor Robert Simpson of the Department of Aeronautics and Astronautics for the time he spent reviewing with me the Aircraft Sequencing algorithms.

My supervisor during my Master's Program in Ocean Engineering, Professor Chryssostomos Chryssostomidis provided me with unlimited access to the equipment and supplies of the computer facility of the Design Laboratory. I am particularly indebted to him for his assistance and encouragement throughout my studies at M.I.T.

I am finally grateful to Tonia Fardis for the remarkable job she has done in typing the manuscript, which, by the way, was not always easy to decipher.

LIST OF FIGURES

<u>Fig.</u>		<u>Page</u>
1.1	Example graph for the MSTP and TSP	33
1.2	Optimal solutions to the MSTP and TSP of the graph Fig. 1.1	33
2.1	A 4-node graph and its corresponding cost matrix $[d_{ij}]$	44
3.1	Graph representation of the ASP	59
3.2	A sequence of aircraft landings	59
3.3	Graph size reduction due to category classifications	64
4.1	ASP-unconstrained Case: Case 1	76
4.2	ASP-unconstrained Case: Case 2	76
4.3	ASP-unconstrained Case: Cases 3 and 4	76
4.4	ASP-unconstrained Case: Case 5	76
4.5	ASP-unconstrained Case: Case 6	80
4.6	ASP-unconstrained Case: Case 7	80
4.7	ASP-unconstrained Case: Further analysis of Case 7	80
4.8	ASP-unconstrained Case: Case 8	80
4.9	ASP-unconstrained Case: Case 9	80
5.1	ASP-CPS Case: Case 1	98
5.2	ASP-CPS Case: Case 2	98
5.3	ASP-CPS Case: Case 3	98
5.4	ASP-CPS Case: Case 4	100
5.5	ASP-CPS Case: Case 5	100
5.6	% improvement in LLT with respect to the FCFS discipline	102
5.7	% improvement in TPD with respect to the FCFS discipline	102
6.1	ASP-Two runway Case: Example	112

LIST OF FIGURES (CONT'D)

<u>Fig.</u>		<u>Page</u>
6.2	ASP-Two runway Case: Case 1	112
6.3	ASP-Two runway Case: Case 2	112
6.4	ASP-Two runway Case: Case 3	115
6.5	ASP-Two runway Case: Case 4	115
6.6	ASP-Two runway Case: Case 5	115
6.7	ASP-Two runway Case: Case 6	115
7.1	Sub-optimality of an existing "dynamic" algorithm	129
8.1	Origins and destinations in the dial-a-ride problem	133
8.2	A vehicle route	133
8.3	Different objectives yield different optimal routes	138
8.4	Infeasible routes	143
8.5	An example of the "static" Case of the dial-a-ride problem	158
8.6	Dial-A-Ride-"Static" Case: Case 1	162
8.7	Dial-A-Ride-"Static" Case: Case 2	163
8.8	Dial-A-Ride-"Static" Case: Case 3	164
8.9	Dial-A-Ride-"Static" Case: Case 4	165
8.10	Dial-A-Ride-"Static" Case: Case 5	166
8.11	Dial-A-Ride-"Static" Case: Case 6	167
8.12	Dial-A-Ride-"Static" Case: Case 7	168
9.1	Updating scheme in the "dynamic" case	172
9.2	Dial-A-Ride-"Dynamic" Case: Case 1, update #1	187
9.3	Dial-A-Ride-"Dynamic" Case: Case 1, update #2	188
9.4	Dial-A-Ride-"Dynamic" Case: Case 1, update #3	189
9.5	Dial-A-Ride-"Dynamic" Case: Case 1, update #4	190

LIST OF FIGURES (CONT'D)

<u>Fig.</u>		<u>Page</u>
9.6	Dial-A-Ride-"Dynamic" Case: Case 1, update #5	191
9.7	Dial-A-Ride-"Dynamic" Case: Case 2, update #1	192
9.8	Dial-A-Ride-"Dynamic" Case: Case 2, update #2	193
9.9	Dial-A-Ride-"Dynamic" Case: Case 2, update #3	194
9.10	Dial-A-Ride-"Dynamic" Case: Case 2, update #4	195
9.11	Dial-A-Ride-"Dynamic" Case: Case 2, update #5	196
10.1	Decision graph of the dial-a-rideproblem for $N=2$	201
11.1	ASP-"Dynamic" Case	232
A.1	ASP-Investigation of group "clustering": Results 1 and 2	257
A.2	ASP-Investigation of group "clustering": Result 3	257
A.3	ASP-Investigation of group "clustering": Result 3	257
A.4	ASP-Investigation of group "clustering": Result 4	257
A.5	ASP-Investigation of group "clustering": Results 6 and 7	262
A.6	ASP-Investigation of group "clustering": Result 7	262
A.7	ASP-Investigation of group "clustering": Result 7	262
A.8	ASP-Investigation of group "clustering": Result 8	262
A.9	ASP-Investigation of group "clustering": Results 10 and 11	266
A.10	ASP-Investigation of group "clustering": Result 12	266
A.11	ASP-Investigation of group "clustering": Result 15	266
A.12	ASP-Investigation of group "clustering": Result 18	266
B.1	"Overtaking" Case ($v_j > v_i$)	274
B.2	"Opening" Case ($v_i > v_j$)	274
C.1	ASP-Reasonableness of $[t_{ij}]$ and consequences	279
D.1	ASP-Equivalence transformations in group clustering	263

LIST OF FIGURES (CONT'D)

<u>Fig.</u>		<u>Page</u>
D.2	ASP-Equivalence transformations in group clustering .	283
D.3	ASP-Equivalence transformations in group clustering	286
D.4	ASP-Equivalence transformations in group clustering	286
D.5	ASP-Equivalence transformations in group clustering	286
D.6	ASP-Equivalence transformations in group clustering	289
D.7	ASP-Equivalence transformations in group clustering	289

LIST OF TABLES

<u>Table</u>		<u>Page</u>
10.1	List of all consistent states of stage $n=4$ for $N=3$	221
10.2	Directory for $N=3$	226
E.1	Typical output of computer program for the single runway-unconstrained case	294
E.2	Typical output of computer program for the single runway - CPS case.	296
E.3	Typical output of computer program for the two-runway-unconstrained case	297
E.4	Typical output of Computer program for the single runway-unconstrained case with a priori clustering 299	299
E.5	Typical output of computer program for the dial-a-ride-"static" case	301
E.6	Typical output of computer program for the dial-a-ride-"dynamic" case.	304

INTRODUCTION AND OUTLINE

The main purpose of this dissertation is to investigate several combinatorial optimization problems that are related to important real-world problems in Transportation. These problems are attracting the attention and research effort of several groups today.

Our investigation encompasses the development of analytical models describing the above real-world problems, as well as the design, testing and refinement of novel, specialized and efficient solution procedures tailored to specific versions of these problems. 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 thesis, 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 of this dissertation is organized into five parts:

- a) Part I presents the necessary background for the investigation to follow.
- b) Part II is devoted to the examination of a specially structured sequencing problem in Air Transportation, the Aircraft Sequencing Problem.
- c) Part III examines a specially structured routing problem in Ground Transportation, the Dial-A-Ride Problem.
- d) Part IV reviews the results of this work and suggests directions for further research.
- e) Finally, Part V includes several appendices with additional technical material on the Aircraft Sequencing Problem and a description of the computer programs implementing the algorithms of the thesis.

Specifically, the organization of the dissertation 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 sequencing and routing problems 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 problems 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 of 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 minimax problem. Our alternative objective is, as before, to minimize TPD. We see that this problem essentially involves the partitioning 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) Part III: The Dial-A-Ride Problem.

In Chapter 7 we introduce an important Vehicle Routing problem, the Dial-A-Ride problem. In general, dial-a-ride involves the dispatching of several vehicles in order to carry customers from specified origins to specified destinations. These customers have notified by phone the

vehicle operating agency about their wishes. We review several versions of this problem and outline the main points of the algorithms used to date for its solution. The version of the problem we examine in detail is a single vehicle, "many-to-many," "immediate-request" one. "Many-to-many" means that the origins (pick-up points) as well as the destinations (delivery points) of the various customers are all distinct points. "Immediate-request" means that every customer requesting service wishes to be serviced as soon as possible. Our general objective function is to minimize a weighted* combination of the time to service all customers and of the total "disutility" they suffer until their delivery. This disutility is assumed to be a linear combination of the time each customer waits to be picked up and of the time he spends riding in the vehicle till his delivery. Our constraints are the following:

- a) Vehicle routes should be legitimate, namely each customer has to be picked up before being delivered.
- b) The vehicle has a certain customer capacity which cannot be exceeded.
- c) Priority constraints, similar to the CPS rules in the ASP (Chapter 5), but now concerning both the pick-up order of each customer, as well as his delivery order must be satisfied. (For details see Chapters 7 and 8.)

In Chapter 8 we develop a D.P. algorithm for solving the "static" version of the above problem. In this version, no new customer requests are considered, until the service of all of the initial set of customers

*The weights are inputs to our problem.

is accomplished. We see that our D.P. formulation takes care of the route legitimacy constraints automatically. In addition, the other constraints of our problem can be formulated mathematically so as to fit our D.P. algorithm in a particularly convenient fashion. The computational effort of the algorithm is an exponential function of the number of customers. Still, our algorithm performs asymptotically better than the equivalent classical D.P. algorithm, when the latter is applied to a TSP of equal size. Computer runs and route plottings for several cases of the problem are presented and discussed.

In Chapter 9 we extend the "static" algorithm to its equivalent "dynamic" version, namely the version where new customer requests occur continually, but unpredictably in time. In this version, we perform an update of our solution whenever a new customer requests service. Our algorithm does not anticipate new customer requests (in a probabilistic or any other way), so that new requests become part of the problem's input only upon appearance. Our priority rules (introduced already in the "static" case) serve in this "dynamic" case to prevent the undesirable phenomenon of indefinite deferment of a customer's request. Several computer runs are performed, examining two different objectives applied to the same chronological sequence of customer requests. The results lead to some interesting observations concerning our "dynamic" optimization philosophy.

In Chapter 10 we present an alternative way to solve the "static" problem by D.P. By contrast to the algorithm described in Chapter 8, the new algorithm solves the problem in a stage-by-stage manner, leading to more efficient use of storage space. An investigation of the

combinatorics of the problem is also made and the indexing problems of the new algorithm are addressed. Depending on the computer facility used, this new algorithm, while not offering any spectacular improvements over the old one, enables us nevertheless to solve optimally problems with up to 8 or 9 customers, a capability, that is, which is sufficiently adequate for most real-world scheduling problems for a single vehicle.

4) Part IV: Final Remarks

In Chapter 11 we review the main results of this thesis, suggest directions for possible extensions and address issues on the implementation of the developed algorithms. In particular, the "dynamic" version of the ASP and the multi-vehicle version of dial-a-ride are seen to constitute very important extensions to this work. Other issues discussed include:

- a) ASP: Extension to the case of multiple runways.
- b) Dial-A-Ride: Inclusion of time constraints.
- c) Dial-A-Ride: Probabilistic extension of the "dynamic" case.
- d) Dial-A-Ride: Possible role of heuristics in the D.P. algorithm.

5) Part V: Appendices

These appendices present additional technical issues related mostly to ASP problem (with the exception of Appendix E). 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 in this thesis.

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 thesis.

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 better than others 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 [TURI 37].

does this efficiently. 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 $n \leq 5$, while the opposite happens for $n \geq 6$.

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 ($n \rightarrow +\infty$). 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 graph is a collection of points, called nodes, which represent entities, and lines, called links, 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" algorithm 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 $\lim_{n \rightarrow \infty} \frac{n^{80}}{1.001^n} = 0$, 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 , $n \log n$, 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 exercise 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 sequencing and routing problems, which we will be studying throughout this dissertation 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).

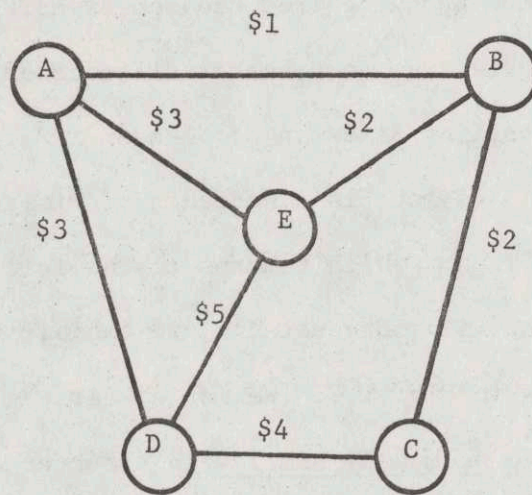


Fig. 1.1: Example graph for the MSTP and TSP

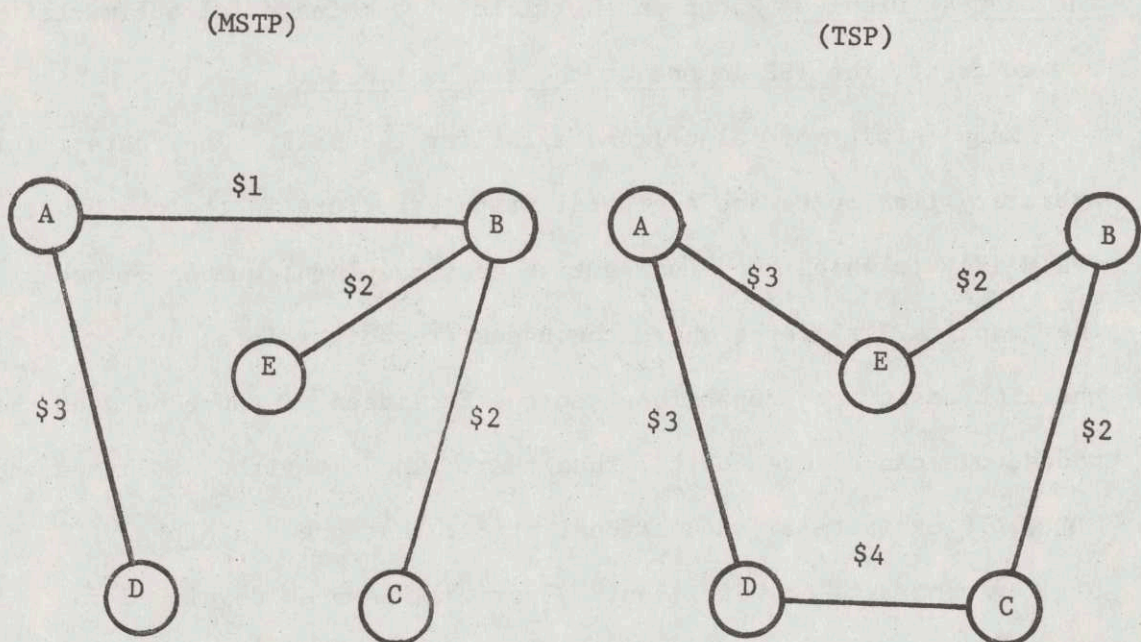


Fig. 1.2 : Optimal solutions to the MSTP and TSP of the graph of figure 1.1.

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 procedures 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 $n \log n$.

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 "efficient" 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 complete 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 not even one 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 should not 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 even larger 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].)

1.2 Remedies for NP-Completeness

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 dissertation. 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 $n \log n$. 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.

The reader of this thesis will have an opportunity to encounter several other problems exhibiting a special structure which can be (and has been) exploited. It will be seen, to state an example, that in some sequencing problems the very fact that we can "classify" the nodes of a graph into distinct "categories" can result in drastic savings in computational effort as compared to using a general solution algorithm. The details on how this (and other savings) can be accomplished will be presented in the appropriate parts of this dissertation.

For the moment we contend ourselves in noting that in every problem which displays a special structure there exists some potential for improvement. The exploitation of this potential can lead to substantial increases in the efficiency of the solution of the problem.

CHAPTER 2

THE DYNAMIC PROGRAMMING SOLUTION TO THE TRAVELLING SALESMAN PROBLEM

2.0 Introduction

It will be seen that the sequencing and routing problems which will be examined in subsequent parts of this dissertation are 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 all of these problems.

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.)

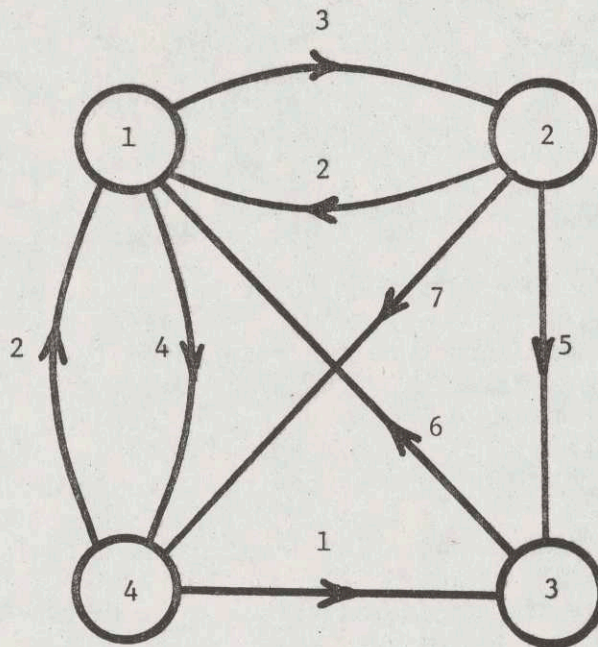
Weighted means that every arc (i,j) is assigned a number, called interchangeably distance, separation or cost, d_{ij} . It is not necessary that the graph is symmetric i.e. that $d_{ij} = d_{ji}$. Also, the cost to go from a node i to another node j of the graph, will depend on what other intermediate 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 $N \times N$ 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} is 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 $\{L_1, L_2, \dots, L_N, L_{N+1}\}$ 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 $L_1 = L_{N+1} = \text{node } 1$.

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, \dots, 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 $V(L, k_1, \dots, k_N)$ as the minimum achievable cost to return to node 1 passing through all unvisited nodes, given our current state is (L, k_1, \dots, k_N)

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

$$V(L, k_1, \dots, k_N) = \min_{x \in X} [d_{L,x} + V(x, k'_1, \dots, k'_N)] \quad (2.1)$$

where

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

and

$$k'_j = \begin{cases} k_j - 1 & \text{if } j = x \\ k_j & \text{otherwise} \end{cases} \quad (2.3)$$

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, \dots, k_N) . Two facts are clear:

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

$$\text{So } V(1, 0, \dots, 0) = 0. \quad (2.4)$$

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

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

*A vector (a_1, \dots, a_N) is said to be lexicographically greater than another vector (b_1, \dots, 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, \dots, 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 $(1, 1, \dots, 1)$ has been reached.

We know that we are initially at node 1 and our state is $(1, 1, \dots, 1)$. The best next node is given by the value of NEXT $(1, 1, \dots, 1)$. Supposing for the sake of argument that this is equal to 3, this means that our state becomes now $(3, 1, 1, 0, 1, \dots, 1)$. 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 $(1, 0, 0, \dots, 0)$.

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 $N \cdot 2^N$. 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 able 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 parts of this dissertation. In fact it will turn out that this approach, in the form of more sophisticated algorithms, will prove itself useful in tackling the kinds of problems to be examined later.

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 will be studied in detail in Part III. 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 ingeniously 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 ingenious 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 in subsequent parts.

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 problems 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 thesis.

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 coalitudinal 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 importantly, 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 dissertation. 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 they 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 IV of the thesis.

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

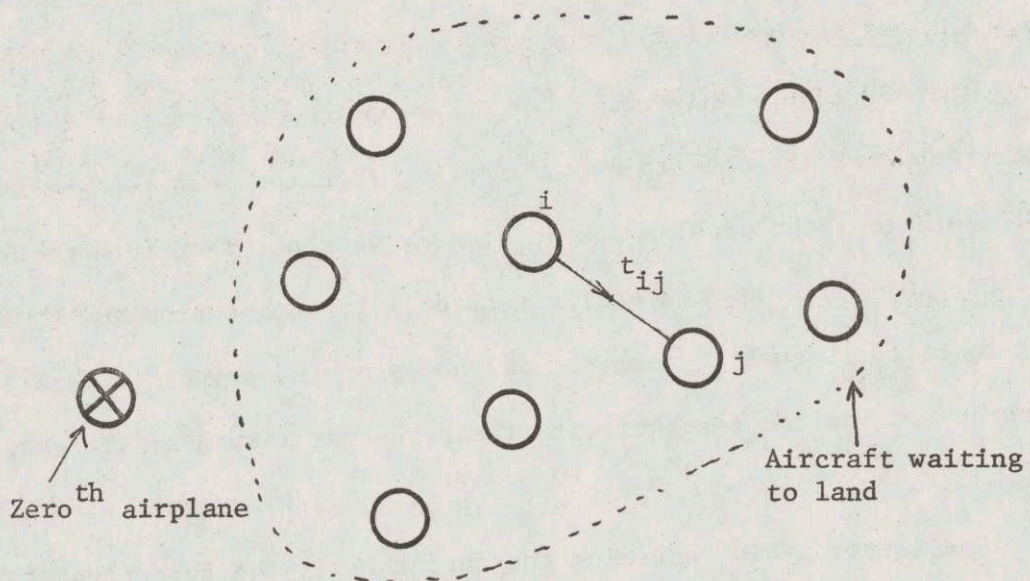


Fig. 3.1 : Graph representation of the ASP.

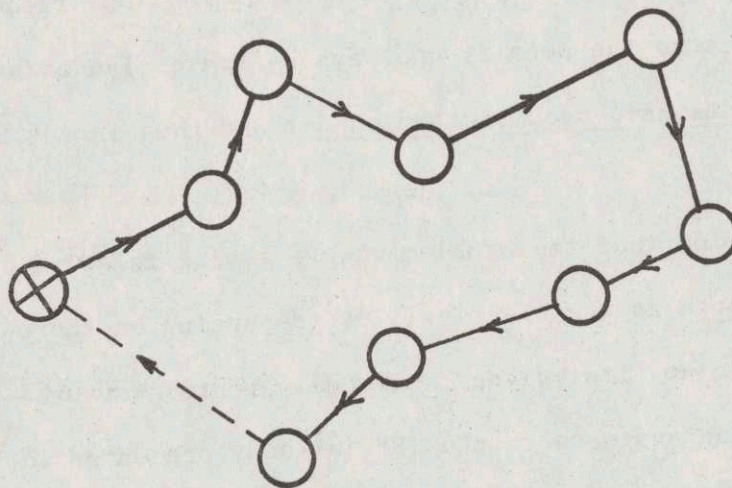


Fig. 3.2 : A sequence of aircraft landings. The dotted line is the zero-cost return arc.

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):

	Category	Types of aircraft included	# of passengers	Landing Velocity (Knots)
1	Wide-body (heavy) jets	B747, DC-10, L-1011 Airbus	250-450	140-160
2	Conventional large jets	B707, DC-8	150-200	140-160
3	Medium-Size jets	B727, B737, DC-9	100-160	120-140
4	Large props, turbo-props and business jets	DC-6, DC-3, Electra	10-100	80-110
5	Small pri- vate props	Piper Cub, Cessna	2-10	60-90

The last two categories include general aviation aircraft, which will not usually be assigned to land on the same runway as the larger commercial aircraft at major airports. Other categories (e.g. SST's) could be devised as necessary.

In any event, the main fact is that the number of categories is relatively small while the number of aircraft per category may be substantially large.

How does this observation affect our problem?

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

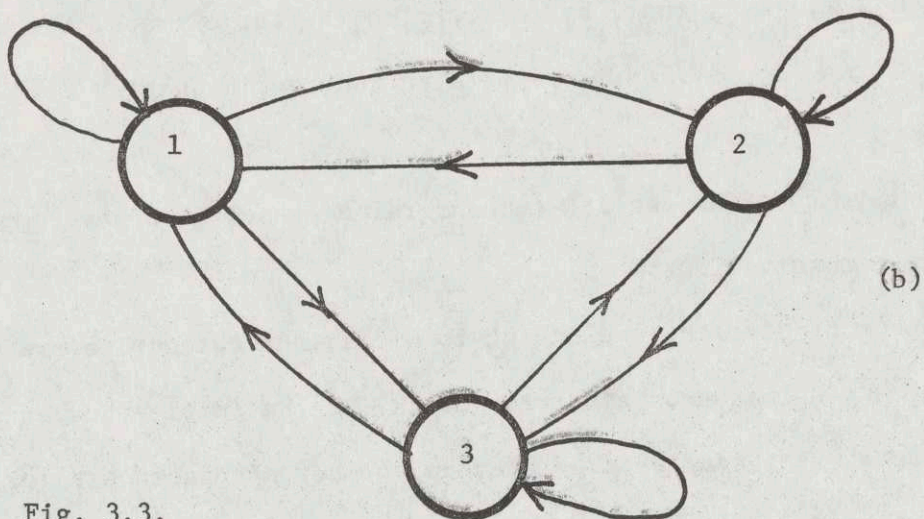
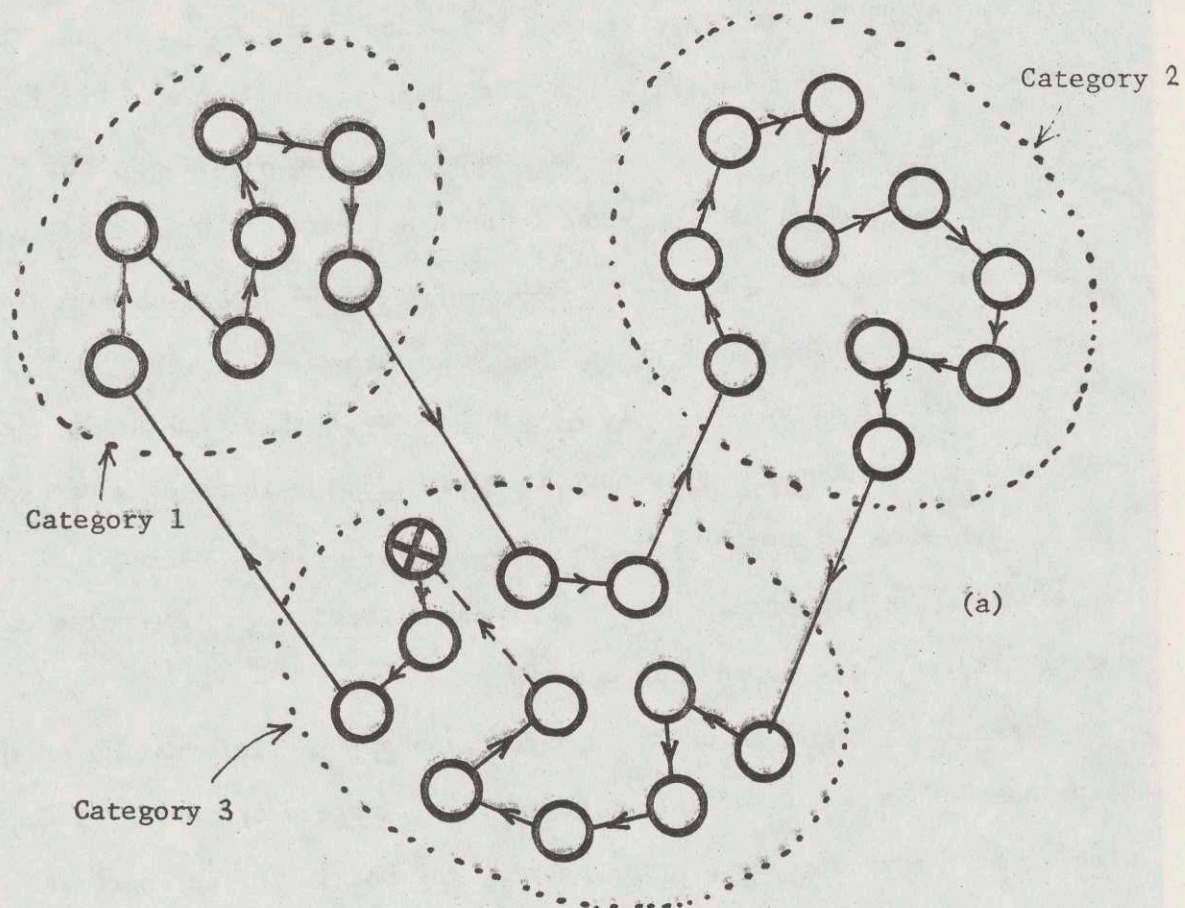


Fig. 3.3.

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. $L \in \{1, \dots, N\}$.
- 2) k_1, \dots, 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, \dots, k'_N) immediately follows the state (L, k_1, \dots, k_N) , then

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

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

4) Optimality recursions: We define $V_Z(L, k_1, \dots, k_N)$ as the optimal value of all subsequent decisions that can be taken from the current state (L, k_1, \dots, k_N) , till the end of the sequencing procedure ($k_1 = \dots = k_N = 0$). 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:

$$V_Z(L, k_1, \dots, k_N) = \min_{x \in X} [W_Z \cdot t_{L,x} + V_Z(x, k'_1, \dots, k'_N)] \quad (4.1)$$

$$\text{where } X = \{y: 1 \leq y \leq N, k_y > 0\} \quad (4.2)$$

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

$$\text{and } k'_j = \begin{cases} k_j - 1 & \text{if } j=x \\ k_j & \text{otherwise} \end{cases} \quad (j=1, \dots, N) \quad (4.4)$$

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

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

By definition it is clear that

$$\text{NEXT}_Z(L, k_1, \dots, k_N) = x, \text{ if}$$

$$V_Z(L, k_1, \dots, k_N) = W_Z \cdot t_{L,x} + V_Z(x, k'_1, \dots, k'_N) \quad (4.6)$$

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 composition of our aircraft "reservoir" (k_1^0, \dots, k_N^0) and on the zeroth landed plane i_0 . Obviously, the state $(i_0, k_1^0, \dots, k_N^0)$ 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 $(i_0, k_1^0, \dots, k_N^0)$. 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 $(k_1^{\max}, \dots, k_N^{\max})$ on the values of (k_1^0, \dots, 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 $k_1 = \dots = k_N = 0$, 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 $j=1, \dots, N$. For each combination (k_1, \dots, 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 $(i_0, k_1^0, \dots, k_N^0)$. 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:

Step 0: (Initialization)

$m=0$

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

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

Step 1: (Termination check)

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

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

Otherwise continue

STEP 2: (Move to best next)

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

$$V_Z(L_{m+1}, k_1, \dots, k_N) = \min_{x=1, \dots, N} [V_Z(x, k_1, \dots, k_N)]$$

Otherwise $L_{m+1} = \text{NEXT}_Z(L_m, k_1, \dots, k_N)$.

Step 3: (Update)

For all $j=1, \dots, N$, 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, \dots, 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 shall have an opportunity to study similar and more complicated issues in detail in Chapter 10 of this thesis. For the moment 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 $C' = kN \cdot 2^{kN}$. Defining as $r = \frac{C'}{C}$, we see that $r = k \left[\frac{2^k}{k+1} \right]^N$ 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 ($=5 \times 150 + 5 \times 120 + 5 \times 100$), 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.

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

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:

$$\text{Case 3: } (i_0, k_1^0, k_2^0, k_3^0) = (1, 2, 4, 3) \quad Z=1$$

$$\text{Case 4: } (i_0, k_1^0, k_2^0, k_3^0) = (1, 2, 4, 3) \quad Z=2$$

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.

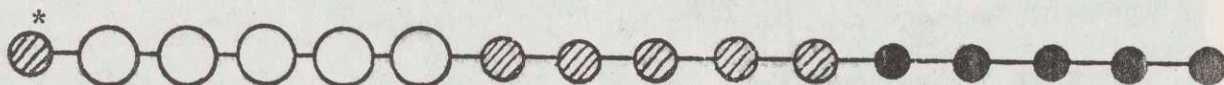


Fig. 4.2 : Case 2.

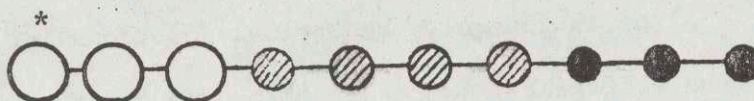


Fig. 4.3 : Cases 3 and 4.

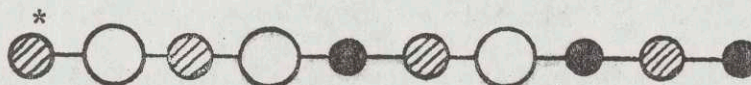


Fig. 4.4 : Case 5.

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: Could it be that this problem is so structured, that the optimal sequence of landings always involves category-clustering? 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:

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

Case 5: $(i_0, k_1^0, k_2^0, k_3^0) = (2, 3, 3, 3) \quad Z=2$

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:

$$\text{Case 6: } (i_0, k_1^0, k_2^0, k_3^0) = (2, 2, 5, 5) \quad Z=2$$

$$\text{Case 7: } (i_0, k_1^0, k_2^0, k_3^0) = (2, 1, 5, 5) \quad Z=2$$

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 being inserted 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 at the head 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:

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

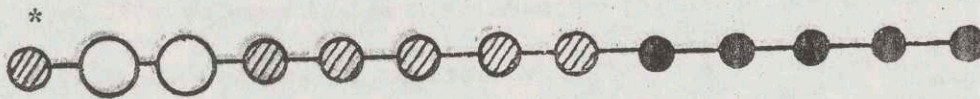


Fig. 4.5 : Case 6.

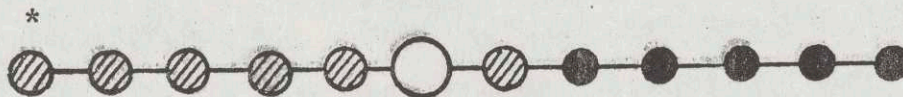
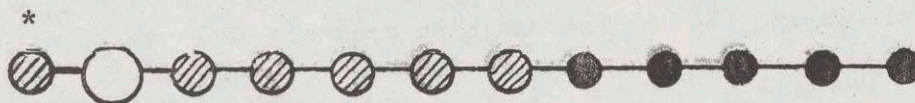
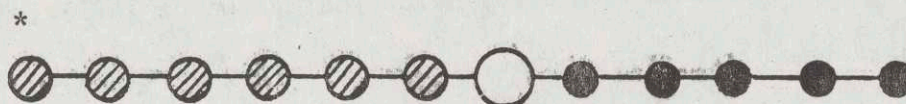


Fig. 4.6 : Case 7.



(a)



(b)

Fig. 4.7.

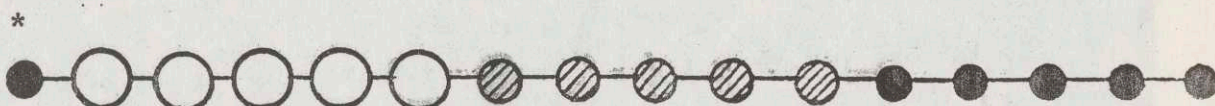


Fig. 4.8 : Case 8.



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 thesis.

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 any 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 12th position in the initial string of arriving aircraft, can land potentially anywhere from the 9th to the 15th 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 "static", 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 i^{th} position in the initial queue is guaranteed to land occupying a position in the landing sequence which falls between $i-\text{MPS}$ and $i+\text{MPS}$. 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 10.

2) Clearly, $\text{MPS}=0$ 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 $\text{MPS} \geq T-1$ 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 $v(\text{MPS})$ 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:

$$v(MPS_1) \geq v(MPS_2)$$

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

$$v(MPS) \geq v(\infty) \quad \text{for } MPS < T-1$$

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

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

$$v(MPS) = v(\infty)$$

then we shall also have

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

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

$$v(MPS_1) = v(MPS_2)$$

then we shall also have:

$$v(MPS_1) = v(MPS) = v(MPS_2) \text{ for all values of MPS between } MPS_1 \text{ and } MPS_2.$$

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, \dots, i_T) where, again, i_0 is the zeroth-landed category and category i_j holds the j^{th} position in the initial sequence.

$i_j \in \{1, 2, \dots, N\}$ for $j=0, 1, \dots, T$.

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 4th 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, \dots, k_N) as they were defined in the previous Chapter, then we know not only how many, but also specifically which airplanes per category have landed so far. This means that the state representation (L, k_1, \dots, k_N) used in the unconstrained case is sufficient 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 reduce the feasible state space, namely there may be state configurations which cannot be feasible. As an example of an infeasible configuration, we consider $(L, k_1, k_2) = (1, 1, 2)$ 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:

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

We can see that the last landed airplane (\$) has landed third while its initial position was fifth. 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 sequence ($j=1, \dots, N$).
- (ii) Let $s_j \equiv k_j^{\max} - k_j$. This is the number of airplanes belonging to category j which have landed, given that k_j airplanes of this category have not landed ($j=1, \dots, N$).
- (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, \dots, 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 necessary condition for (L, k_1, \dots, 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 not sufficient 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.

Nevertheless, 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, \dots, k_N) is feasible with respect to an initial sequence and an integer MPS, if and only if both (A) and (B) are true:

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

(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, \dots, k'_N) is feasible, where:

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

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, \dots, k_N) is infeasible, then we do not have to execute the recursion at all. If (L, k_1, \dots, k_N) is feasible, then the recursion is:

$$V_Z(L, k_1, \dots, k_N) = \min_{x \in X_1} [W_Z \cdot t_{L,x} + V_Z(x, k'_1, \dots, k'_N)]$$

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, \dots, k'_N) is feasible. If $k_1 = \dots = k_N = 0$, there are no "next" states and $V_Z(L, k_1, \dots, k_N) = 0$ (Provided of course (L, k_1, \dots, 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, \dots, 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, \dots, k_N) is feasible or not. We do this by

using backward recursion, starting from $(L, 0, 0, \dots, 0)$ where only (5.1) is needed and proceeding lexicographically up to $(L, k_1^{\max}, \dots, k_N^{\max})$ for $L=1, \dots, N$ 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 need not be separate but may as well be merged 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 feasible combinations (x, k_1, \dots, 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 procedures. 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 $FEASBL(L, k_1, \dots, k_N)$ with values "true" or "false" depending on whether (L, k_1, \dots, k_N) is feasible or not. The size of this array is $C = N \cdot \prod_{j=1}^N (k_j^{\max} + 1)$ 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 and 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 thesis. 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 only one sequence will be feasible, the initial FCFS sequence, itself: (i_0, i_1, \dots, 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 $V_Z(L, k_1, \dots, k_N) = -1$ 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:

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

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:

$$(P_1, P_2, P_3) = (300, 150, 100)$$

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).

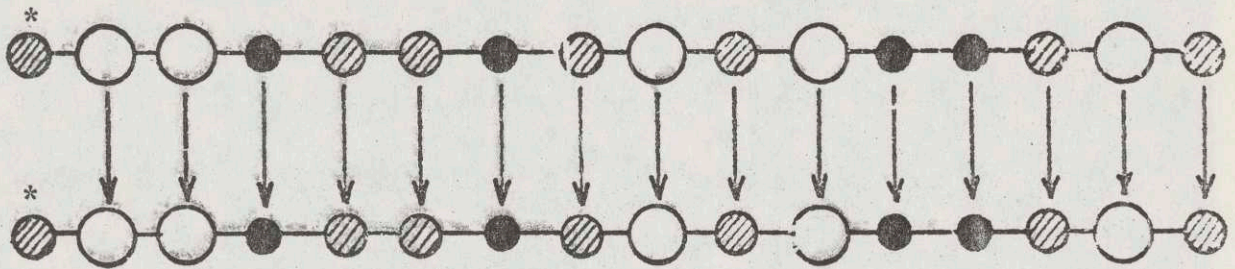


Fig. 5.1 : Case 1. In figures 5.1 through 5.5 the top sequence is the initial FCFS sequence and the bottom one is the optimal sequence. Arrows depict the position shifts of the various airplanes.

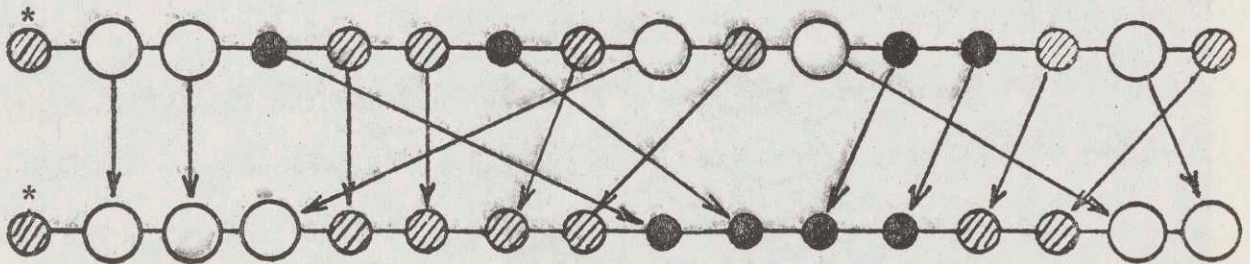


Fig. 5.2 : Case 2.

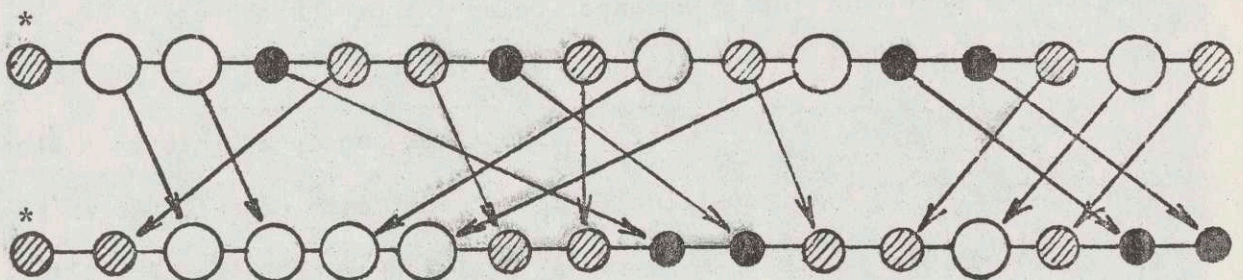


FIG. 5.3 : Case 3.

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. ($MPS=\infty$.)

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.

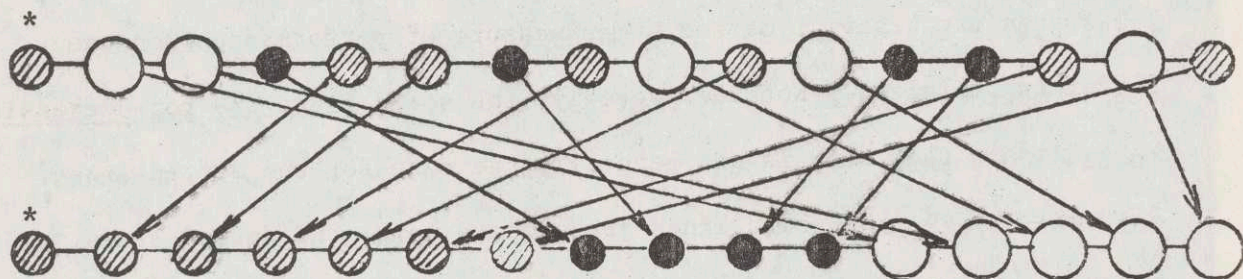


Fig. 5.4 : Case 4.

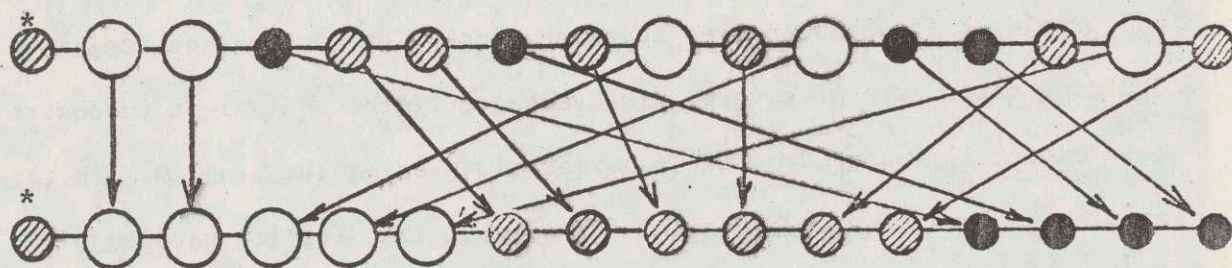


Fig. 5.5 : Case 5.

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 10). 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.

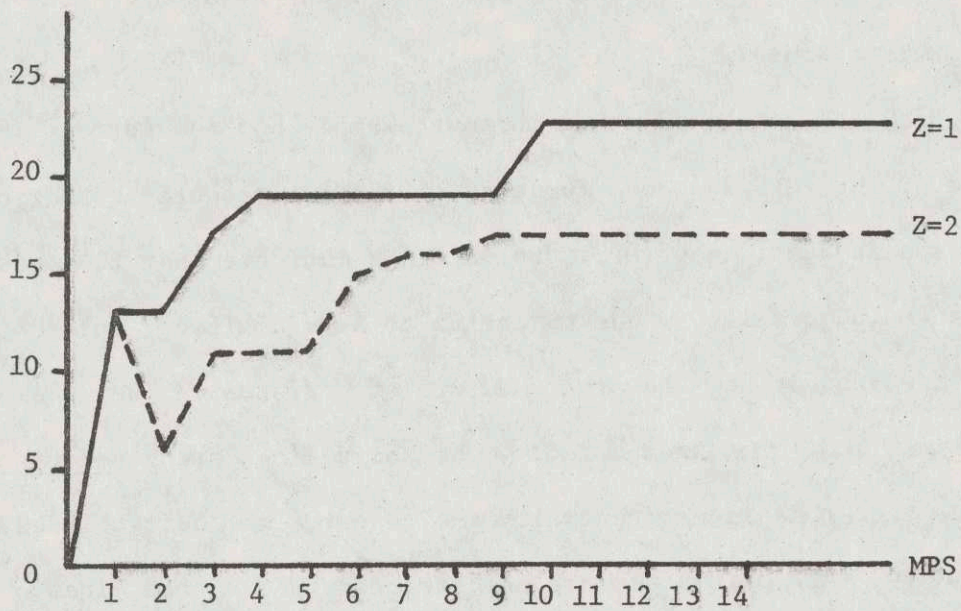


Fig. 5.6 : % improvement in LLT with respect to the FCFS discipline.

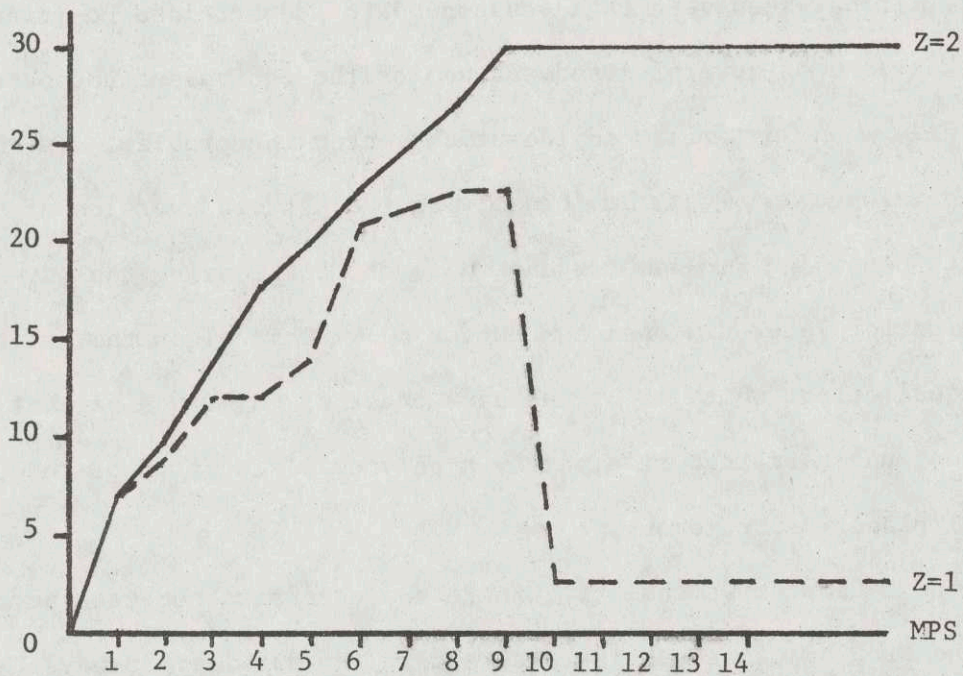


Fig. 5.7 : % improvement in TPD with respect to the FCFS discipline.

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.

$(x_1^*, \dots, x_N^*) / (k_1^0 - x_1^*, \dots, k_N^0 - x_N^*)$ which is optimal.

To evaluate a particular partition $(x_1, \dots, x_N) / (k_1^0 - x_1, \dots, k_N^0 - x_N)$ 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 $\vec{x} = (x_1, \dots, x_N)$ and the objective Z , is given by:

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

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

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

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, after we have decided on the partition.

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

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 all possible partitions (i.e., all combinations of (x_1, \dots, x_N) with each x_j between 0 and k_j^0) corresponding to the initial composition (k_1^0, \dots, 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 $V_Z(L, k_1, \dots, k_N)$ and $NEXT_Z(L, k_1, \dots, k_N)$ 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, \dots, 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 $\vec{x} = (x_1, \dots, x_N)$ with $0 \leq x_j \leq k_j^0$ and select the one which minimizes the quantity:

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

Let $\vec{x}^* = (x_1^*, \dots, x_N^*)$ 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 $(i_{01}, x_1^*, \dots, x_N^*)$ and one for Runway 2 with initial state $(i_{02}, k_1^0 - x_1^*, \dots, k_N^0 - x_N^*)$.

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 $(i_{02}^0, k_1^* - x_1^*, k_2^0 - x_2^*) = (2, 2, 0)$ 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 $(i_{01}^*, i_{02}^0, k_1^0 - x_1^*, k_2^0 - x_2^*, k_3^0 - x_3^*)$ as defined earlier. A parameter for each run is our objective Z , as also defined earlier.

The outputs for each run are:

- 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.
- The optimal sequences for each of the two runways. These will be depicted in accompanying figures.
- 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:

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

$$C = C_1 + C_2$$

$$\text{and } \frac{C}{t} = \frac{C_1}{t_1} + \frac{C_2}{t_2}$$

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)

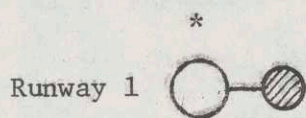


Fig. 6.1.

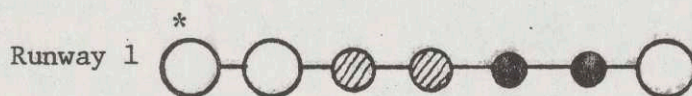
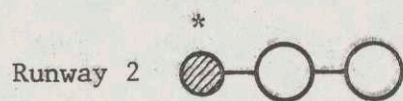


Fig. 6.2 : Case 1.

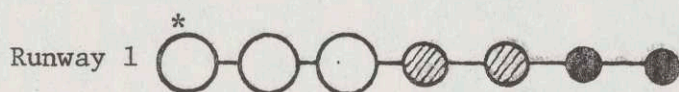
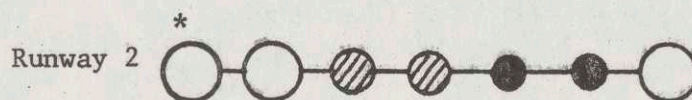
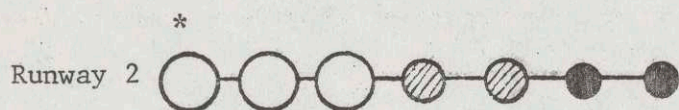


Fig. 6.3 : Case 2.



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:

$t = 660$

$t = 666,600$

$\bar{t} = 303$

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 examining 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:

$t = 402$

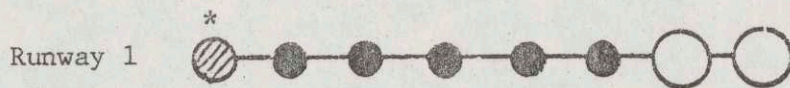


Fig. 6.4 : Case 3.

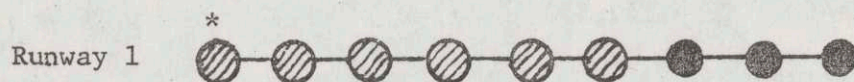
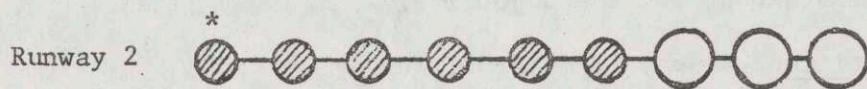


Fig. 6.5 : Case 4.

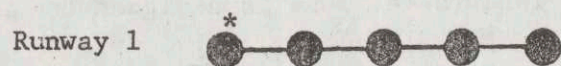
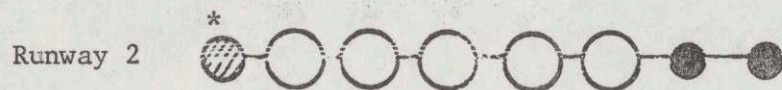


Fig. 6.6 : Case 5.

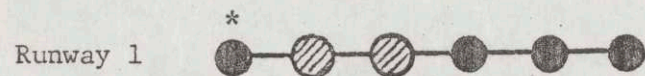
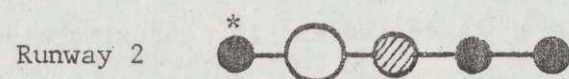


Fig. 6.7 : Case 6.



$$C = 332,100$$

$$\bar{t} = 266$$

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:

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

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

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 three (or more) independent runways?
- c) What happens if the problem becomes "dynamic"?

We shall discuss such issues in Part IV of this dissertation. 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

THE DIAL-A-RIDE PROBLEM

CHAPTER 7

DIAL-A-RIDE: PROBLEM DESCRIPTION

7.0 Introduction

The purpose of this first chapter of Part III of the thesis is to introduce the reader to the dial-a-ride problem and provide the necessary background for our subsequent investigation of the problem.

Dial-A-Ride involves the dispatching of a fleet of vehicles to satisfy demands from customers who call a vehicle operating agency requesting service, requesting that is, to be carried from specified points to other similarly specified points. We shall examine several versions of this service later.

Such demand-responsive transportation systems have been in operation in several metropolitan and suburban areas of the U.S. during recent years. Perhaps the most important example of such a system is the one providing service to several suburbs of Rochester, N.Y. This system was inaugurated in 1973. The planning, research and management of the project were assigned to M.I.T., under contract from the Rochester-Genessee Regional Transportation Authority (R-GRTA). The system evolved into a complex, computer-assisted vehicle-dispatching operation. Its total research budget has been of the order of \$3.6 million, provided by the Urban Mass Transit Administration (UMTA). Details of the Rochester dial-a-ride system are described in a series of M.I.T. Reports, such as [WILS 77a]. In addition to Rochester, there are several similar systems in operation in other

areas of the U.S., as well as abroad today (e.g. Ann Arbor, Mich.; San Jose, Cal.; Tokyo, Japan).

Several versions of dial-a-ride service exist today, giving rise to several types of what we shall subsequently call the "dial-a-ride problem." We shall discuss below some of these types:

1) A usual classification is to divide these problems into the so-called "many-to-one" and "many-to-many" versions. "Many-to-one" involves the pickup of a number of customers from distinct points and the subsequent delivery of them to one common destination. The operation of a "feeder" service, carrying passengers to a bus terminal or airport is a typical example of a "many-to-one" configuration. The reverse situation is obviously an entirely similar problem ("one-to-many")*. By contrast, the problem is characterized as "many-to-many," if some, or all customers have their own distinct origin and destination points. It is clear that the "many-to-one" problem can be considered as a special case of the "many-to-many" problem and thus it can be at most as difficult as the latter.

2) A classification according to another criterion takes into account the time for which the service is requested by the customers. Thus, a customer may request immediate or advance-request service. Under the first scheme, the customer requests to be serviced as soon as possible. Under the second, he or she wishes to be serviced at a specified time in the future (e.g. "at 10 a.m. next day," or "no later than 5 p.m.," or "not earlier than 7 p.m.," etc.). Since an immediate-request can be viewed as an advance-request for which the earliest pick-up time is the instant

*Massport's Share-A-Cab system is a good example of a "one-to-many" configuration.

when the request is made, it follows that the immediate-request version of the problem is a subproblem to the advance-request version and thus can be at most as difficult as the latter. An alternative name for advance-request service is "subscription service," implying that customers who are to be serviced at some time in the future are considered subscribers to a certain list (subscription list).

3) According to the number of vehicles involved, our problem may be single-vehicle or multi-vehicle. It is clear that the multi-vehicle problem should be at least as difficult as the single-vehicle one, for an additional decision to be made if many vehicles are present involves the assignment of the various vehicles to the set of customers requesting service.

4) A final classification of the problem is along the lines of "static" vs. "dynamic". In the "static" case, the problem inputs (such as the number of customer and their geographical locations) do not change during the execution of the vehicle route. This means that if a new customer requests service during the time a vehicle is executing its assigned route, this new input to the problem is not taken into consideration until the completion of the current route, even if this current route passes "very near" the origin or destination points of that customer. By contrast, in the "dynamic" version of the problem, this new input is taken into account as part of the problem at its time of appearance. An "updated" problem is solved at that time (or as soon as possible) and in general whenever a new input appears. It is clear therefore that the "dynamic" version of the problem, in addition to being closer to reality than the "static" one, may theoretically never "terminate," being by nature open-ended.

7.1 Measures of Performance and Constraints

So far our description of dial-a-ride as a problem has been incomplete, because we have not mentioned anything about objectives or constraints.

Several forms of objective functions may be considered for this problem. In [WILS 77b], Wilson discusses the concept of user disutility, namely the degree of dissatisfaction caused to all customers of the system. According to this report, the important factors influencing user disutility which are under the control of the scheduling algorithm are:

- 1) Wait time: The time between a request for service being made and the corresponding pick-up.
- 2) Ride time: The time between a customer's pick-up and delivery.
- 3) Pick-up deviation: The time between a customer's promised pick-up time and the actual pick-up time.

(A fourth factor, delivery deviation, might be defined if necessary.)

The assumed disutility function for every customer is a weighted sum of the squares of the above defined three terms.

In the same report, mention of a so-called marginal system resources term is also made. This term reflects the impact of a new customer-to-vehicle assignment on the system's ability to serve future customers. It is argued that this term is a quadratic function of the tour lengths for the vehicle being considered for assignment before and after a new customer requests service and of the mean tour length for all active vehicles.

We can see that the objective function for this version of the dial-a-ride problem is quadratic.

Alternatively, one may assume an objective function of linear form.

This will make sense if our goal is to minimize total distance travelled, an objective identical to that of the classical Travelling Salesman Problem (Chapter 2). A linear objective function will also make sense if we assume that the disutility function of each customer is itself linear and our goal is to minimize total passenger disutility. In fact, linearity in the objective function will be one of our assumptions as we shall see later.

As far as the problem constraints are concerned, these also depend in general on the version of the problem examined.

A constraint which is likely to be encountered in all versions of the problem concerns vehicle capacity. Thus, vehicle routes resulting in the vehicle carrying more customers than its capacity, are to be excluded from consideration. The degree to which this constraint is likely to be binding for the problem, depends on the vehicle size and the demand for service. (In the Rochester system, for example, the vehicle fleet included several 25-passenger buses, several 17-passenger buses, etc. It is conceivable however that a system may include vehicles of even smaller capacity, such as cab-like vehicles of 3-4 passenger capacity.)

Another constraint which exists in all many-to-many problems, is that of route legitimacy: The obvious observation that each customer has to be picked up before he is delivered constitutes an important constraint, concerning which routes are eligible for consideration.

The existence of other constraints depends upon the particular version of the problem examined.

For example, in the advance-request version, time constraints may be imposed. Thus, if a passenger requests to be picked up not later than a

specified time, an appropriate upper bound constraint on his time of pick-up is in effect. Corresponding lower bound constraints may also be considered. In addition, it is conceivable that time constraints may exist in the immediate-request version as well: The vehicle operating agency may for example have a policy of guaranteeing to every customer requesting immediate service that he will be serviced not later than a prescribed time interval from the time of call*.

We shall have the opportunity to discuss in detail a "similar" type of constraint when developing our version of the dial-a-ride problem. The "similarity" is only with respect to the concept of a guarantee. We shall see that this guarantee does not explicitly involve time but rather the priority of each customer.

7.2 Existing solution procedures.

Solution procedures for the dial-a-ride problem depend in general on the particular version examined. It is important however, before summarizing the existing algorithms for this problem, to make an observation regarding the problem's nature.

Perhaps the most important observation we can make in that respect is that the simplest version of the problem cannot be easier to solve than the Travelling Salesman Problem. In fact, the many-to-one single vehicle "static" dial-a-ride problem where the objective is to minimize total distance travelled and no capacity or time constraints are imposed, is exactly equivalent to a Travelling Salesman Problem. But we already have seen in Chapters 1 and 2 that the TSP belongs to a class of problems which are

*Or, alternatively, if this is not possible, notifying the customer accordingly.

particularly difficult to solve, the class of NP-complete problems. Thus, any version of the dial-a-ride problem, being at least as difficult as the version we just mentioned (e.g. more vehicles, many-to-many, "dynamic," quadratic objectives, capacity & time constraints, etc.) is bound to carry with it all the difficulty of the TSP and quite likely on a much larger scale. However disappointing this observation may be, it is at least comforting to have realized a priori the degree of difficulty of this problem and thus not be particularly optimistic about the potential for "efficient" solution procedures.

As a result of the Rochester project, a considerable effort was made at M.I.T. in the direction of algorithm development for the dial-a-ride problem. Several reports have been published in that respect [WILS 76, WILS 77a, WILS 77b]. It is not our intention to cover here all the details of these algorithms. However, we can briefly describe their basic philosophy as follows:

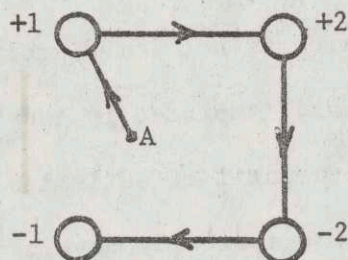
The version of the problem considered is a multi-vehicle, many-to-many "dynamic" dial-a-ride problem with a quadratic objective function and time constraints (existence of advance-requests).

The first generation algorithm [WILS 76] has been called "immediate assignment algorithm" because, regardless of customer request type (i.e. immediate or advance), the algorithm makes the assignment regarding what vehicle will service a particular customer, immediately upon being requested to do so*. The assignment decision is made using a branch-and bound

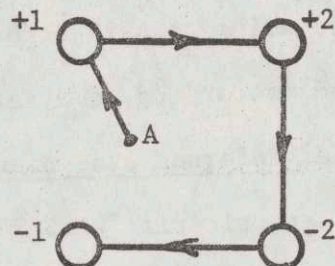
*This is done immediately for immediate customers and at a prescribed time before the desired pick-up time for advance customers. In the second generation algorithm we shall see that the assignment decision may be deferred according to a set of rules [WILS 77b].

technique. Thus, at any point in time, each vehicle has a set of passengers already assigned to it and a corresponding ordered sequence of pick-ups and deliveries. If no new passengers appear, each vehicle completes its route as scheduled by the algorithm. When a new customer appears, the algorithm evaluates for each vehicle all the possible insertions of pick-up and delivery stops into all available existing vehicle routes and subsequently selects the best of these insertions for each vehicle. An assignment decision then determines the best vehicle to service this new customer. It is important to note at this point that this algorithm does not have the capability to reconsider its past decisions. In other words, even if an ordered sequence of pick-ups and deliveries is optimal at a certain point in time, the inclusion of a new pair of stops may make the above order sub-optimal. An illustrative example of this is depicted in Fig. 7.1. It is also conceivable that the assignment of a certain customer to a particular vehicle at a certain point in time may not remain the best possible assignment after a new set of customers have appeared, in the sense that under this new input, it would be better to assign that particular customer to another vehicle. Thus, in both these respects (routing of each individual vehicle and assignment of customers to vehicles) this algorithm cannot reconsider "locally optimal" decisions made in prior times and yields therefore suboptimal solutions.

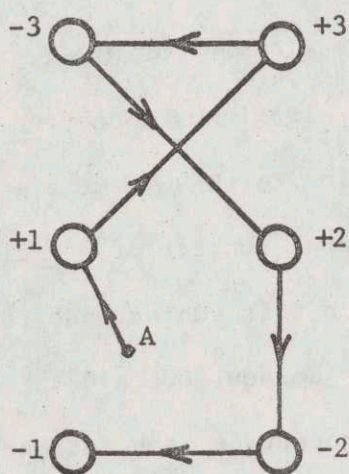
As an alternative to the above re-optimization capability (which was recognized to represent a difficult combinatorial problem), the concept of deferred assignment was introduced [WILS 77b]. According to this, the algorithm need not assign a demand immediately upon receiving its request for service but may instead defer a decision concerning it for a



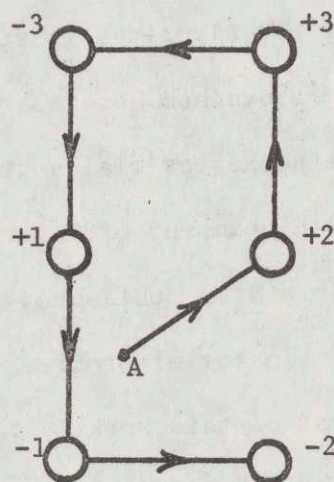
(a)



(b)



(c)



(d)

Fig. 7.1 : In this figure, as well as in the rest of Part III, pick-up points are depicted by a "+" and delivery points by a "-". Here the vehicle starts from point A and the objective is to minimize the total distance travelled up to the delivery of the last customer. The space is assumed Euclidean.

- (a) Optimal route for customers 1 and 2.
- (b) While vehicle is still at A, customer 3 requests service.
- (c) Best of all insertions of pick-up and delivery stops of customer 3 into existing optimal route. This solution is sub-optimal.
- (d) Optimal route for customers 1, 2 and 3.

finite period of time, in the hope that, in the meanwhile, additional information will be received which will enable the algorithm to make a better decision. Considerations like the above led to the proposal of a second generation algorithm.

According to this algorithm, every demand which is not immediately assigned (according to a set of heuristic criteria), is placed in the so called deferred assignment pool, remaining there until it satisfies some other criteria [WILS 77b]. These criteria were specially developed so that all customers for whom service is deferred are in fact serviced "satisfactorily" in the sense that indefinite deferment is not encouraged by the procedure. The second generation algorithm treats the first generation algorithm as a subroutine. It should be noted at this point that the new algorithm is still at the stage of development at M.I.T.

The above has been a brief discussion of the algorithms connected with the Rochester dial-a-ride project. In addition to this work, a considerable amount of literature on the more general subject of Vehicle Routing has been published. An excellent survey on this subject and related combinatorial problems has been published by Golden [GOLD 76]. The thrust of this work is focused on heuristic procedures, the rationale being that a "good" solution obtained easily is preferable to the exact optimal solution, if it is very difficult to obtain the latter.

While we agree with this basic premise, we nevertheless feel that the ability to obtain the exact solution of a "difficult" problem, despite the high price associated with it, may provide additional valuable knowledge about the structure of the problem (structure which may be difficult to expose through a heuristic) and thus lead to significant theoretical and

also practical results. This feeling was reinforced with regard to the specific problem of dial-a-ride by the fact that no research effort on this problem has been channeled in that direction so far.

It is in this spirit that this thesis will now proceed to a detailed discussion of an exact approach to the solution of a specific version of the dial-a-ride problem. Both the "static" and the "dynamic" problems will be addressed, the latter considered as an extension of the former. Our approach, based on dynamic programming, departs significantly from the philosophy used in the algorithms discussed above as well as from other heuristic approaches. Special priority constraints will be seen to take care of the possibility of indefinite deferment arising from the approach's capability to reconsider its past decisions. An extensive discussion of the combinatorics of the algorithm as well as of some aspects connected with the real-time operation of it will also be presented.

CHAPTER 8

DIAL-A-RIDE-"STATIC" CASE: DYNAMIC PROGRAMMING SOLUTION

8.0 Introduction

The "static" case we shall present in this Chapter is a major step toward the development of the equivalent "dynamic" case which will be studied subsequently.

The version of the dial-a-ride problem we shall examine is a single vehicle, immediate-request many-to-many problem, with a linear objective function, capacity constraints and some special priority rules. Specifically, the following clarifications are important:

We assume that at a specific point in time, which we shall denote by $t=0$, the vehicle becomes available. Customer requests prior to that time have been arranged in an ordered (according to some criterion) list. Each customer wishes to be picked-up as soon as possible from a prescribed point (origin) and delivered to another prescribed point (destination). We assume that each customer has his own distinct origin and destination. The location of the vehicle at $t=0$ is at Point A (Fig. 8.1), which may be a central depot or any other similarly specified point.

We require that the list into which the customer requests have been recorded prior to $t=0$, becomes "closed" at that time, in the sense that no new customer requests are appended to it during the execution of the route. We shall call this list the current list of the problem. The current task of the vehicle operator is to provide service to the customers that the

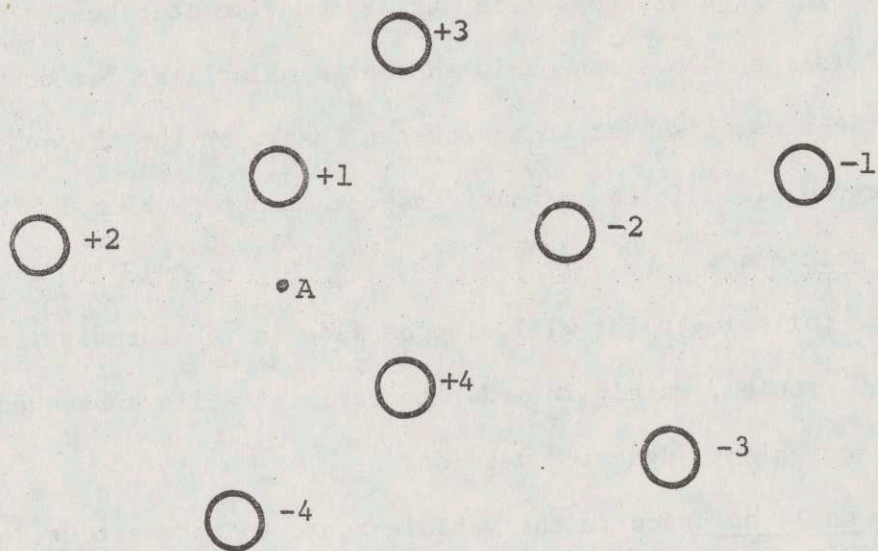


Fig. 8.1.

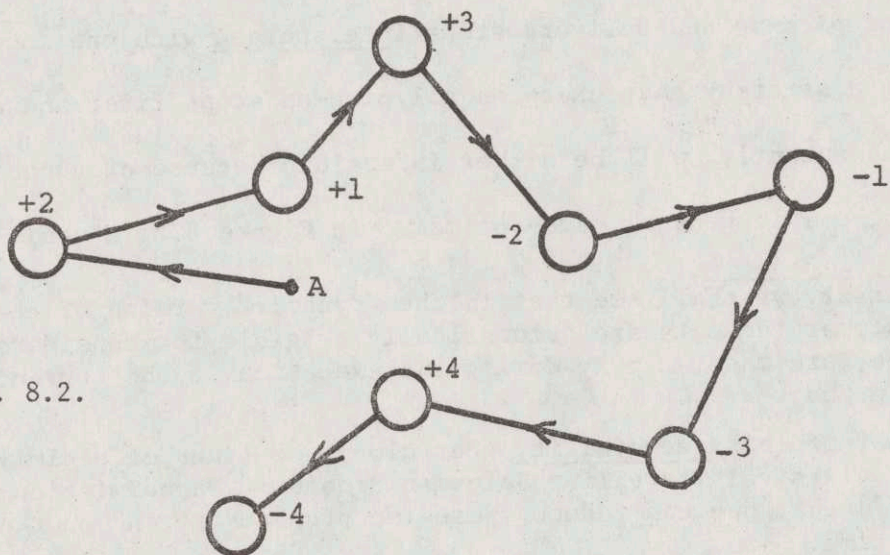


Fig. 8.2.

current list contains and only to those. This task can be considered accomplished when all customers on the current list have been serviced. New customer requests appearing during the time it takes to execute the current task can be arranged in another similar list for consideration after the accomplishment of the current task, or for the scheduling of another vehicle. It is in this sense, that our problem in this chapter is a static one*.

The following point will also be made in the formulation of the "static" problem, mainly in order to facilitate its subsequent extension to the equivalent "dynamic" version:

We shall not require the vehicle route to form a tour, ending at the starting point A (as the classical Travelling Salesman Problem requires). Instead, we shall require the route to form an (open) path, ending with the delivery of a (last) customer. This assumption is not binding on our formulation, as the "closed" and "open" route problems are reducible to one another**.

For N customers a vehicle route will consist of 2N "legs," a sequence of pick-up and delivery stops interspersed with one another. Clearly, a strategy that executes all pick-up stops first and all delivery stops subsequently, will be either infeasible because of capacity constraints, or, in general, sub-optimal. In Figure 8.2, a vehicle route

*By contrast, we shall see that in the "dynamic" version of the problem new customer requests are automatically eligible for consideration at the time they are made, namely during the execution of the current task. Details in Chapter 9.

**The proof of this reducibility goes along the lines of a similar proof for the classical Travelling Salesman Problem: Papadimitriou has shown [PAPA 77] that one can reduce these two problems to one another in linear time.

can be seen. Pick-up points will be marked from now on with a "+", delivery points with a "-". Numbers on the nodes of the graph serve for customer identification, according to their order in the current list.

We finally assume that the time it takes to go from any of the $2N+1$ points of our problem, i , directly to any other point, j , is a known and fixed quantity t_{ij} .

8.1 Discussion of alternative objectives.

For a particular vehicle route, let T_j be the duration of the j^{th} leg of the trip ($j=1, \dots, 2N$).

One may decide to judge the performance of a particular sequencing strategy by how fast the current task is accomplished. [Since the formulation of our problem does not include vehicle speed as a decision variable, an exactly equivalent measure of performance would be the total distance travelled by the vehicle till all customers are delivered, if the t_{ij} and T_j were given in distance units.]

Therefore, a reasonable objective for our problem is to minimize $\sum_{j=1}^{2N} T_j$
This is an equivalent objective to that of the classical TSP, although the structure of our problem is not the same. We shall index the above objective with $Z=1$.

Let now WT_i be the time interval customer i spends waiting till his pick-up and RT_i the time interval the same customer spends riding in the vehicle till his delivery ($i=1, \dots, N$). WT_i is measured from $t=0$ for all customers; the time interval elapsed from the instant customer i called for service till the vehicle became available at $t=0$ will be neglected*

*To neglect this quantity will not be in general justifiable. For example, it is not difficult to see that one cannot do this if the problem's objective is not linear, as ours is.

as a quantity that cannot be changed by any of our decisions after $t=0$ (sunk cost).

Alternatively to the minimization of the total distance travelled, one may decide to judge the performance of a particular sequencing strategy by the degree of dissatisfaction caused to customers while they are waiting to get to their destination. In this respect, we assume that each customer's dissatisfaction is reflected by a disutility function U_i , which, in our version, is a linear combination of WT_i and RT_i , of the following form:

$$U_i = \alpha.WT_i + (2-\alpha).RT_i \quad (i=1, \dots, N)$$

Here α is a constant between 0 and 2, assumed equal for all customers.

This constant is a parameter of our problem and reflects the customers' preference between waiting for the vehicle and riding in the vehicle.

Thus, if $\alpha=0$, then WT_i is irrelevant and all that counts for every customer is the time he spends in the vehicle. Similarly, if $\alpha=2$, then RT_i is irrelevant and all that counts for every customer is the time till the vehicle arrives to the pick-up stop. Intermediately, if $\alpha=1$, then $U_i = WT_i + RT_i$, namely the customer's dissatisfaction is reflected by the total time elapsed till his delivery, irrespectively of how this time has been spent (waiting for the vehicle or riding inside it). For psychological reasons, it is usually expected that $\alpha > 1$, although we shall not impose this restriction on our problem.

Thus, another reasonable objective for our problem is to minimize $\sum_{i=1}^N U_i$ where U_i is given by the suggested linear formula above. This calls for the minimization of the total degree of dissatisfaction caused to all

customers by the amounts of time they have to spend in order to get to their destinations. We shall index this latter objective with $Z=2$.

It is perhaps important to realize at this point that the two alternative objectives we have suggested so far will in general yield different optimal routes. In other words, these two objectives are indeed different and it is not possible to obtain the one by a suitable transformation upon the other. A very simple illustration of the possible conflict between these two objectives appears in Figure 8.3. The parameter α is equal to 1.

A final observation concerning the problem's objective is the following:

If one is willing to assign weights (or costs) to both the above measures of performance, it would be possible to bring them to a "common denominator." Thus a generalized objective would be to minimize a weighted combination of the above two measures of performance, in other words minimize $w_1 \cdot \sum_{j=1}^{2N} T_j + w_2 \cdot \sum_{i=1}^N (\alpha \cdot WT_i + (2-\alpha) \cdot RT_i)$ where w_1 and w_2 are the relative weights (or cost coefficients) for $Z=1$ and $Z=2$ respectively.

Since this new objective is a generalization of both $Z=1$ and $Z=2$ and thus incorporates these two objectives as special cases ($Z=1$ corresponds to $w_1=1, w_2=0$ and $Z=2$ to $w_1=0, w_2=1$) we shall work directly with it in our subsequent investigation of this problem, specializing to $Z=1$ and 2 only for illustration purposes.

8.2 Constraints

As we already have pointed out in the previous chapter, a constraint present in many-to-many problems is that of route legitimacy, namely the vehicle route must be such that the pick-up stop of every customer precedes

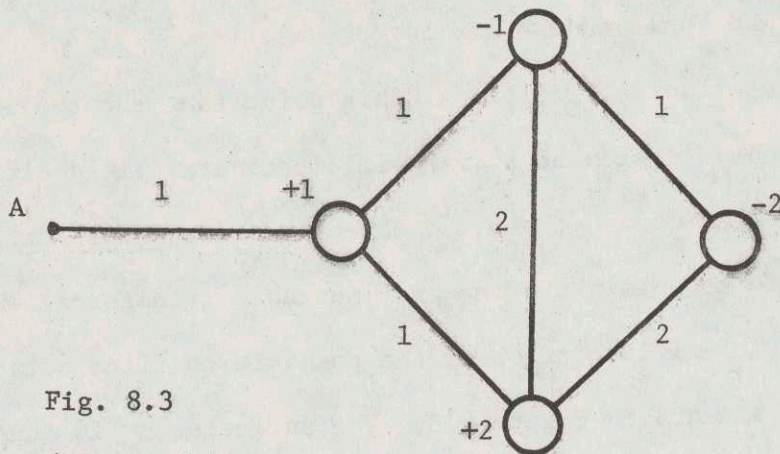
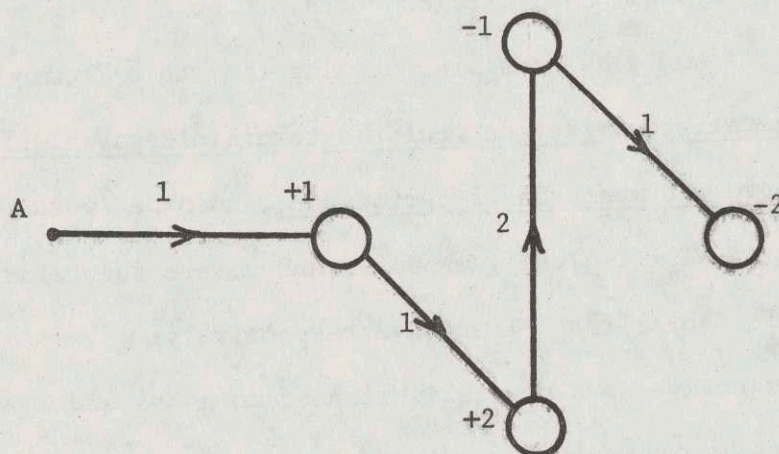


Fig. 8.3

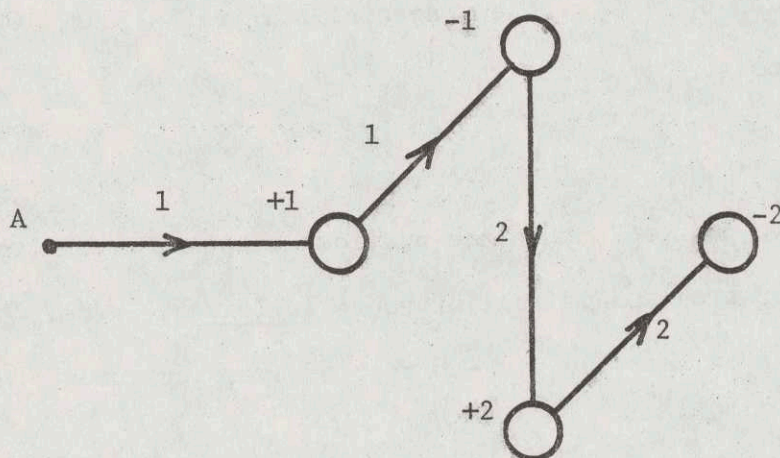
(Time and distance
units are the same).



Z=1

Total distance = 5

Total disutility= 9



Z=2

Total distance = 6

Total disutility= 8

his delivery stop.

Another constraint which we shall incorporate into our problem is that of vehicle capacity. According to that, the number of customers inside the vehicle at any point in time should not be greater than a specified number of persons, C.

We now come to the discussion of a special type of priority constraint which our problem will incorporate. The role of these constraints will be seen to be especially important for the "dynamic" version of the problem which we shall discuss in the next chapter. Nevertheless, these constraints will be incorporated in the "static" version as well, for motivation purposes and to facilitate the extension to the "dynamic" version. The following observations will explain the nature of the constraints.

For a particular vehicle route (which is feasible as far as the constraints of route legitimacy and vehicle capacity are concerned) the order among all N pick-up stops, in which a particular customer will actually be picked-up will not in general be the same as the position that particular customer held in the current list. Thus, for that particular customer, we shall observe a certain position shift concerning his pick-up. This shift (by definition) is positive if the pick-up order is less than the position in the list, negative if the opposite happens, or zero if these orders happen to be the same.

Similarly we shall, in general, notice a position shift with regard to the order of delivery of that customer, also with respect to his position in the current list. The shifts in the order of pick-up and in the order of delivery need not necessarily be equal.

Under our priority constraints in our version of the problem we shall

not allow either of the above two shifts to be greater in magnitude than a prescribed number MPS (for Maximum Position Shift), for all customers*.

Concerning the purpose of introducing this somehow peculiar type of constraint we can initially observe that these constraints can be considered as guarantees to customer i (this is the customer occupying the i^{th} position in the current list) that his positions in the sequence of pick-ups and of deliveries, although not necessarily equal to each other, will be between i -MPS and i +MPS. The following example will illustrate this guarantee concept:

Suppose that customers 1, 2, 3, 4, 5 and 6 have called, in that order, the vehicle agency for service. Suppose furthermore that MPS=2 and that we decide to service these customers in the following order:

Stop	Pick-ups	Deliveries
1	Pick-up 3	-
2	Pick-up 2	-
3	-	Deliver 2
4	Pick-up 1	-
5	-	Deliver 1
6	-	Deliver 3
7	Pick-up 6	-
8	Pick-up 4	-
9	-	Deliver 6
10	Pick-up 5	-
11	-	Deliver 5
12	-	Deliver 4

*The reader who is familiar with Part II of this Thesis (Aircraft Sequencing Problem) will undoubtedly recognize these new constraints as the Constrained Position Shifting rules introduced there. For the dial-a-ride problem, the formulation is slightly different because of the existence of two shifts instead of one.

We can check that the above order does satisfy the $MPS=2$ guarantee, by constructing the following table:

Customer	Position of Customer in Sequence of Pick-ups	Position of Customer in Sequence of Deliveries
1	3	2
2	2	1
3	1	3
4	5	6
5	6	5
6	4	4

We can then verify that, for every customer i , the above positions are between $i-MPS$ and $i+MPS$.

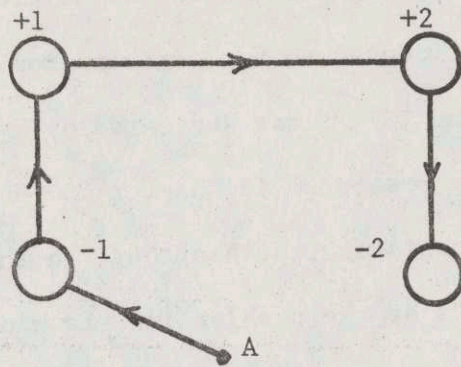
What is the significance of the MPS constraints?

At this point we should note that this guarantee does not explicitly involve the time dimension per se, by guaranteeing for example time bounds on the pick-up or delivery time for each customer. What is therefore the practical value (if any) of the above, somewhat vague and intangible guarantee to any particular customer? And assuming there is a value, how will this customer check that this guarantee has been kept for him*?

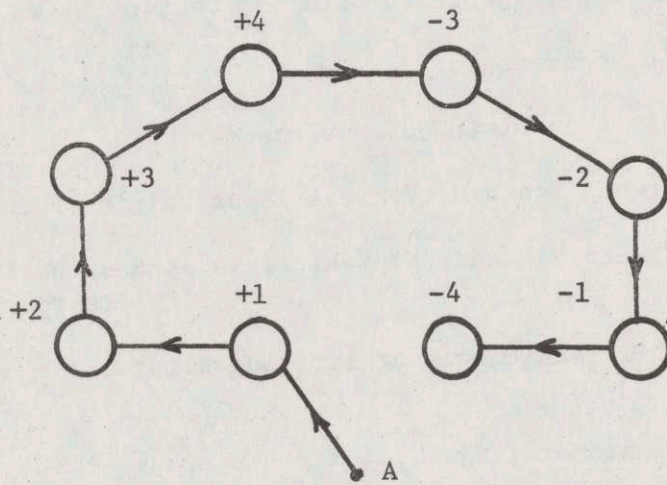
*The nature of the dial-a-ride problem is such that a particular customer has no idea of what the requests from other customers are. To him, the other customers are, more or less, "invisible." He will in general see some of them inside the vehicle, but he will not be able to check such subtle details as initial position, pick-up order, delivery order etc., to verify if he has been treated fairly. This "invisibility" does not exist in the Constrained Position Shifting version of the Aircraft Sequencing Problem examined in Part II, where each pilot can, by actually being aware of the other airplanes, verify himself whether he has been treated fairly or not.

To answer these questions, we state that the purpose of introducing MPS constraints to the dial-a-ride problem is not so much to maintain a degree of "fairness" to the customers of the problem. Rather, these constraints can be viewed as a safety device, necessary for the stability of the operation of the "dynamic" version algorithm, being in fact a means to take care of the danger of indefinite deferment of customer requests. Leaving the detailed discussion of this issue for the next chapter, we briefly state at this point that in the "dynamic" case it is conceivable that the (geographical) location of a customer may be so unfavorable, that the algorithm continuously avoids servicing that customer. Under MPS constraints however, the algorithm will have to service any such customer sooner or later.

We should note here that the vehicle capacity constraints are redundant when $C \geq N$. Also, our priority constraints are redundant when $MPS \geq N-1$. This corresponds to the unconstrained case of our problem. In general however, the constraints will not be redundant and thus, not all vehicle routes will be feasible. For example in Fig. 8.4a, the route is infeasible because it is not even legitimate: The origin of customer 1 is visited after his destination. In Fig. 8.4b, the route will be infeasible if the vehicle capacity is less than 4 because 4 customers will be in the vehicle during the 5th leg of the trip. Finally in Fig. 8.4c, the route will be infeasible if $MPS < 2$ because the shift in the order of pick-up of customer 3 is +2 and the shift in the order of delivery of customer 1 is -2.

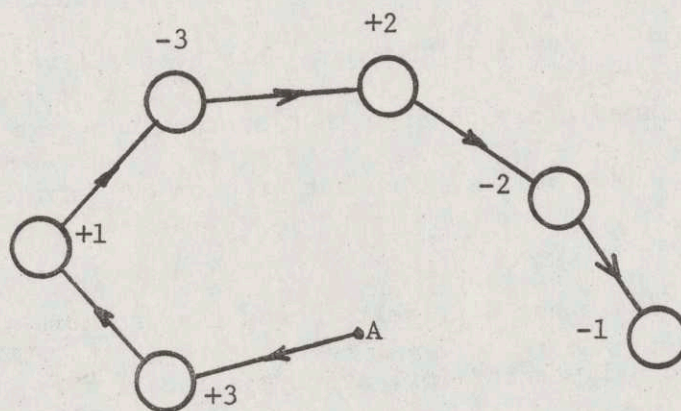


(a)



(b)

(C < 4)



(c)

(MPS < 2)

Fig. 8.4 : Infeasible routes.

8.3 Preliminary Considerations for the Dynamic Programming Approach:

The Dynamic Programming algorithm we shall develop for the solution of the problem we have introduced above, derives from the general Dynamic Programming approach to the TSP, first suggested by Held and Karp [HELD 62], which we already have presented in Chapter 2. However our procedure will be seen to be more sophisticated than that, in the sense that it will incorporate the constraints of the problem in a fashion particularly suited to the nature of Dynamic Programming. In addition, the two alternative objectives can be evaluated with equal ease.

The state representation is achieved with the vector (L, k_1, \dots, k_N) , where we use the following convention:

a) L : Point the vehicle is visiting. We assume that:

$$L = \begin{cases} \text{Between } 1 \text{ and } N: \text{ Vehicle is at origin of customer } L \\ \text{Between } N+1 \text{ and } 2N: \text{ Vehicle is at destination of} \\ \hspace{15em} \text{customer } L-N \\ =2N+1: \text{ Vehicle is at starting point A.} \end{cases}$$

b) k_j : "status" of customer j ($j=1, \dots, N$). We assume that:

$$k_j = \begin{cases} 3 : \text{ Customer } j \text{ has not been picked-up so far} \\ 2 : \text{ Customer } j \text{ is in the vehicle} \\ 1 : \text{ Customer } j \text{ has been delivered.} \end{cases}$$

We can immediately observe that, since L can take $2N+1$ values and each of the N k 's three values, then our problem can have a maximum of $(2N+1)3^N$ different states.

However it is not difficult to see that no more than $2N3^{N-1}+1$ of these states can actually exist (or, as we shall call them, be consistent). The reason is due to the redundancy of our state vector: First if $1 \leq L \leq N$,

then we know that k_L should be equal to 2, since customer L is being picked up at that point. This leaves us with the remaining $N-1$ k's which can take three values each. Similarly, if $N+1 \leq L \leq 2N$, then we know that k_{L-N} should be equal to 1, since customer L-N is being delivered at that point. Finally, if $L=2N+1$ then we know that all k's should be equal to 3, since no customer has been picked up so far. This leaves us with only one consistent state when $L=2N+1$. Adding the states for the three cases above we find the suggested number $2N3^{N-1}+1$.

Observations like the above, although not crucial as far as the algorithm is concerned, will be seen to be very important for the efficient utilization of storage space in a computer implementation of our algorithm. More details will be presented in Chapter 10, a chapter devoted to the computational and combinatorial aspects of the dial-a-ride problem.

Another observation at this point is that although our problem is a multi-stage one, no explicit mention of the stage variable has been initially made. The structure of our algorithm will be such that this variable will not be necessary for the moment. Again, computational issues will necessitate the introduction of this variable and this will also be done in Chapter 10. For now we only state without proof that if we define the stage variable n to be zero when the vehicle is at A, one at the first pick-up stop and so on, until $n=2N$ at the last delivery stop, then $n=2x_1+x_2$ where x_j is the number of k's in the state vector which are equal to j ($j=1,2$). (We shall prove this result in Chapter 10.) Thus the stage variable can be calculated at any moment from the information contained in

the state vector.

8.4 Feasibility considerations

Returning to our state representation (L, k_1, \dots, k_N) , we observe that the existence of the problem constraints will make several states infeasible. We therefore must devise a way to determine which of the states are feasible and which not, subject to our constraints.

First we formalize our earlier observations concerning the state redundancy and consistency. A state (L, k_1, \dots, k_N) is consistent if all the three following conditions are met:

$$1) \text{ if } 1 \leq L \leq N, \text{ then } k_L = 2 \quad (8.1)$$

$$2) \text{ if } N+1 \leq L \leq 2N, \text{ then } k_{L-N} = 1 \quad (8.2)$$

$$3) \text{ if } L = 2N+1 \text{ then } k_j = 3 \text{ for all } j = 1, \dots, N. \quad (8.3)$$

An inconsistent state is obviously infeasible, so state consistency is a necessary (but not sufficient) condition for feasibility. In addition to consistency, a feasible state (L, k_1, \dots, k_N) must satisfy the following constraints:

1) Vehicle capacity constraint: Trying to find an expression for the number of customers in the vehicle as a function of the state vector (L, k_1, \dots, k_N) , we observe that this number is equal to the number of k 's which are equal to 2. Letting therefore $X_2 = \{i: k_i = 2\}$, the vehicle capacity can be written as:

$$|X_2| \leq C \quad (8.4)$$

where $|X_2|$ is the cardinality of X_2 .

It should be noted that (8.4) holds if L is a pickup stop ($1 \leq L \leq N$).

An additional examination of vehicle capacity when L is a delivery stop ($N+1 \leq L \leq 2N$) would be redundant because this constraint will have been satisfied anyway, at the first pick-up stop preceding this delivery stop. Still, one may want to examine vehicle capacity constraints only at delivery points. Then, it is easy to see that the following should hold:

$$|X_2| \leq C-1 \quad (8.5)$$

where X_2 is the same set as defined above.

If (L, k_1, \dots, k_N) does not satisfy the above constraint, it is infeasible.

2) Priority Constraints: Here we have to distinguish between pick-up stops and delivery stops:

a) If the stop we are examining is a pick-up stop (i.e. if $1 \leq L \leq N$), then the position of the corresponding customer in the current list is simply L . In addition, the order, among all pick-up stops, according to which that customer is being actually picked-up, is equal to the number of customers who have been picked up so far, namely to the number of customers, for which the corresponding k is not equal to 3 (see definition of k earlier). Thus, the number of k 's which are not equal to 3 is the pickup order of customer L . The corresponding shift in the order of pick-up of that customer is therefore L minus that pick-up order.

b) If the stop we are examining is a delivery stop (i.e. if $N+1 \leq L \leq 2N$) then the position of the corresponding customer in the current list is simply $L-N$. In addition, the order, among all delivery stops, according to which that customer is being actually delivered, is equal to the number of customers which have been delivered so far, namely to the number of customers, for which the corresponding k is equal to 1. Thus, the number

of k's which are equal to 1 is the delivery order of customer L-N.

The corresponding shift in the order of delivery of that customer is therefore L-N minus that delivery order.

c) Obviously if we are at the starting point (L-2N+1) then we don't have to check priority constraints.

We are now in a position to formulate our priority constraints mathematically:

Pick-up: Let $X'_3 = \{i: k_i \neq 3\}$

If $1 \leq L \leq N$, then

$$|L - |X'_3|| \leq \text{MPS} \quad (8.6)$$

Delivery: Let $X_1 = \{i: k_i = 1\}$

If $N+1 \leq L \leq 2N$, then

$$|L-N - |X_1|| \leq \text{MPS} \quad (8.7)$$

where MPS stands for Maximum Position Shift.

A clarifying observation at this point is that the exact form of the inequality shown above is not binding for our subsequent formulation. For example, the values of MPS in the pick-up and delivery constraints need not be the same. Neither the inequalities need to be two-sided and symmetric. Finally, each customer i may have his own MPS_i . Thus, the most general case will consist of four distinct bounds for every individual customer, imposed pairwise on his position shifts for pick-up and delivery, an upper and a lower bound for each. If any of these bounds is plus or minus infinity, then the corresponding inequality will be one-sided.

The above observation was made to point out that the form of the MPS constraints we assumed, causes no loss of generality and thus will be maintained throughout our subsequent formulation. If the state vector does not

satisfy these constraints, it is obviously infeasible.

3) Route legitimacy constraints: It is important to state at this point that these constraints are taken care of automatically by the structure of our algorithm. We shall return to this point later.

Summarizing so far:

A set of conditions, necessary for the feasibility of a given state vector (L, k_1, \dots, k_N) has been developed. If the state vector fails to obey any one of these conditions, then it cannot be feasible.

Before investigating whether these conditions are sufficient for feasibility, let us present some examples:

Suppose we are dealing with $N=5$ customers numbered 1 through 5. The state vector is $(L, k_1, k_2, k_3, k_4, k_5)$. If $1 \leq L \leq 5$, then the vehicle picks-up customer L , if $6 \leq L \leq 10$, then the vehicle delivers customer $L-5$ and if $L=11$, then the vehicle is at the starting point A. Let us furthermore assume that the vehicle capacity is $C=3$ persons and that $MPS=2$. Several infeasible configurations follow:

$(L, k_1, k_2, k_3, k_4, k_5)$	Reason for Infeasibility
$(1, 3, 2, 2, 1, 2)$	Inconsistency (8.1): $L=1$ and $k_L \neq 2$
$(7, 1, 2, 3, 2, 1)$	Inconsistency (8.2): $L=7$ and $k_{L-N} \neq 1$
$(11, 3, 3, 2, 2, 1)$	Inconsistency (8.3): $L=11$ and not all k 's are 3
$(4, 2, 2, 2, 2, 1)$	Violates capacity constraint (pick-up stop) (8.4): $X_2 = \{1, 2, 3, 4\}$, $ X_2 = 4 > C=3$
$(10, 1, 2, 2, 2, 1)$	Violates capacity constraint (delivery stop) (8.5): $X_2 = \{2, 3, 4\}$, $ X_2 = 3 > C-1=2$
$(5, 3, 2, 3, 3, 2)$	Violates MPS constraint (pick-up stop) (8.6): $X'_3 = \{2, 5\}$, $ X'_3 = 2$, $ L-2 = 3 > MPS=2$
$(2, 3, 2, 3, 1, 3)$	Violates MPS constraint (delivery stop) (8.7): $X_1 = \{4\}$, $ X_1 = 1$, $ L-N-1 = 3 > MPS=2$

We now come to the following question: What happens if a state configuration "passes" all the feasibility tests discussed above? This is the same question as asking whether the conditions for feasibility that have been developed are also sufficient, and if not, under what supplementary conditions we can definitely reach a conclusion on the feasibility of a particular state.

First of all, in general, the conditions already developed are not sufficient for feasibility. This can be demonstrated by the following simple example: Consider $N=3$, $C=3$, $MPS=0$ and the state $(L, k_1, k_2, k_3) = (5, 3, 1, 1)$.

This state passes all tests for feasibility we have mentioned so far:

- a) It is consistent, because $L=5$ and $k_{L-N}=1$
- b) It obeys the capacity constraint, because $|X_2|=0 < C-1 = 2$
- c) It obeys the MPS constraint, because $|X_1|=2$ and $|L-N-2| = 0 = MPS$

But if we take a more detailed look into what this state really means, we observe that, according to the state-description vector, the vehicle is currently delivering customer 2, customer 1 has not been picked up so far and customer 3 has already been delivered. The fact that customer 3 has been serviced before customers 2 and 1 while $MPS=0$ is a clue that our state cannot be feasible. To prove this rigorously, we see that the only potential continuation of that state is the immediate next step to finish the route, i.e. pick-up customer 1. The corresponding state is $(1, 2, 1, 1)$, which fails to pass the MPS constraint, because $|X_3'| = 3$ and $|L-3|=2 > MPS=0$. Since the only potential continuation of our state is an infeasible state, the same should hold for the state itself.

In general, the state which we will be examining will have more than

one "next" states. (We shall define this term mathematically below.) If none of them is feasible, then the state itself cannot be feasible. By contrast, if there is at least one of the "next" states which we know is feasible, and our original state has passed the other feasibility tests, then we can say that indeed that state is feasible.

An important observation at this point is that in order to complete the feasibility investigation of a particular state we must have feasibility information about all "next" states, therefore we have somehow to have performed similar investigations for them at prior stages of our analysis. This constitutes the recursive nature of feasibility, a property which we shall see is particularly suited to the nature of Dynamic Programming. According to this property, the feasibility of a particular state depends not only upon certain criteria that his state itself may or may not satisfy, but also on the feasibility of its "next" states.

Coming now to the problem of determining the set of states which are "next" to a given consistent state (L, k_1, \dots, k_N) , we can see that the definition of the k 's given earlier implies that we can, in one step, move from (L, k_1, \dots, k_N) to (x, k'_1, \dots, k'_N) , where x belongs to either one of the following two sets:

- a) The set of "next" pick-up stops

$$X_1 = \{i: 1 \leq i \leq N \text{ with } k_i = 3\} \quad (8.8)$$

- b) The set of "next" delivery stops

$$X_2 = \{i: N+1 \leq i \leq 2N \text{ with } k_{i-N} = 2\} \quad (8.9)$$

Also the new k -vector (k') is given by

$$k'_j = \begin{cases} k_j - 1 & \text{if } j=x \text{ or } j=x-N \\ k_j & \text{otherwise} \end{cases} \quad (8.10)$$

for all $j=1, \dots, N$

Thus, the set of stops "next" to the stop corresponding to the state (L, k_1, \dots, k_N) is the union of the sets X_2 and X_3 defined above.

It is conceivable of course that this set is the null set. This would mean that all k 's in (L, k_1, \dots, k_N) are equal to 1, namely all customers have been delivered. We shall call any such consistent state a terminal state.

A terminal state has no "next" states. The implication of this is that in examining the feasibility of a terminal state, the conditions (8.1) through (8.7) we have introduced already, are not only necessary but also sufficient to determine whether this state is feasible or not. If the state is not terminal, then these conditions are not sufficient and we furthermore have to examine the feasibility of the states "next" to it as these are defined in (8.8) through (8.10)*.

It is at this point that we can see why the route legitimacy constraints are automatically satisfied. The definition of a "next" state above implies that a customer's pick-up stop (change of k from 3 to 2 for that customer) will always precede his delivery stop (change of k from 2 to 1).

Summarizing all our observations concerning feasibility we conclude that feasibility in the dial-a-ride problem is of a recursive nature. We shall see later how we can implement this recursive characteristic in the execution of a part of our algorithm (called "Screening"). At this point

*The above observations are essentially similar to those that lead to the Recursive Feasibility Theorem presented for the Aircraft Sequencing Problem in Chapter 5.

we only state that this part of the algorithm creates feasibility information for all states (L, k_1, \dots, k_N) , that it determines whether each state is feasible or not. This information will be used for our subsequent consideration (and the algorithm's corresponding part) concerning optimality.

8.5 Optimality Considerations

So far we have not stated anything concerning the problem's objective and how optimization can be achieved. In fact, feasibility considerations are completely independent of the objective-optimization aspects of the problem.

For a particular state (L, k_1, \dots, k_N) which we know is feasible (from the information created by "Screening" earlier), we define a quantity $V(L, k_1, \dots, k_N)$ as our optimal value function. This is the optimal value (measured in terms of our specific objective function) of all our subsequent decisions from (L, k_1, \dots, k_N) till the accomplishment of the current task. Remembering that the general objective function we consider in this problem is to minimize $w_1 \cdot \sum_{j=1}^{2N} T_j + w_2 \cdot \sum_{i=1}^N [\alpha \cdot WT_i + (2-\alpha) \cdot RT_i]$, we conclude that, V must obey an optimality recursion of the following form:

$$V(L, k_1, \dots, k_N) = \min_{x \in X} [M \cdot t_{L,x} + V(x, k'_1, \dots, k'_N)] \quad (8.11)$$

where X is the set of x 's corresponding to feasible states (x, k'_1, \dots, k'_N) which are "next" (according to the definition in (8.8) through (8.10)) to (L, k_1, \dots, k_N) . Actually X is the union of X_3 and X_2 defined in (8.8) and (8.9).

Some questions are still unanswered about (8.11). First of all, what is the proportionality factor M by which the time $t_{L,x}$ needed to go from L to x should be multiplied? And second, what happens if X is the null set?

To estimate M , we evaluate all the marginal contributions to the value of our objective due to the fact that the vehicle travelled from L to x . These can be divided into two categories:

a) Those which are reflected into the term $w_1 \cdot \sum_{j=1}^{2N} T_j$. Since T_j is the duration of j^{th} leg of the trip, the corresponding marginal contribution is simply $w_1 \cdot t_{L,x}$.

b) Those which are reflected into the term $w_2 \cdot \sum_{i=1}^N [\alpha \cdot WT_i + (2-\alpha)RT_i]$. Here WT_i and RT_i are the waiting and riding times for customer i . But how many customers are waiting for the vehicle during $t_{L,x}$ and how many of them are riding in it? The answer is not very difficult to obtain. Letting $X_3 = \{i: k_i=3\}$, it is easy to see that $|X_3|$ is the number of customers which have not been picked up, i.e. are still waiting for the vehicle while the latter leaves state (L, k_1, \dots, k_N) . Also, the number of customers riding in the bus from L to x has already been seen to be equal to $|X_2|$ with $X_2 = \{i: k_i=2\}$. Thus, the marginal contribution to this term of the objective is $w_2 \cdot (\alpha |X_3| + (2-\alpha) \cdot |X_2|) \cdot t_{L,x}$.

We can see therefore that

$$M = w_1 + w_2 \cdot (\alpha \cdot |X_3| + (2-\alpha) \cdot |X_2|) . \quad (8.12)$$

All the parameters shown above are either known a priori (w_1, w_2, α) , or can be readily estimated from the vector (L, k_1, \dots, k_N) .

Concerning now the question of what happens if X is empty, we can say that if (L, k_1, \dots, k_N) is not a terminal state (i.e. there is at least one k which is not equal to 1), then X cannot be empty. The reason for this lies with the recursive nature of feasibility already discussed earlier. If (L, k_1, \dots, k_N) is feasible, then by definition there should be at least one "next" state (x, k'_1, \dots, k'_N) which is also feasible, unless (L, k_1, \dots, k_N)

is a terminal state. If now the latter happens, then V is equal to zero, for we already have finished. This constitutes the boundary condition of our optimality recursion:

$$V(L, 1, \dots, 1) = 0 \quad (N+1 \leq L \leq 2N) \quad (8.13)$$

If (L, k_1, \dots, k_N) is not feasible, then (8.11) has no meaning and should not be executed.

8.6 The Dynamic Programming Algorithm

Our observations above lead to the development of a Dynamic Programming algorithm, consisting of three parts:

1) The "Screening" part, where we determine recursively whether any state (L, k_1, \dots, k_N) is feasible or not. We start from all terminal states $(L, 1, \dots, 1)$ where only conditions (8.1) to (8.7) suffice for feasibility and then move lexicographically (with respect to the k -vector) up to $L=2N+1$ and $k_j=3$ for all j . For a non-terminal state (L, k_1, \dots, k_N) to be feasible, in addition to seeing if (8.1) to (8.7) hold, we determine if there exists at least one feasible "next" state. We tabulate this information into the logical array $FEASBL(L, k_1, \dots, k_N)$ with values "true" or "false" depending on whether the corresponding state is feasible or not. This information will be used in the subsequent "Optimization" part.

2) The "Optimization" part consists of applying the optimality recursion (8.11), using also (8.8) through (8.10) plus (8.12), (8.13), only for those states characterized as feasible by the "Screening" part. The order in which this recursion is performed is the same as in "Screening" (lexicographic with respect to the k -vector and for all values of L). Each time we apply the recursion, we store two pieces of information for subsequent use by the algorithm: First, the value of V (to be used in

subsequent steps of the same part of the algorithm, when the current state appears as a "next" state in the right-hand-side). Second, our corresponding best choice, namely the element x of X in (8.11) which minimizes the right-hand-side. We store this x in an array which we call $NEXT(L, k_1, \dots, k_N)$. This will be used in the third part of our algorithm.

3) The "identification" part consists of using the information of the array $NEXT$ created previously, to identify the optimal sequence. We start from the state $(2N+1, 3, 3, \dots, 3)$ (vehicle at starting point A). The subsequent point will be given by the value of $NEXT(2N+1, 3, 3, \dots, 3)$. Adjusting the k -vector accordingly, we determine the new state and so forth, until, after $2N$ steps, we reach a terminal state*.

8.7 Computational Effort and Storage Requirements.

It is not difficult to see that if we keep the state formulation (L, k_1, \dots, k_N) redundant as it is, then we have to use $(2N+1)3^N$ memory locations for each of the three arrays $FEASBL$, V and $NEXT$. Under an elementary transformation, we can bring this figure down to $2N \cdot 3^{N-1} + 1$ locations. This transformation will be presented in Chapter 10, where some additional, more subtle efforts to "streamline" the efficiency of the dial-a-ride algorithm will also be presented. For the moment, our algorithm, as it stands, has to use the former amount of storage space. Running time for the algorithm grows in a similar fashion, a fact which does not permit characterization of the algorithm as "efficient." Still, as compared with the computational effort of the Dynamic Programming

*This is essentially the same procedure as that developed in Chapter 4 for the Aircraft Sequencing Problem.

algorithm developed to solve the TSP [HELD 62], and for a graph of the same size ($2N+1$ nodes), the dial-a-ride algorithm performs asymptotically better: In fact, the ratio $\frac{(2N+1)3^N}{(2N+1)2^{2N+1}} = \frac{1}{2}\left(\frac{3}{4}\right)^N$ goes to zero as $N \rightarrow \infty$. The reason is that the dial-a-ride algorithm presented here takes advantage of the fact that not all TSP tours are feasible for the dial-a-ride problem.

The vehicle capacity, C , and the Maximum Position Shift, MPS, also play roles in determining the algorithm's computational effort. These roles however, are not easy to evaluate in an exact quantitative manner at this moment. An attempt in this direction will be made in Chapter 10. For the moment, we make the obvious qualitative observation that the computational effort of the algorithm increases as C or MPS increase. This is due to the fact that by increasing C or MPS we make more states feasible and thus, we have to execute the optimality recursion (8.11) more times.

8.7 Computer runs-discussion

We present now a series of cases regarding the specific graph of 15 nodes ($N=7$ customers) presented in Figure 8.5. We have assumed, without loss of generality, that the distance metric of the problem is Euclidean, so that Cartesian coordinates may be used to describe the location of each point. For our problem, these coordinates are given in the following table:

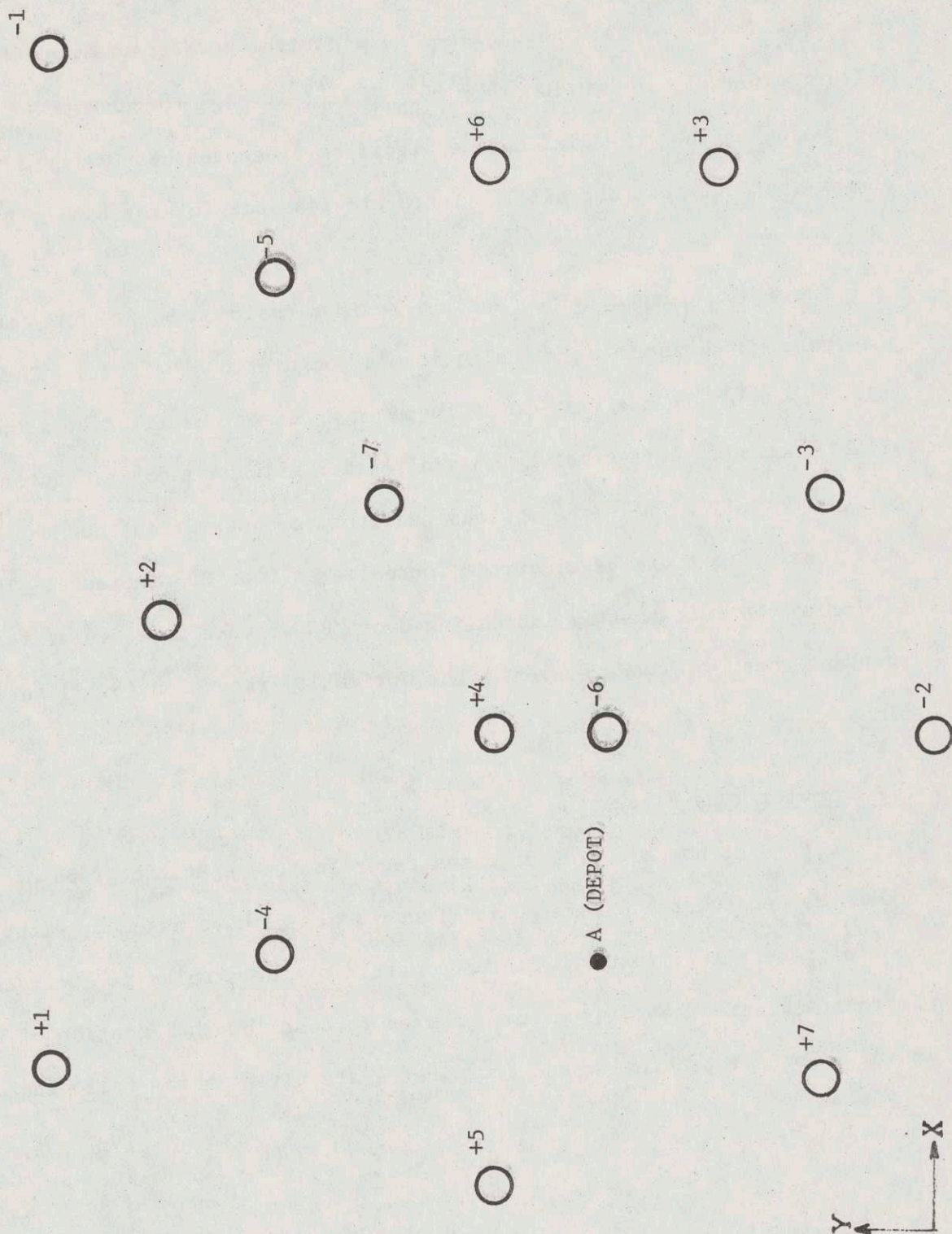


Fig. 8.5 : An example of the "static" case of the dial-a-ride problem.

Several cases concerning this example are depicted in figures 8.6 to 8.12.

C u s t o m e r	Pick-up point coordi- nates (miles)		Delivery point coordi- nates (miles)	
	X	Y	X	Y
1	2	8	11	8
2	6	7	5	0
3	10	2	7	1
4	5	4	3	6
5	1	4	9	6
6	10	4	5	3
7	2	1	7	5

The starting point is located at (3,3). Without loss of generality, a constant vehicle speed of 30 mph. is assumed.

Our cases vary as far as the objective function, the vehicle capacity and MPS constraints are concerned. Concerning the objective, we have examined only the cases $Z=1$ (minimize time till delivery of last customer) and $Z=2$ (minimize total customer disutility). In all cases the input value of α is equal to 1.00.

The cases examined are graphically represented in Figures 8.6 through 8.12. These are the plotted outputs of the corresponding computer runs. The starting point is arbitrarily named "DEPOT".

The following table summarizes some of the quantitative results of the cases examined. An asterisk denotes that the corresponding value is optimal:

Case	Objective Z	MPS	C	Time till Delivery of Last customer (mins.)	Total Passenger Disutility (pass X mins.)
1	2	6	7	87	*363
2	1	6	7	* 79	385
3	2	2	3	101	*472
4	1	2	3	* 99	483
5	2	6	1	136	*476
6	2	0	7	126	*528
7	1 or 2	0	1	*139	*596

Commenting on these cases, we note the following:

1) Cases 1 (Fig 8.6) and 2 (Fig. 8.7) represent the unconstrained situation (when no MPS and capacity constraints essentially exist). One can notice position shifts up to 6 in both cases (Customers 1 and 7 in case 1 and customer 7 in case 2 with respect to both pick-up and delivery). The actual maximum number of passengers in the vehicle is 4 and occurs at the 5th leg of the vehicle route in Case 2. The difference in optimal routes can be observed. Note for instance that the route for case 2 looks very much like a typical good route in a Travelling Salesman Problem.

2) In Cases 3 (Fig. 8.8) and 4 (Fig. 8.9) both capacity and MPS constraints have been imposed. A corresponding change in the optimal routes, as well as a deterioration in the values of the measures of performance can also be observed.

3) Cases 5 (Fig. 8.10), 6 (Fig. 8.11) and 7 (Fig. 8.12) are extreme in the sense that in each of them either the vehicle capacity constraint,

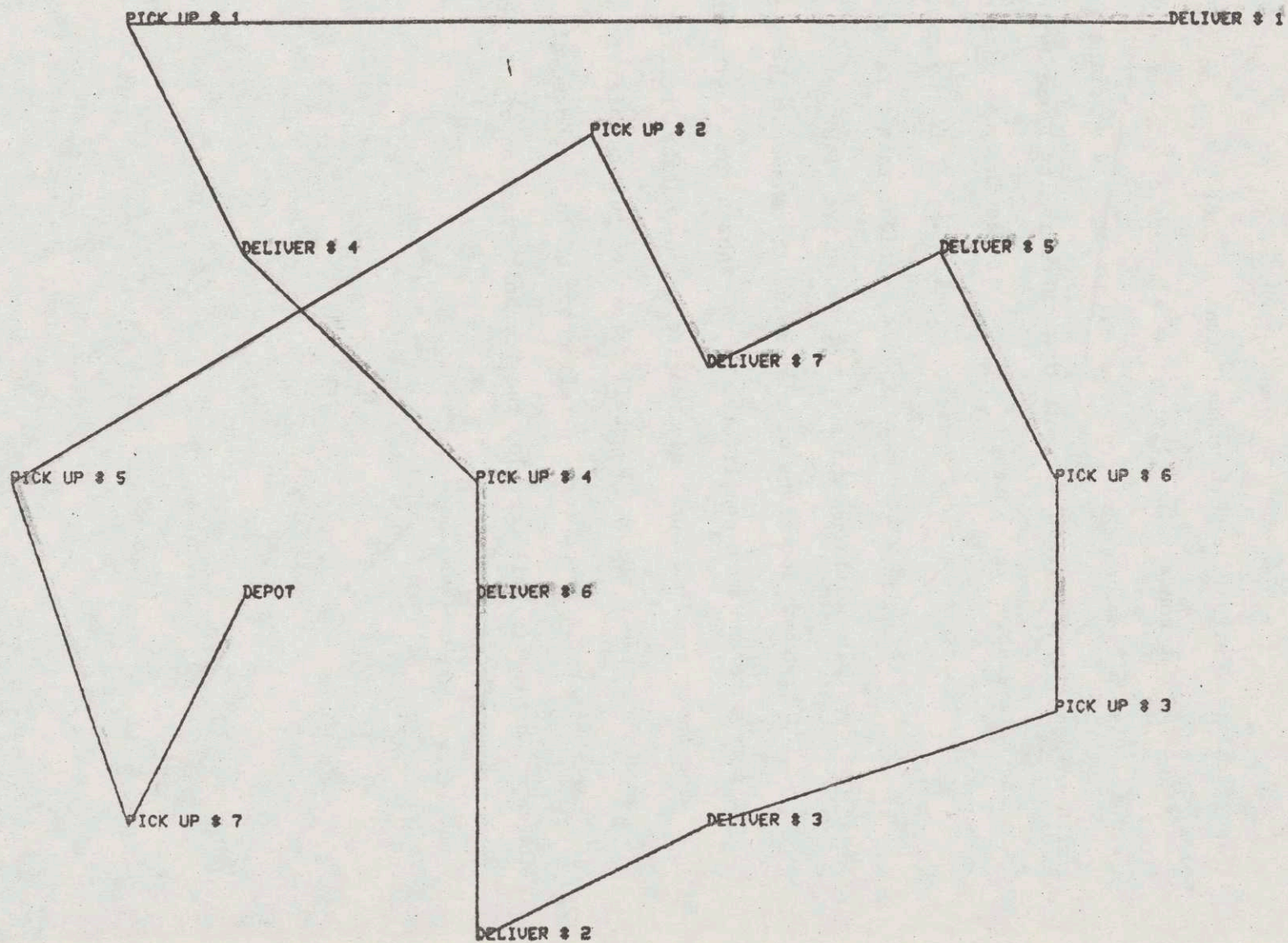
or the MPS constraint, or both, are as tight as possible: Case 5 examines a hypothetical situation with minimal (unit) capacity, but no MPS constraints. As expected, the route is an alternating sequence of pick-up and delivery stops. Case 6 examines the First-Come-First-Served discipline ($MPS=0$) but with no capacity constraints. Note that there is still one degree of freedom for optimization left, namely how one will actually merge the two FCFS sequences of pick-ups and deliveries. Finally Case 7 is the most restrictive of all: $MPS=0$ and unit capacity. It is clear that this problem has only one feasible solution, whatever the objective might be.

4) For "random" locations of customers, the shape of the optimal route gets more and more complicated as we tighten the constraints.

The above conclude our basic considerations on the "static" version of our dial-a-ride problem. We shall return to this problem in Chapter 10, for the details concerning the refinement of the proposed algorithm. The next Chapter (9) will develop the "dynamic" version of our problem.

Fig. 8.6 : Case 1.

- 162 -



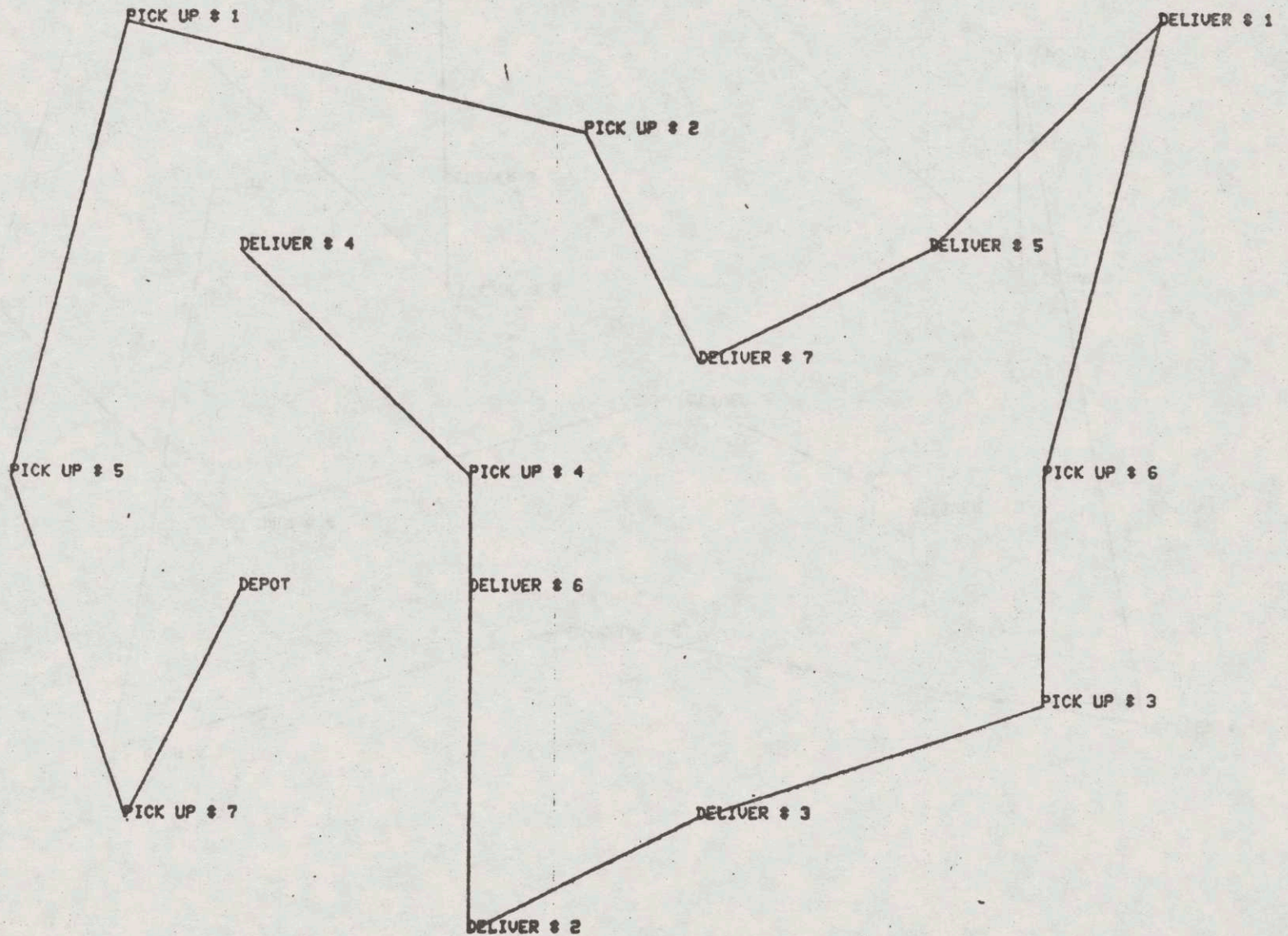


Fig. 8.7 : Case 2.

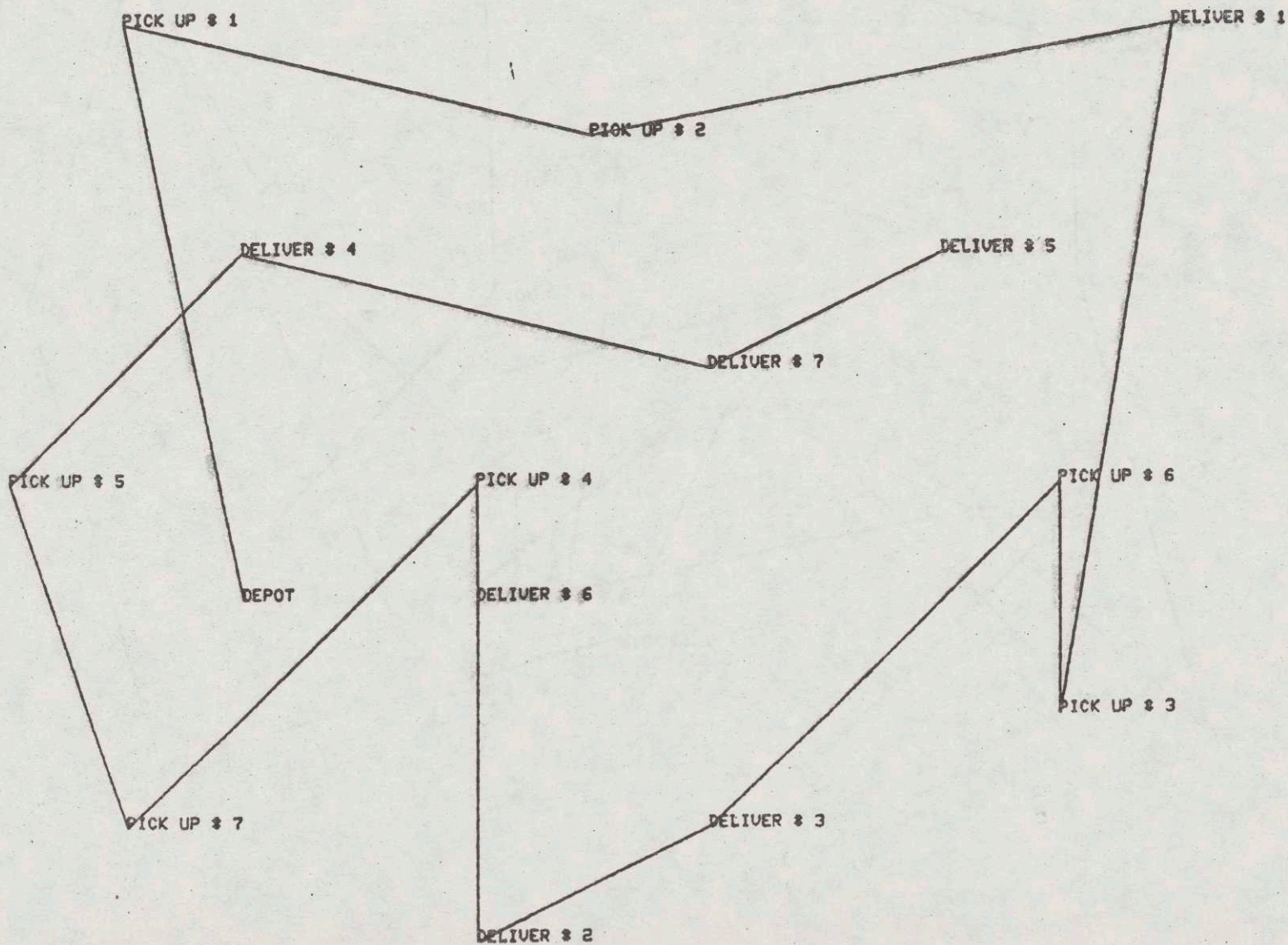
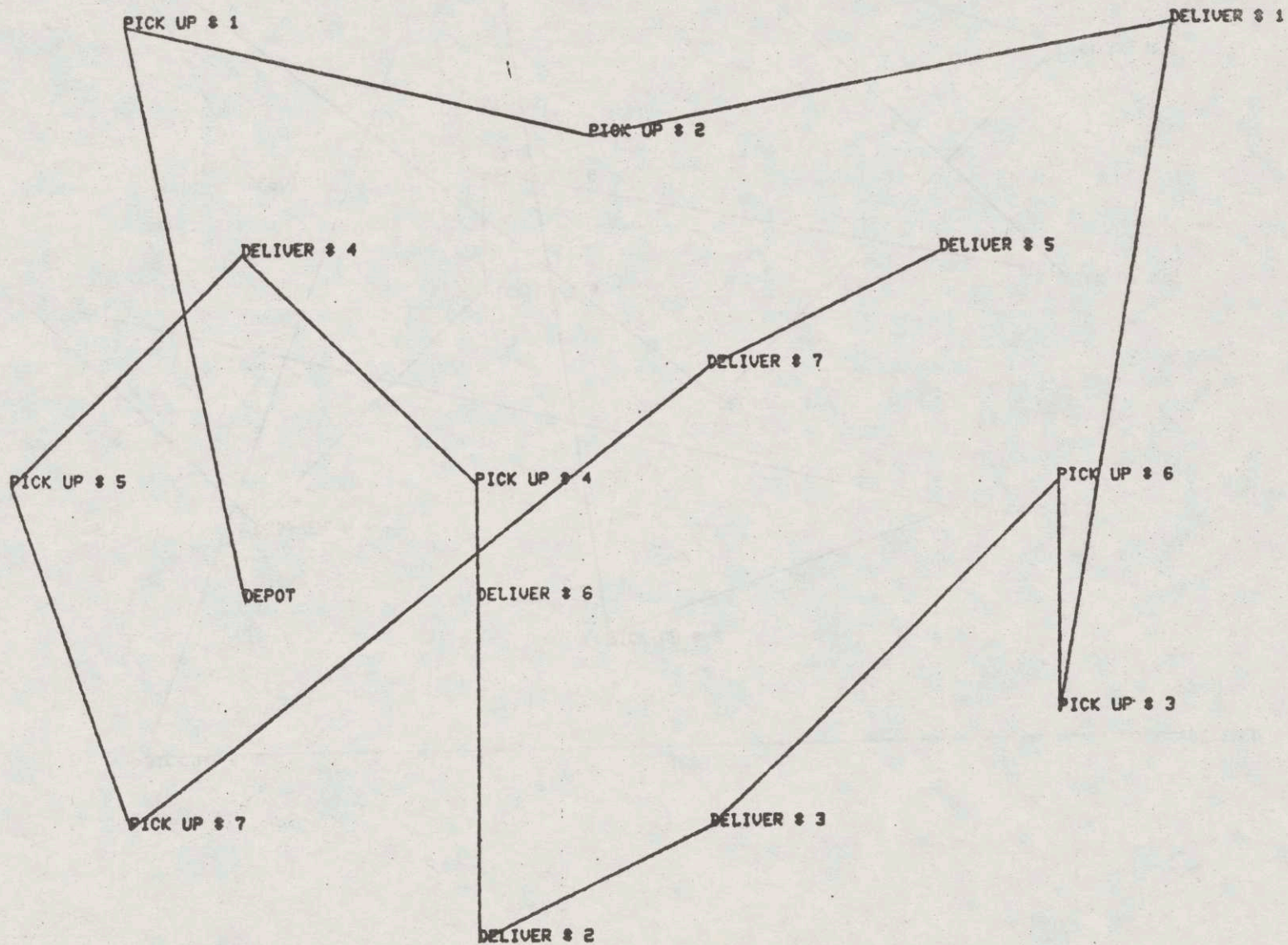


Fig. 8.8 : Case 3.

Fig. 8.9 : Case 4.

- 165 -



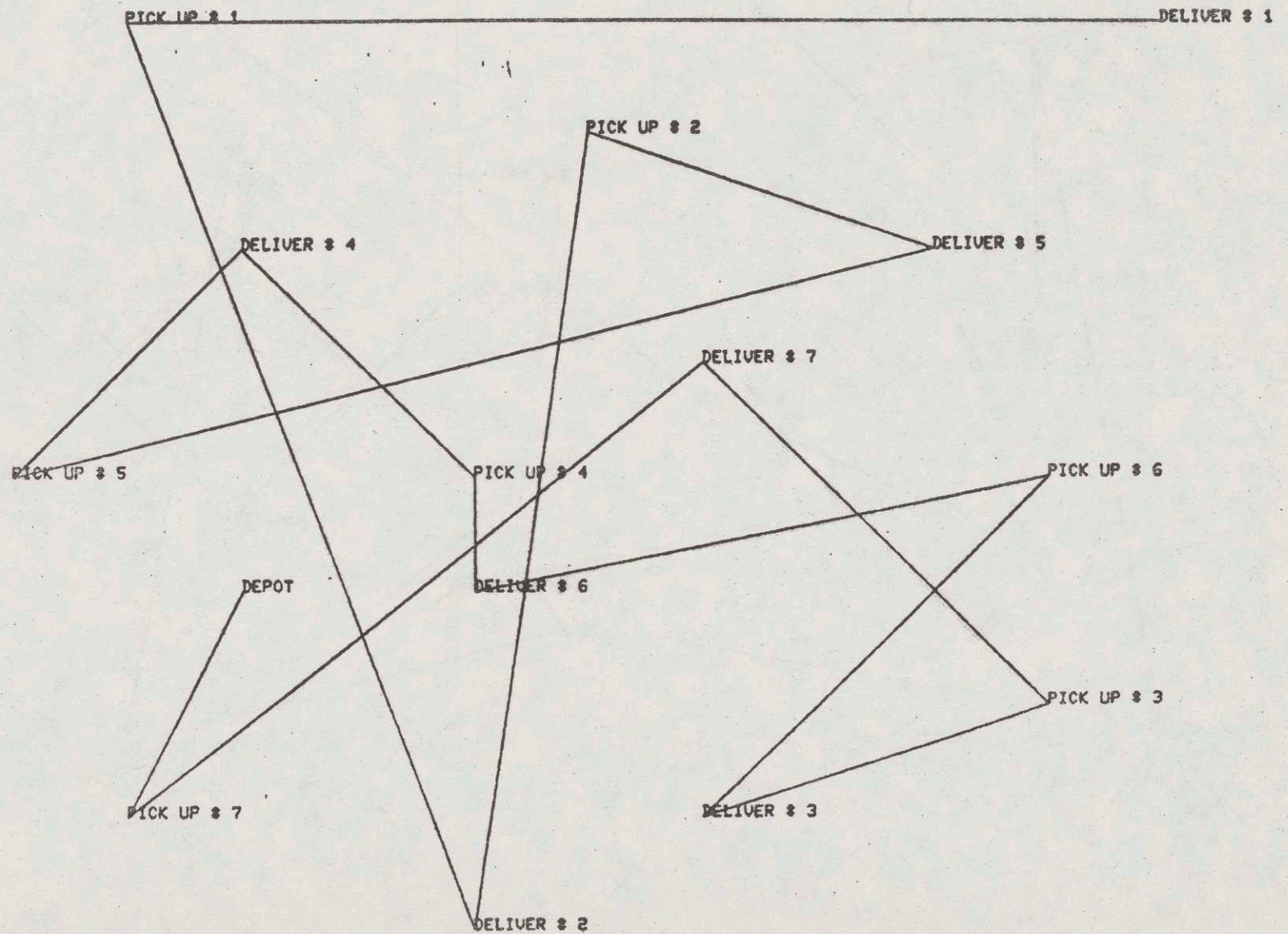


Fig. 8.10 : Case 5.

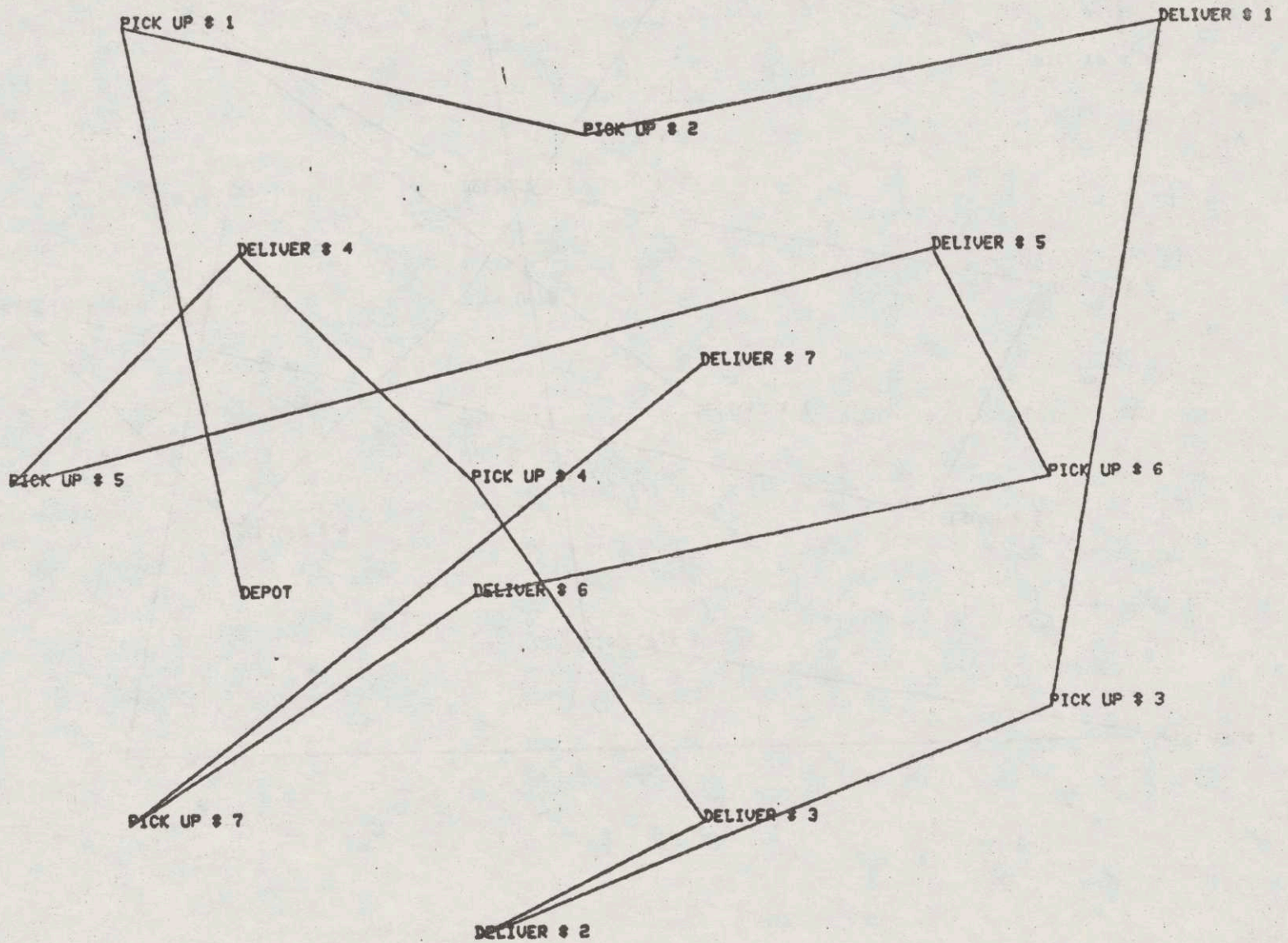


Fig. 8.11 : Case 6..

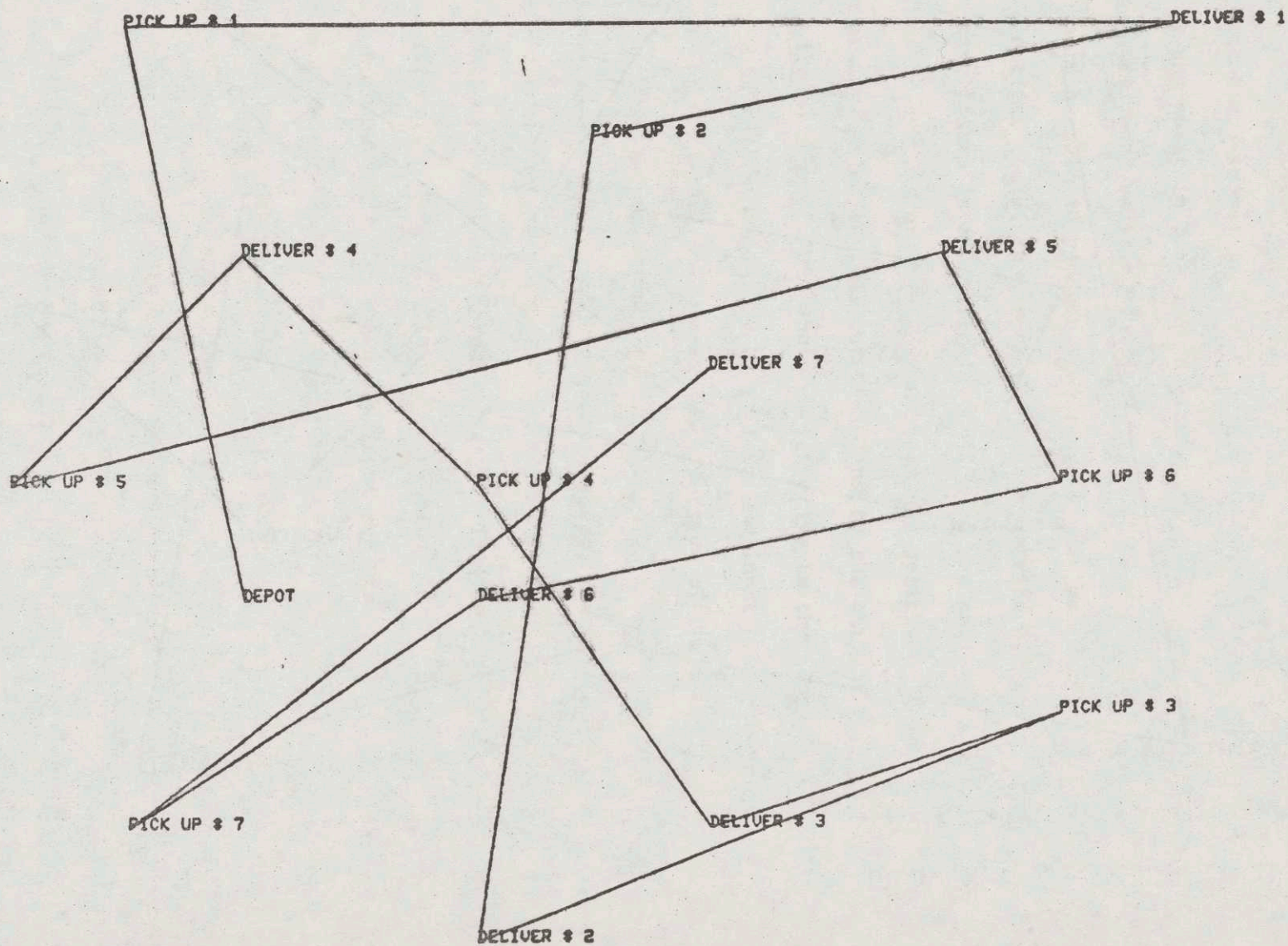


Fig. 8.12 : Case 7.

CHAPTER 9

DIAL-A-RIDE-"DYNAMIC" CASE: DYNAMIC PROGRAMMING SOLUTION

9.0 Introduction: Problem Formulation and Approach

We saw in the previous chapter that the main assumption of the "static" version of the dial-a-ride problem is that at a given point in time ($t=0$) the list of customer requests is closed and no new requests are further considered, until all the customers that the list contains at $t=0$ are serviced. In this chapter we shall investigate what happens if we drop the above assumption. The corresponding version constitutes the "dynamic" case of the dial-a-ride problem.

In the "dynamic" case, each new customer request is automatically eligible for consideration at the time it appears. (As in the "static" case, we shall be concerned only with immediate requests.) Thus, each time such a request appears, the input of our problem is changed and one should expect a corresponding change to the problem's solution as well. This change in the solution, for obvious reasons, cannot affect the portion of the route already executed prior to the time the new request appears but will certainly change the remaining portion of the route. Since new requests may be arriving for an indefinite period of time, it is clear that the "dynamic" version of the problem is by nature open-ended, and the corresponding solution procedure must be active as long as these new inputs keep entering our problem. Although at this point it is somewhat premature to discuss specific details of this solution procedure, we

can, at least qualitatively, suggest several principal features that this procedure should possess:

1) At any point in time and for the current set of problem inputs (current position of vehicle, current status of all customers), the procedure should be able to produce upon request what we shall call the tentative optimal route, in a reasonably short period of time. The tentative optimal route is the optimal solution to the equivalent "static" version of the problem at the time, namely the optimal solution (according to a prescribed objective) corresponding to the current set of inputs and only to those. This route is tentative in the sense that it will indeed be totally optimal if and only if no new input appears during its execution. By contrast, the non-executed portion of the route is subject to revision, upon the appearance of a new input.

2) Each time a new customer request appears, the procedure should be able to incorporate this new information as part of the problem's input and somehow solve the new problem again. This updating process must also be performed in a reasonable amount of time. A new tentative optimal route will be produced each time a new input appears.

3) The procedure should not defer the service of any particular customer indefinitely. In other words, the conceivable case of a customer being continually assigned the last position in the sequence because of his unfavorable geographical characteristics, should not be permitted to occur. We recall that a similar argument was used for the development of the second generation dial-a-ride algorithm in [WILS 77b]. We have already seen in Chapter 8 why this algorithm and, a fortiori, the earlier proposed first generation one [WILS 76] are sub-optimal with respect to their reassessment

capabilities.

Let us present a very simple example on how a "dynamic" procedure might operate: In Figure 9.1a, the procedure is given a set of inputs (A: position of vehicle, locations of origins and destinations for customers 1 and 2) and subsequently produces the tentative optimal route (A, +2, +1, -2, -1). At the time the vehicle is at B, a new input appears, in the form of a request by customer 3 (Fig. 9.1b). At this point in time, the procedure revises the non-executed portion of the route and produces the new tentative optimal route shown in Figure 9.1c. Note that the new route does not have B as origin, but a point B', slightly ahead of B. This is due to the fact that it will, in general, take some time for the procedure to process the new input and to solve the new problem and this time is reflected by the segment BB'.

From Figure 9.1c it can be seen that, conceivably, the new tentative optimal route will bear a minimal resemblance to the old, non-executed part of it. This process is repeated indefinitely as long as new inputs are entering our problem.

Before elaborating on the details of the algorithm of the "dynamic" version, we have several additional remarks:

1) Let $(A \equiv P_0, P_1, \dots, P_n)$ be a particular sequence of stops which constitutes a tentative optimal route. It can be shown that for any i ($0 \leq i \leq n$), the sequence $(B, P_{i+1}, P_{i+2}, \dots, P_n)$, where B is any intermediate point situated on the arc (P_i, P_{i+1}) , is also a tentative optimal route for the set of stops $P_{i+1}, P_{i+2}, \dots, P_n$, provided no new input appears while the vehicle travels between A and B. The proof of the above property is by contradiction, as follows:

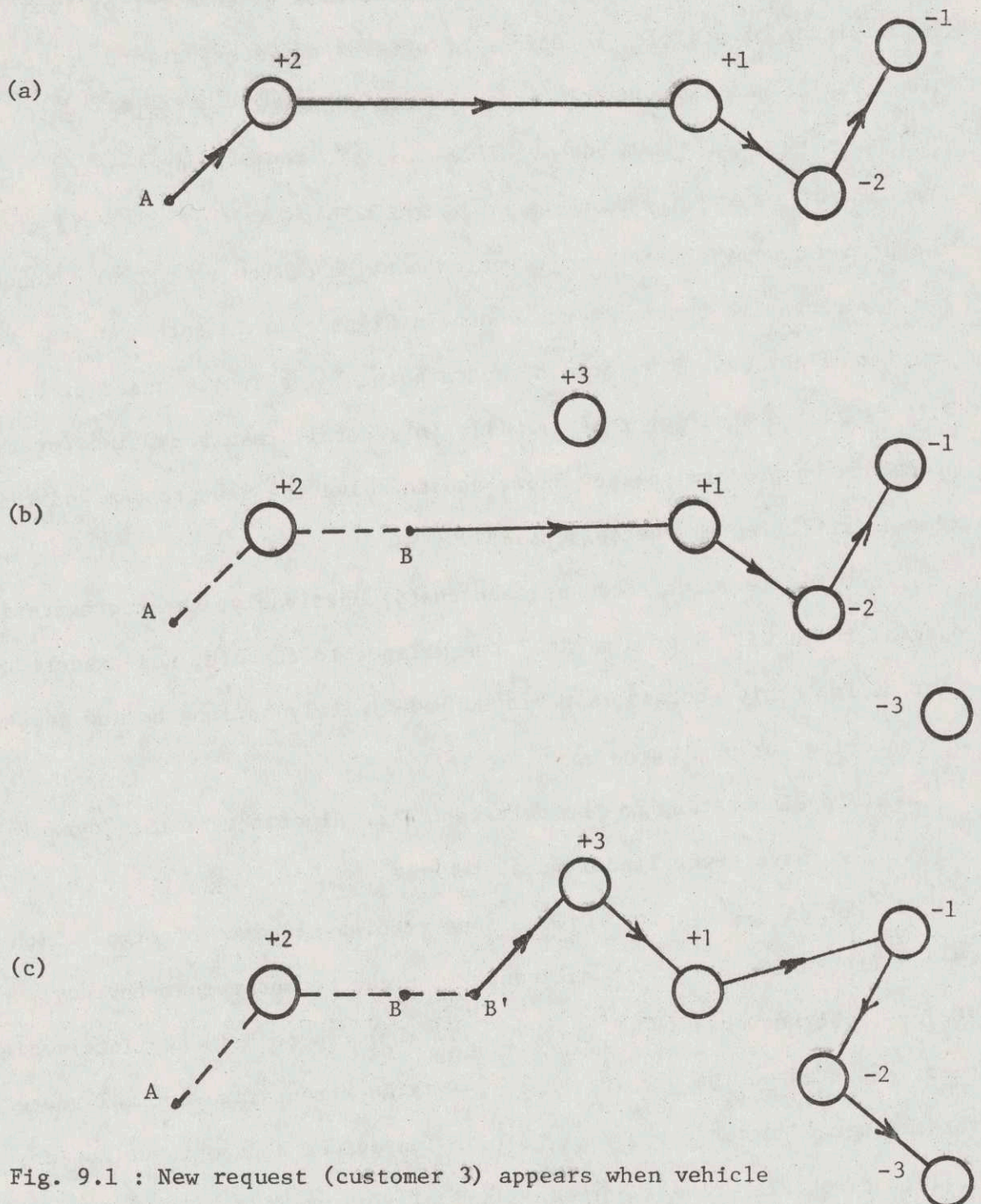


Fig. 9.1 : New request (customer 3) appears when vehicle is at B. New tentative optimal route is produced when vehicle is at B'.

Suppose that the sequence $(B, P_{i+1}, P_{i+2}, \dots, P_n)$ is not a tentative optimal route. Then let $(B, P'_{i+1}, P'_{i+2}, \dots, P'_n)$ be the one which is (this last sequence originates from B and visits all of the unvisited nodes $(P_{i+1}, P_{i+2}, \dots, P_n)$ but in a different order $(P'_{i+1}, P'_{i+2}, \dots, P'_n)$). But if this is true, then we can always replace $(B, P_{i+1}, P_{i+2}, \dots, P_n)$ by $(B, P'_{i+1}, P'_{i+2}, \dots, P'_n)$ in the initial sequence $(A \equiv P_0, P_1, \dots, P_1, B, P_{i+1}, \dots, P_n)$ and get a sequence $(A \equiv P_0, P_1, \dots, P_1, B, P'_{i+1}, \dots, P'_n)$ of smaller "cost." But this is a contradiction, since we know that $(A \equiv P_0, P_1, \dots, P_n)$ is tentatively optimal, therefore of minimum "cost." Note that the separability of the objective function of our problem plays an essential role in the above argument. In Chapter 2 we have already argued that separability is an essential property for the solution of sequencing problems by dynamic programming.

2) A clarification, which actually reflects our basic optimization philosophy for the "dynamic" version of the problem is that each time we update the problem, we shall be considering as the problem's inputs all the information we have on specific and known customer requests. By contrast, we shall not be concerned with any, possibly available, probabilistic information describing the spatial or time distributions of other future requests. In this spirit, no customer request will be either anticipated or in any other way taken into account before it actually occurs. It is not difficult to see that if we had taken such probabilistic information into account, this would have resulted in a significant reformulation of our problem and an equivalently significant increase of its computational

difficulty as well*. We should note at this point that our decision not to incorporate probabilistic information into the problem reflects only the way our algorithm "thinks" and should not be confused with the use of probabilistic forms to test the algorithm (simulation).

3) The above argument facilitates the justification of the alternative objectives of the "dynamic" version. To begin with, suppose that we adopt the same general form of objective introduced in the previous chapter, namely to minimize the function:

$$w_1 \cdot \sum_j T_j + w_2 \cdot \sum_i [\alpha \cdot WT_i + (2-\alpha) \cdot RT_i] \quad (9.1)$$

We did not specify ranges for the indices i and j for the moment, because the following issues must be resolved first:

- a) Which route legs will be accounted for?
- b) Which customers will be included?
- c) How do we count WT_i and RT_i ?
- d) What happens to the customers, which are in the vehicle at the moment of our decision? (Recall that no such customers exist in the "static" version of the problem we have examined in Chapter 8.)
- e) What happens to this objective when a new customer requests service?

To address the above issues, we note first that each time an update

*The inclusion of probability density functions for the locations of future customer pick-up and delivery stops, as well as for the time intervals between two successive requests would imply that any measure of performance would be a random variable, which should therefore be minimized on the average. Under this scheme, it is even conceivable that the vehicle trajectories will not be straight lines (even if the distance metric is Euclidean) but rather curves, depending on the form of the probability distributions in a, most likely, complex fashion.

of the problem is performed, there is no reason to worry about the portion of the route which has already been executed, because no subsequent decision on our part can influence what has happened in the past. Because of this, all "costs" (i.e. changes in the value of the objective function) that have occurred in the past, are "sunk costs" and, since these costs are additive, can be neglected from further consideration.

What our decisions can affect however, are the costs associated with the future, namely, the additional costs to service the rest of the customers. In this respect, it is reasonable to state that our objective (9.1) can be interpreted as the minimization of a weighted combination of the following two quantities:

- i) The time to service all customers which have requested service but have not been serviced so far. We start counting this time from the instant our update can be implemented. (This is the instant t_B , the vehicle reaches the point B' as in Figure 9.1c)
- ii) The additional disutility to all of the above customers. It is clear that at the instant t_B , our update can be implemented, all active customers are either inside the vehicle, or are still waiting to be picked up. (A customer already delivered is obviously not considered anymore.) In the first case, the customer's additional disutility is solely measured by the term $(2-\alpha).RT_1$, where we start counting RT_1 from t_B . In the second case, we should also consider the term $\alpha.WT_1$ we start counting WT_1 from t_B , as well as RT_1 from the instant of the corresponding pick-up.

When a new customer requests service, we incorporate the information concerning him in the problem input. We also remove from the problem

inputs any customers which have been serviced. We "reset" our time origin at t_B , and solve the problem again.

4) It is at this point that we consider the priority constraints for the "dynamic" version of the problem. It was mentioned in the previous Chapter that the purpose of these constraints is to take care of the undesirable possibility of indefinite deferment in the "dynamic" case. A detailed mathematical formulation of these constraints has already been presented in the previous Chapter for the "static" case. The formulation is essentially the same for the "dynamic" case. A minor difference is due to the fact that the number of passengers we are examining is not a fixed quantity N as in the static case, but a quantity which in general takes different values from one update to another.

9.1 Solution Algorithm

With the above considerations in mind we can see that our "dynamic" solution procedure will essentially be a series of applications of the "static" algorithm, one application for every update. The "static" algorithm has been already described in detail in the previous Chapter. Several points concerning the required modifications follow:

1) The "static" version assumes that the vehicle starts empty from A. By contrast, whenever we shall be doing an update in our "dynamic" version, the vehicle may have some customers in it. It is easy to see that the latter configuration can be easily transformed to one similar to the former. This is done by resetting the locations of the pick-up stops of all customers in the vehicle to coincide with the location of the decision point B' . In this way, the algorithm will immediately assign all these customers to the vehicle before any other stop is visited. The

creation of these dummy pick-up stops will have the same effect as the vehicle starting with these customers in it.

2) We keep our dynamic programming state representation (L, k_1, \dots, k_N) as it has been introduced in the previous Chapter, but we also introduce a labelling scheme to uniquely identify each of the customers that our algorithm processes. Thus, we assign labels to customers according to the order in which they call for service. $\text{LABEL}(i)$ is the label assigned to the i^{th} of the N customers our algorithm considers in each of its updates*.

3) The above labelling scheme will be seen to be very helpful for the mathematical formulation of our priority constraints: We further define as D the number of customers which have been fully serviced since the beginning of the vehicle trip (i.e. since we assigned the label "1" to a customer). Then it is not difficult to see that the pick-up MPS constraint (needed to determine whether a state (L, k_1, \dots, k_N) is feasible or not) is:

$$|\text{LABEL}(L) - D - |X'_3|| \leq \text{MPS}$$

where $X'_3 = \{i: k_i \neq 3\}$

This constraint is applicable if $1 \leq L \leq N$. If $N+1 \leq L \leq 2N$, then we have a delivery MPS constraint, the following

$$|\text{LABEL}(L-N) - D - |X_1|| \leq \text{MPS}$$

where $X_1 = \{i: k_i = 1\}$

*It should be noted that we should also have a relabelling scheme, because customers which are being delivered, exit our system and should not be incorporated in future updates. Such a scheme is included in the computer implementation of our algorithm.

One can see the validity of the above relations by noting that $D+|X'_3|$ is the total number of pick-up stops so far and that $D+|X_1|$ is the total number of delivery stops.

The capacity constraints are formulated in exactly the same way as they have been in the previous Chapter.

4) Some words are also necessary concerning the determination of the location of the decision point B' : Recall that due to the non-instantaneous processing of the new input, an interval of time will elapse between the instant t_B of the request (when the vehicle is at B) and the instant t'_B , that any decision on our part can be actually implemented. This time interval $(t'_B - t_B)$ is reflected in the segment BB' . So actually, at the instant t_B the vehicle is at B , an estimate of the time needed to "run" the algorithm, produces the point B' and it is for this new origin that the subsequent update is done. For large distances or fast computers one may assume that B' coincides with B .

9.2 Computer Runs-Discussion of the Results

We shall now present two cases to which we shall apply our "dynamic" algorithm. The only difference between the two cases is in our objective. Case 1 concerns the minimization of the time to service all (known) non-serviced customers and Case 2 the minimization of their total disutility. α is assumed equal to 1, $MPS=3$ and $C=4$ for both cases. Both cases are for the same chronological sequence of customer requests and for the same location of customers. Vehicle speed is assumed constant at 30 mph and the distance metric of the problem is assumed Euclidean.

At $t=0$ the vehicle becomes available at point $(1,4)$, where numbers note cartesian coordinates in miles. At that time, the following customers

have already requested service;

Customer	Pick-up point coordinates (miles)		Delivery point coordinates (miles)	
	X	Y	X	Y
1	1	2	5	2
2	3	1	1	8
3	5	4	2	6
4	6	3	4	7
5	10	8	11	6
6	8	1	7	5

In addition, we shall assume that the following chronological sequence of additional customer requests (the same for both cases) will occur:

Time of request (minutes from $t=0$)	Customer	Pick-up Point Coordinates (miles)		Delivery Point Coordinates (miles)	
		X	Y	X	Y
20	7	4	6	11	3
40	8	7	7	9	4
55	9	4	1	5	3
80	10	5	4	7	4

It is our assumption, of course, that we are not able to anticipate these requests ahead of time and act accordingly. These requests become part of our input only after the customer files a request at the indicated time.

Figures 9.2 and 9.7 represent the tentative optimal routes for Cases 1 and 2 respectively, for the first update at $t=0$. Note that these routes are not identical. (All figures appear at the end of the Chapter.)

At $t=20$ minutes, customer 7 appears. At that time the vehicle is at point (7.741, 1.259) for Case 1 and at point (4.418, 4.388) for Case 2. These points are marked by a * on our tentative routes (Figs. 9.2, 9.7).

Figures 9.3 and 9.8 show the new tentative optimal routes for Cases 1 and 2 respectively, for this second update at $t=20$ minutes (without loss of generality, we assume that the time to process the new input and produce the new solution is negligibly small). The portion of the route between $t=0$ and $t=20$ has already been executed and is shown by dotted lines in these figures.

Before proceeding to the next update, we make the following observations in comparing Case 1 to Case 2:

1) Because of the difference of the optimal solutions between our first update of Cases 1 and 2 (Figs. 9.2, 9.7), the problem configuration we are facing in the second update ($t=20$ mins.) is not the same for the two cases. In fact, Customer 6 has already been picked up for Case 1 while in Case 2 this pick up is still tentative (Figs. 9.3, 9.8). The opposite happens for customers 3 and 4. Thus, it makes little sense in checking the measures of performance between the second updates of Cases 1 and 2, since the problems corresponding to each case for these updates are totally different. We shall come back to this point later.

2) The appearance of Customer 7 at $t=20$ mins. has produced changes' in the solutions between the first two updates for both cases. Note however a difference between Cases 1 and 2; In Case 1, the vehicle does not change course at the instant of the update (Compare Figs. 9.2 and 9.3). Rather the change is observed in later legs of the route (After the delivery of Customer 4). By contrast, in Case 2, the vehicle changes course at the instant of the update, to pick up the new Customer 7 (Compare Figs. 9.7 and 9.8). Note also the change in the delivery of Customer 4 in this Case.

Let us now examine our third update. At time $t=40$ minutes, customer 8 appears. At that time the vehicle is at point (1.267, 7.911) for Case 1 or at point (3.863, 8.000) for Case 2. We again mark these points with a * on our tentative routes (Figs. 9.3, 9.8).

Figures 9.4 and 9.9 show the new tentative optimal routes for Cases 1 and 2 respectively, for this third update at $t=40$ minutes. Again, dotted lines show the portion of the route between $t=20$ and $t=40$ which has already been executed.

Our fourth update occurs at $t=55$ minutes, when Customer 9 appears. At that time the vehicle is at point (7.000, 5.457) for Case 1 or at point (10.467, 7.066) for Case 2. These points are shown by a * in Figs. 9.4, 9.9.

In Figures 9.5 and 9.10 we show the new tentative optimal routes for Cases 1 and 2 respectively, for this fourth update at $t=55$ minutes. We keep our previously established convention about dotted lines.

Our fifth (and final) update occurs at $t=80$ minutes, when Customer 10 appears. At that time the vehicle is at point (8.724, 3.834) for Case

1 or at point (5.090, 1.000) for Case 2. These points are shown by a * in Figs. 9.5, 9.10.

In figures 9.6 and 9.11 we show the new tentative optimal routes for Cases 1 and 2 respectively, for this fifth update at $t=80$ minutes.

An observation we can make in Fig. 9.11 is that the delivery of Customer 6 precedes the delivery of Customer 10, only because of the $MPS=3$ constraint. In fact, if no such constraint existed, it would be better to delivery Customer 10 first and then Customer 6. The reader may have noticed already that in Figures 9.7, 9.8, 9.9, and 9.10 Customer 6 was always assigned to be delivered last. In our fifth update however, the MPS constraint prevents this deferment to continue and Customer 6 is finally given priority over a more favorable customer.

For our example, no more customers appear after 10, so the tentative-ly optimal routes will be finally executed. If this were not the case our updating process would be indefinitely pursued. The following tables summarize the final vehicle schedules:

<u>Case 1</u>			
Time	<u>Location (Miles)</u>		Description of Action
	X	Y	
0	1.000	4.000	Decision Point A_1 (update #1)
4.000	1.000	2.000	Pick-up 1
8.472	3.000	1.000	Pick-up 2
12.944	5.000	2.000	Delivery 1
19.269	8.000	1.000	Pick-up 1
20.000	7.741	1.259	Decision Point A_2 (update #2)

Case 1 (cont'd)

Time	<u>Location (Miles)</u>		Description of Action
	X	Y	
24.926	6.000	3.000	Pick-up 4
27.754	5.000	4.000	Pick-up 3
34.965	2.000	6.000	Deliver 3
39.437	1.000	8.000	Deliver 2
40.000	1.267	7.911	Decision Point A_3 (update #3)
45.762	4.000	7.000	Deliver 4
47.762	4.000	6.000	Pick-up 7
54.086	7.000	5.000	Deliver 6
55.000	7.000	5.457	Decision Point A_4 (update #4)
58.086	7.000	7.000	Pick-up 8
64.411	10.000	8.000	Pick-up 5
68.883	11.000	6.000	Deliver 5
74.883	11.000	3.000	Deliver 7
79.355	9.000	4.000	Deliver 8
80.000	8.724	3.834	Decision Point A_5 (update #5)
91.017	4.000	1.000	Pick-up 9
95.489	5.000	3.000	Deliver 9
97.489	5.000	4.000	Pick-up 10
101.489	7.000	4.000	Deliver 10

Case 2

Time	<u>Location (miles)</u>		Description of Action
	X	Y	
0	1.000	4.000	Decision point A_1 (update #1)
4.000	1.000	2.000	Pick-up 1
8.472	3.000	1.000	Pick-up 2
12.944	5.000	2.000	Deliver 1
15.773	6.000	3.000	Pick-up 4
18.601	5.000	4.000	Pick-up 3
20.000	4.418	4.388	Decision point A_2 (update #2)
23.331	4.000	6.000	Pick-up 7
25.331	4.000	7.000	Deliver 4
29.803	2.000	6.000	Deliver 3
34.275	1.000	8.000	Deliver 2
40.000	3.863	8.000	Decision point A_3 (update #3)
46.586	7.000	7.000	Pick-up 8
52.910	10.000	8.000	Pick-up 5
55.000	10.467	7.066	Decision point A_4 (update #4)
57.383	11.000	6.000	Deliver 5
63.383	11.000	3.000	Deliver 7
67.855	9.000	4.000	Deliver 8
74.179	8.000	1.000	Pick-up 6
80.000	5.090	1.000	Decision point A_5 (update #5)
82.179	4.000	1.000	Pick-up 9
86.651	5.000	3.000	Deliver 9

Case 2 (cont'd)

Time	<u>Location (miles)</u>		Description of Action
	X	Y	
88.651	5.000	4.000	Pick-up 10
93.123	7.000	5.000	Deliver 6
95.123	7.000	4.000	Deliver 10

A surprising observation we can immediately make, is the fact that Case 1 yields a longer route time than Case 2, despite the fact that time minimization is the objective of Course 1 and not of Case 2!

Is the algorithm wrong therefore?

The answer is no. The reason is that it does not make sense to check the "optimality" of an executed route a posteriori. We put the word optimality in quotation marks, because it is important to understand what "optimal route" means for our dynamic problem. The fact that at each update we optimize only over the known customer requests makes any such combination of updates in general sub-optimal. If we had a way to anticipate the customer requests at $t=20, 40, 55$ and 80 mins, it would certainly be necessary for us to take advantage of this information and it would certainly make sense to compare our results afterwards. But since the nature of our algorithm is by construction "myopic" (i.e. nothing is anticipated and no action is taken until the request appears), its objective cannot be to optimize any measure of performance which cannot be evaluated a priori, but only a posteriori.

We shall discuss such an extension of this algorithm briefly in

Chapter 11.

We conclude this Chapter by giving (just for the sake of comparison and not to verify or reject the validity of our algorithm) the values of the total passenger disutilities for both Cases. These are 412.293 and 370.871 passenger-minutes for Case 1 and 2 respectively.

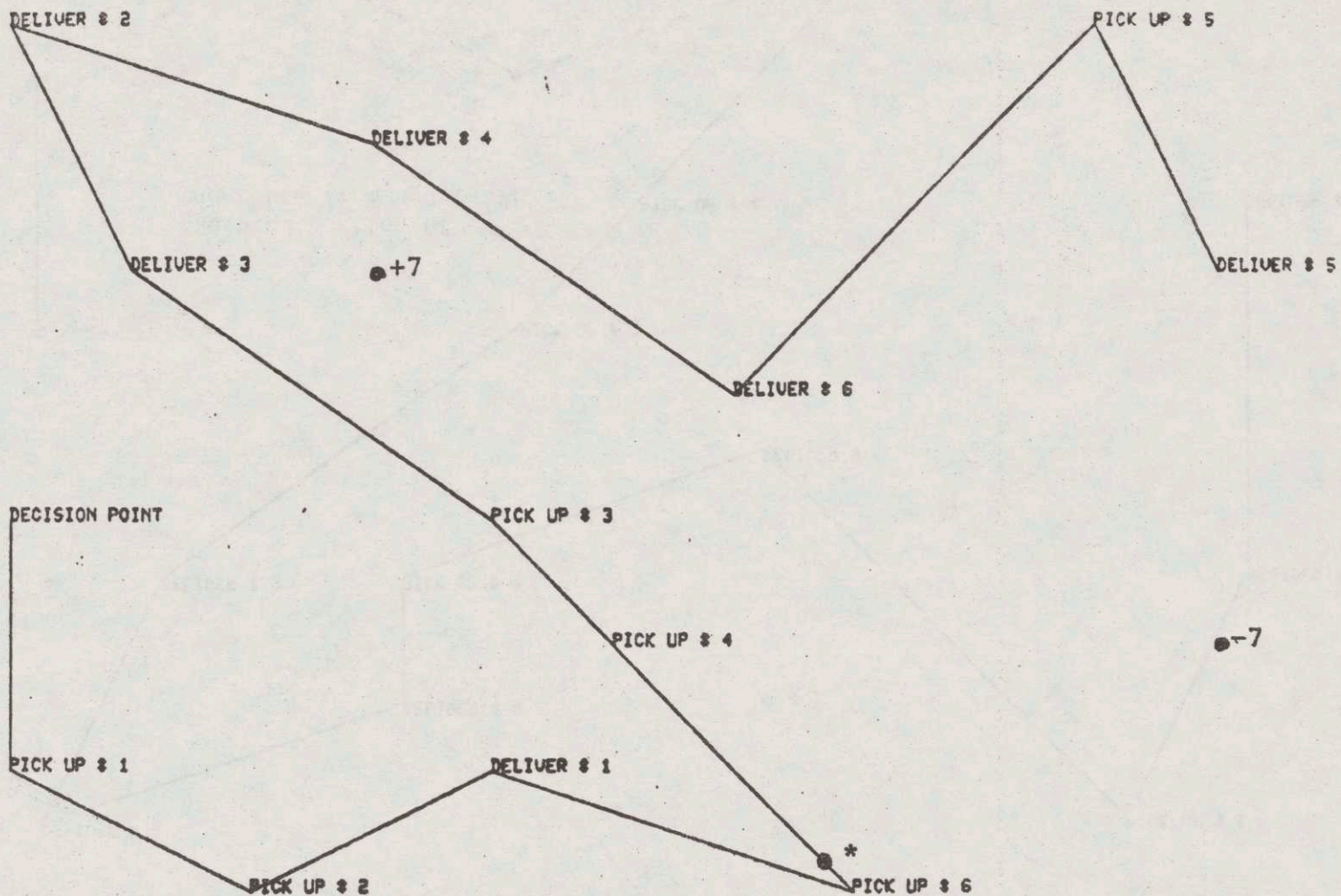


Fig. 9.2 : Case 1, update #1. Customer 7 appears when vehicle is at *.

Fig. 9.3: Case 1, update #2. Customer 8 appears when vehicle is at *.

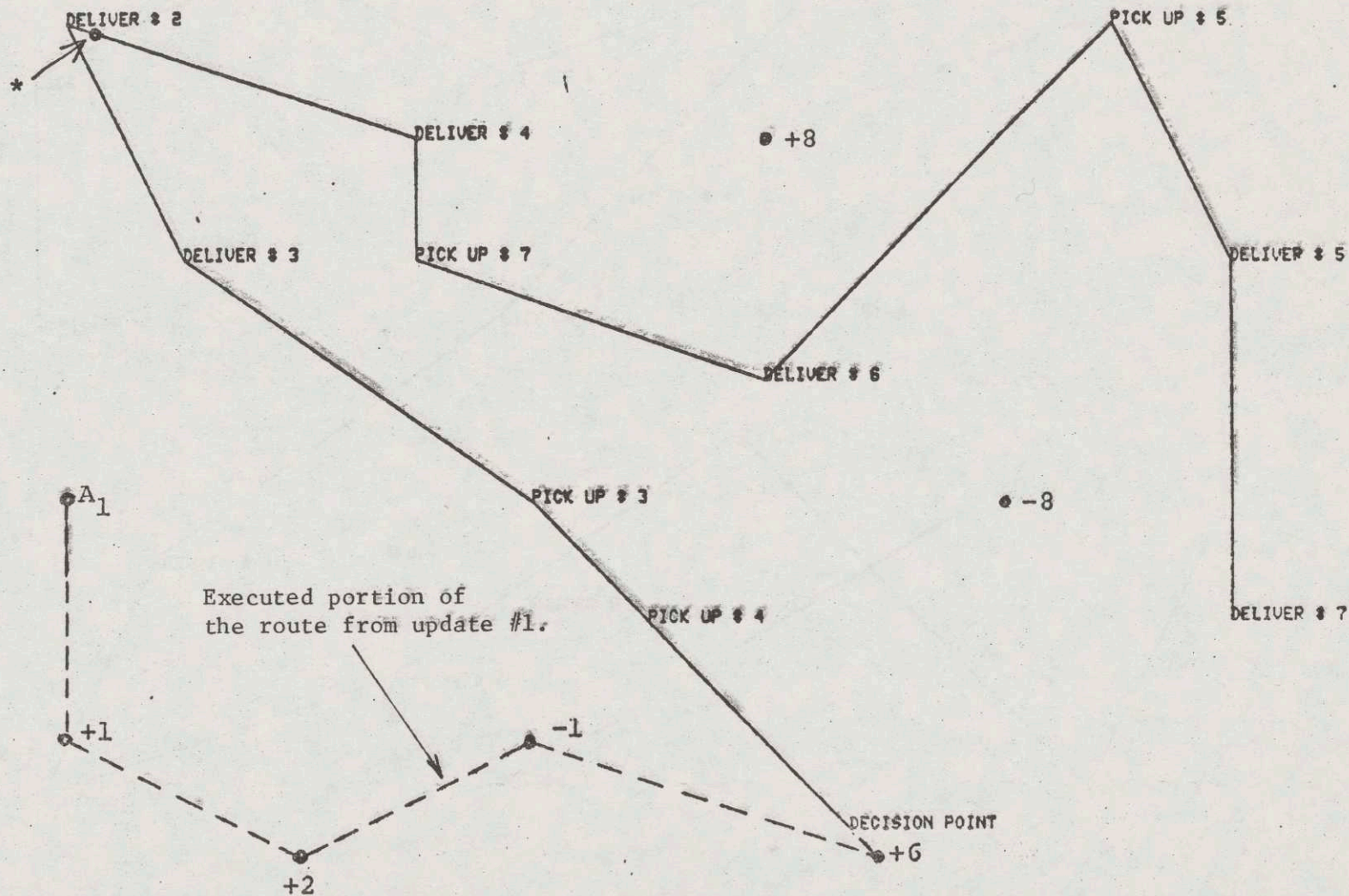


Fig. 9.4 : Case 1, update #3. Customer 9 appears when vehicle is at *.

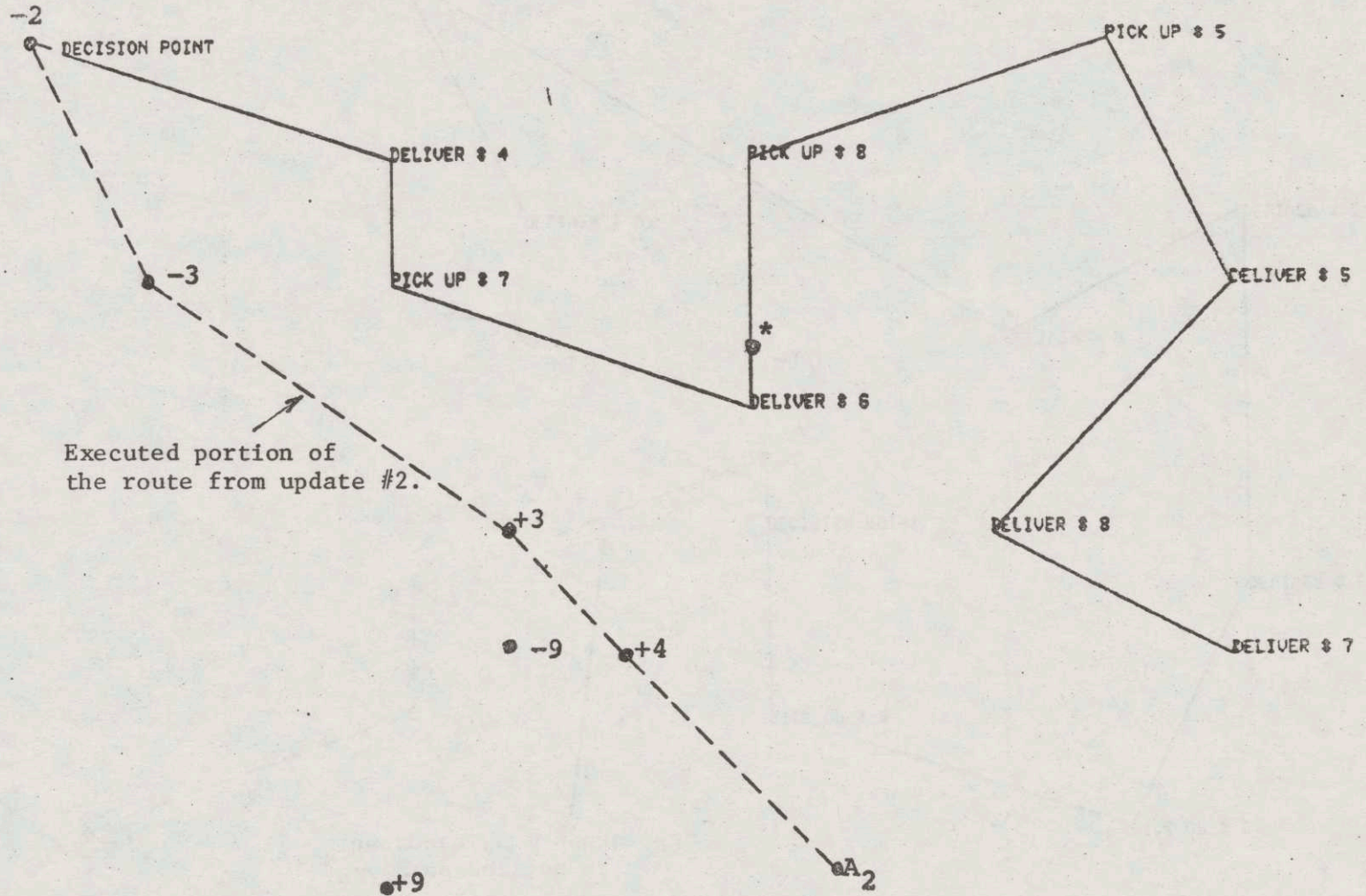


Fig. 9.5 : Case 1, update #4. Customer 10 appears when vehicle is at *.

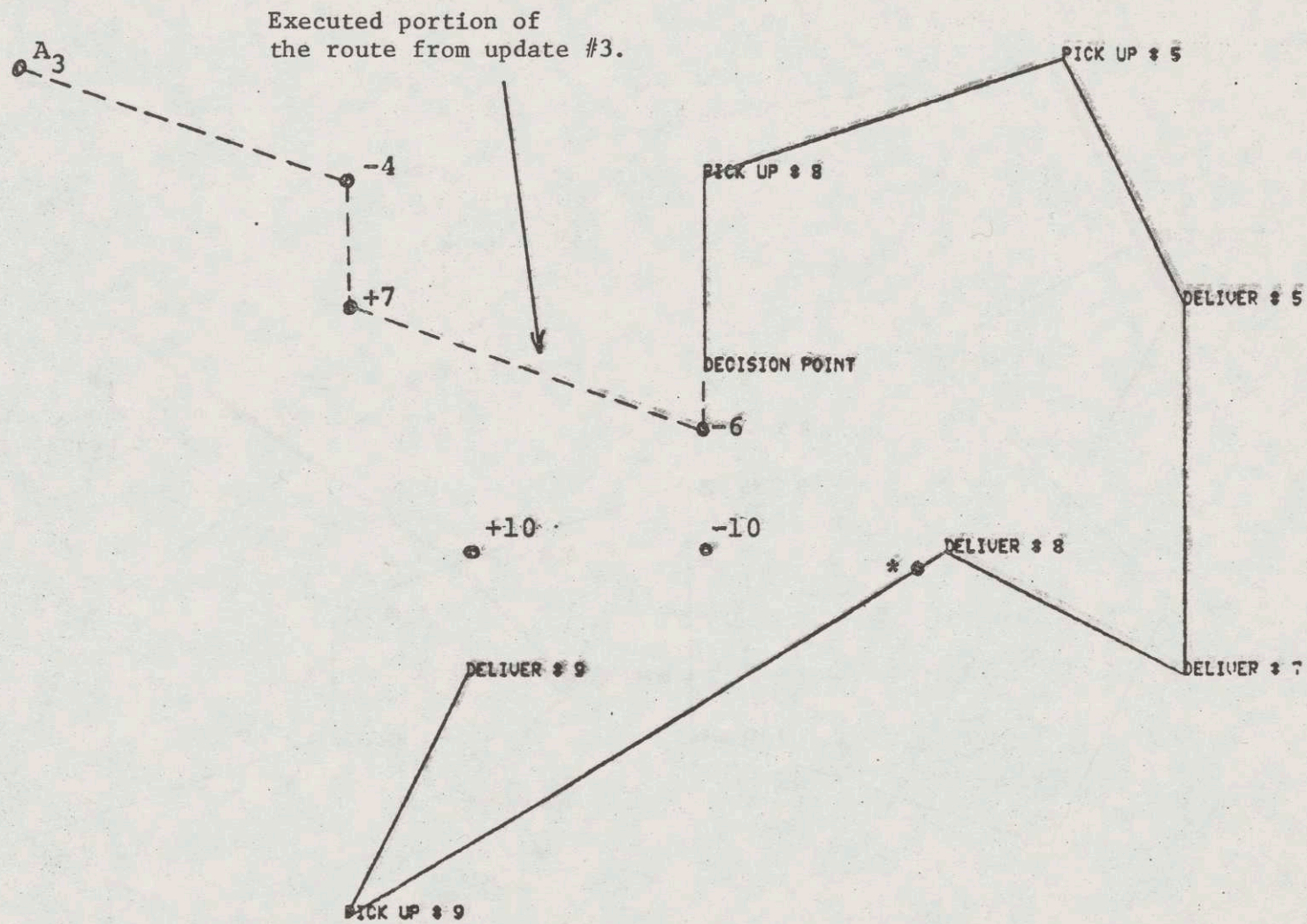
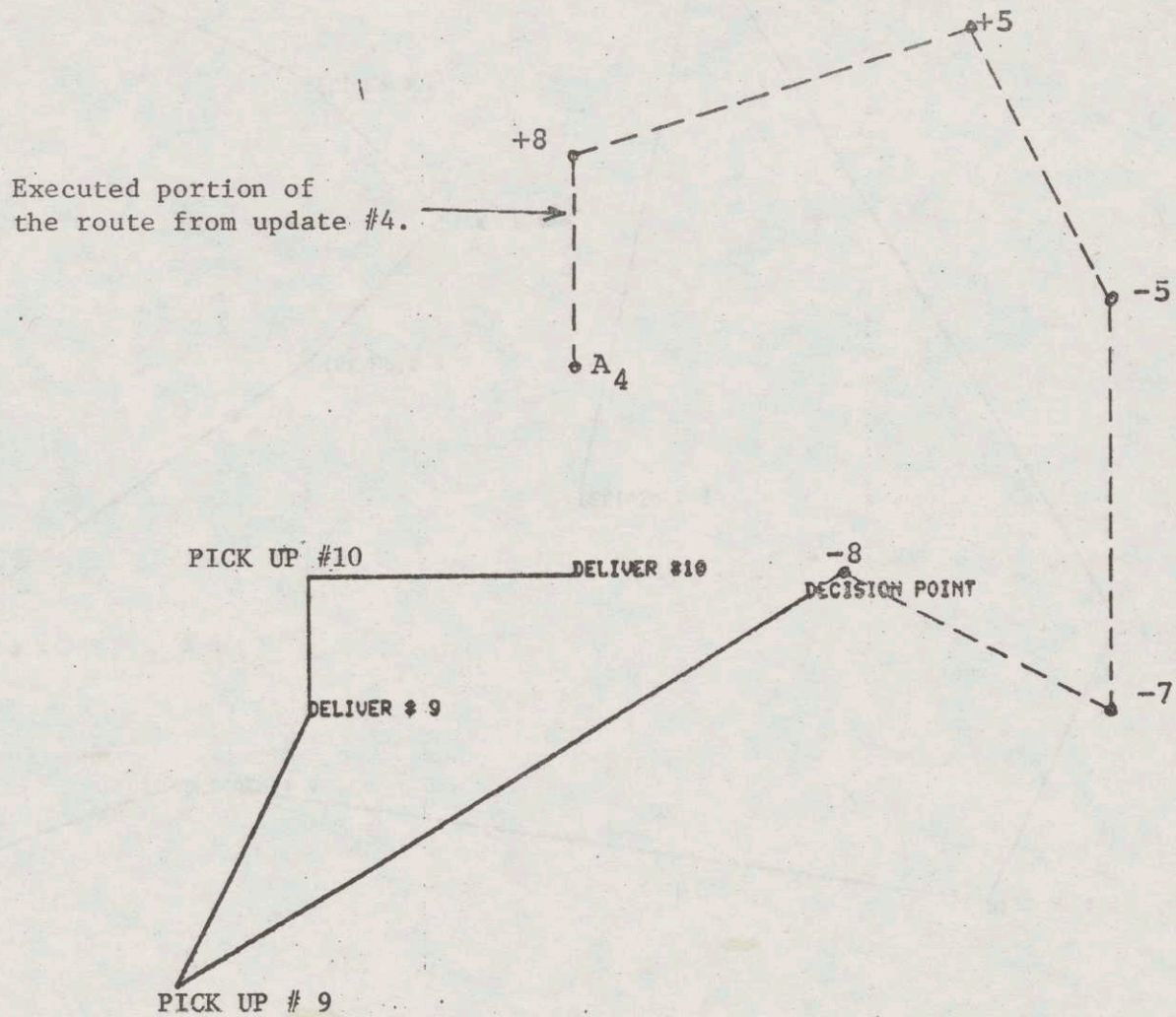


Fig. 9.6 : Case 1, update #5. No more customers will appear.



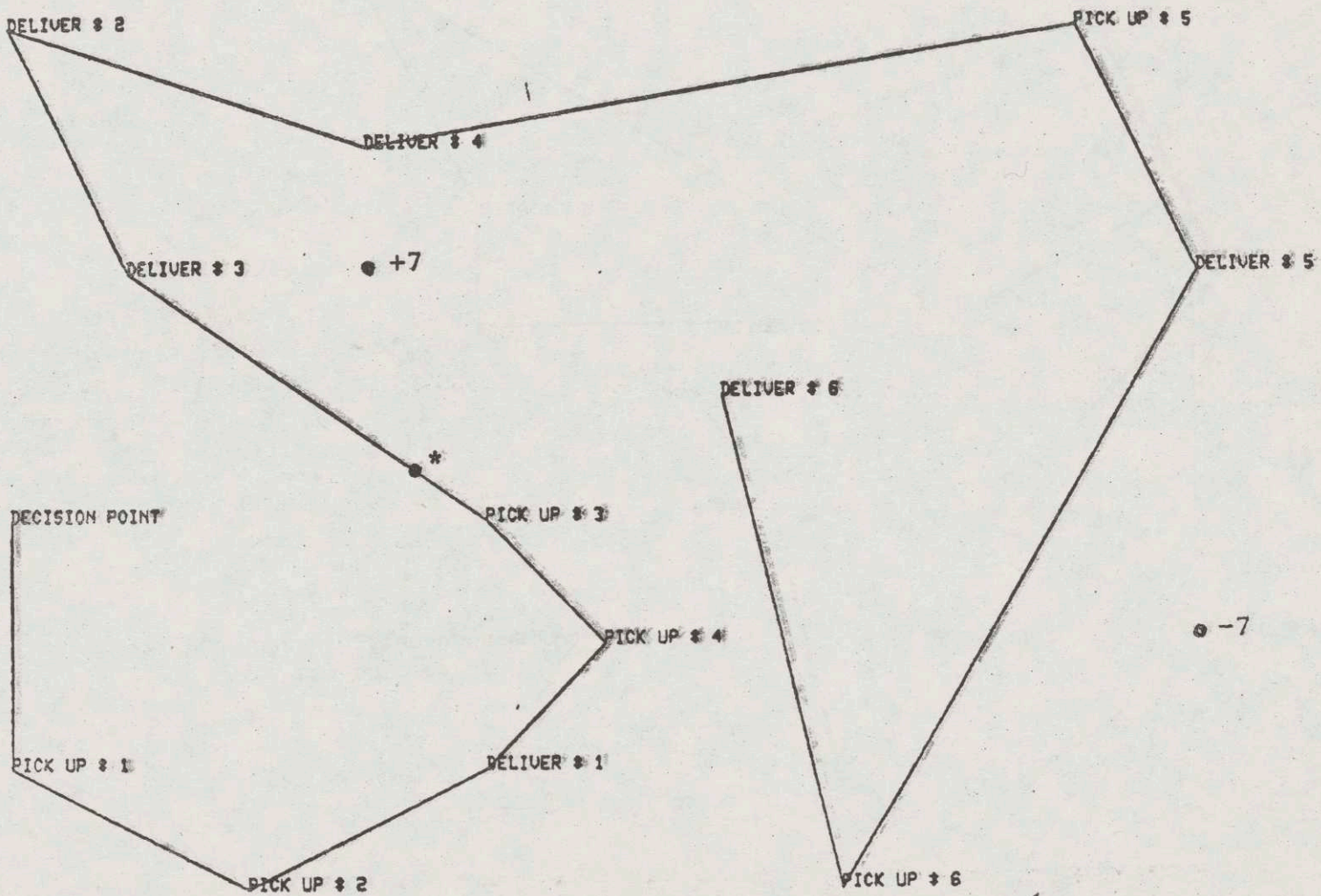


Fig. 9.7 : Case 2, update #1. Customer 7 appears when vehicle is at *.

Fig. 9.8 : Case 2, update #2. Customer 8 appears when vehicle is at *.

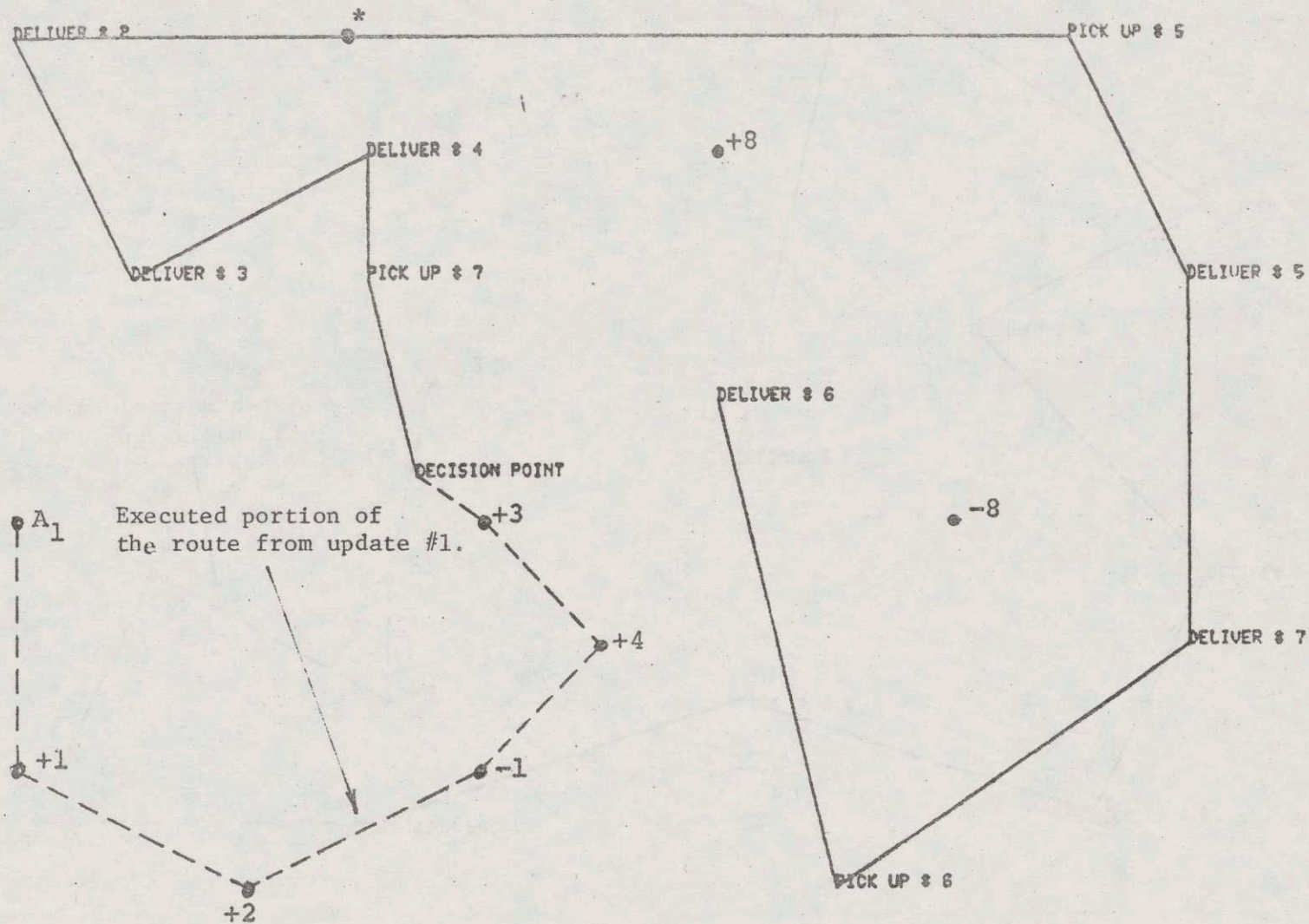


Fig. 9.9 : Case 2, update #3. Customer 9 appears when vehicle is at *.

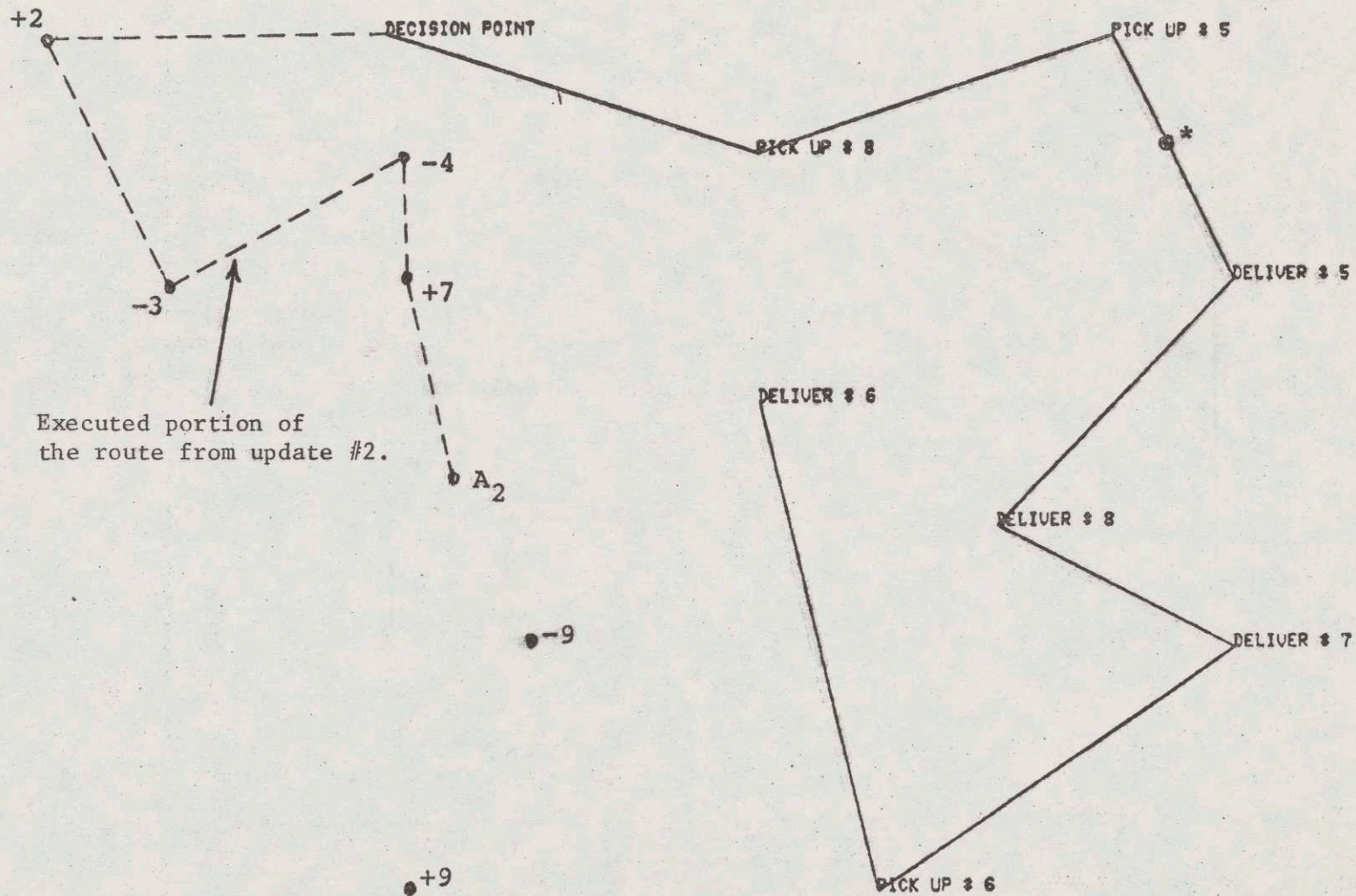
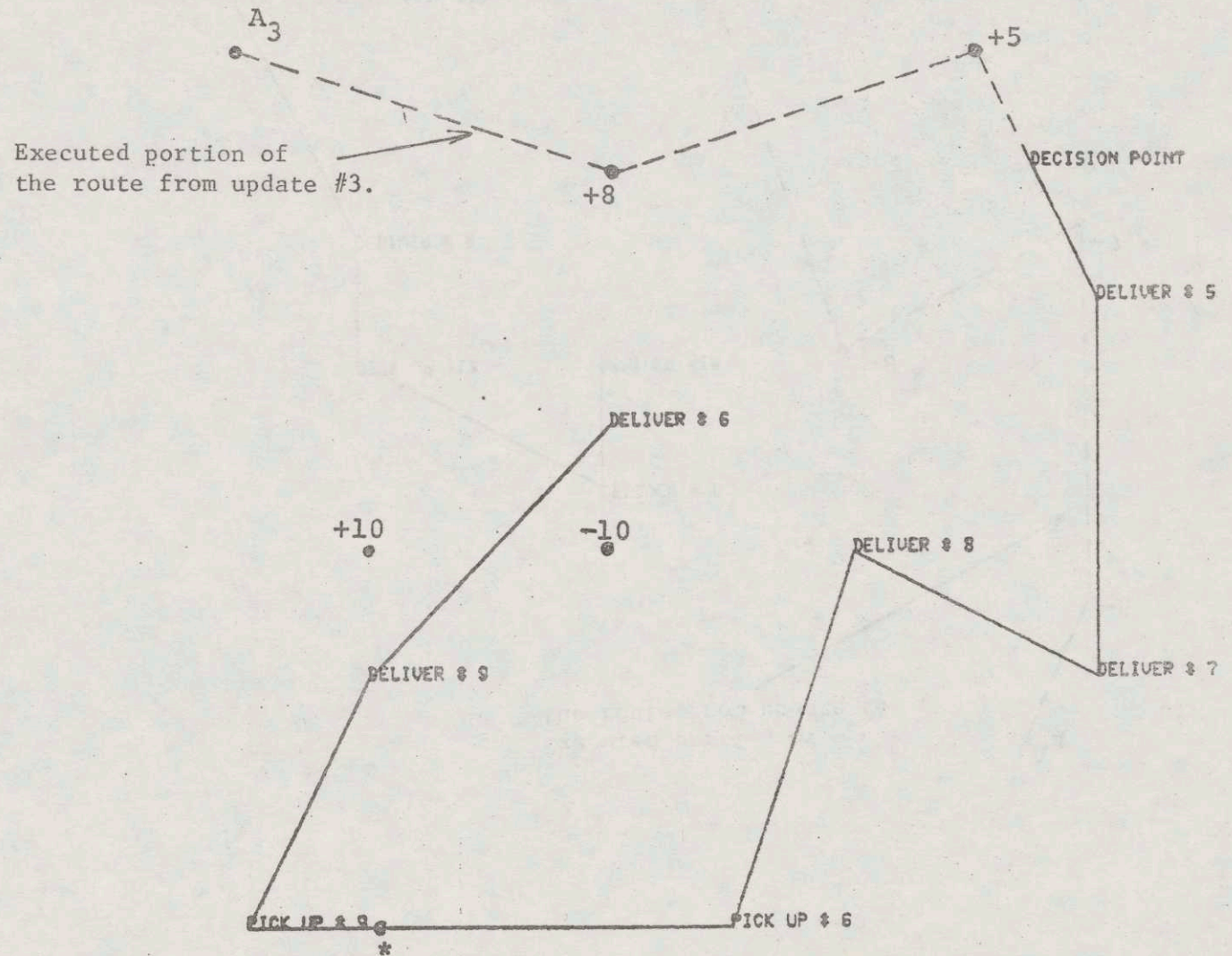


Fig. 9.10 : Case 2, update #4. Customer 10 appears when vehicle is at *.



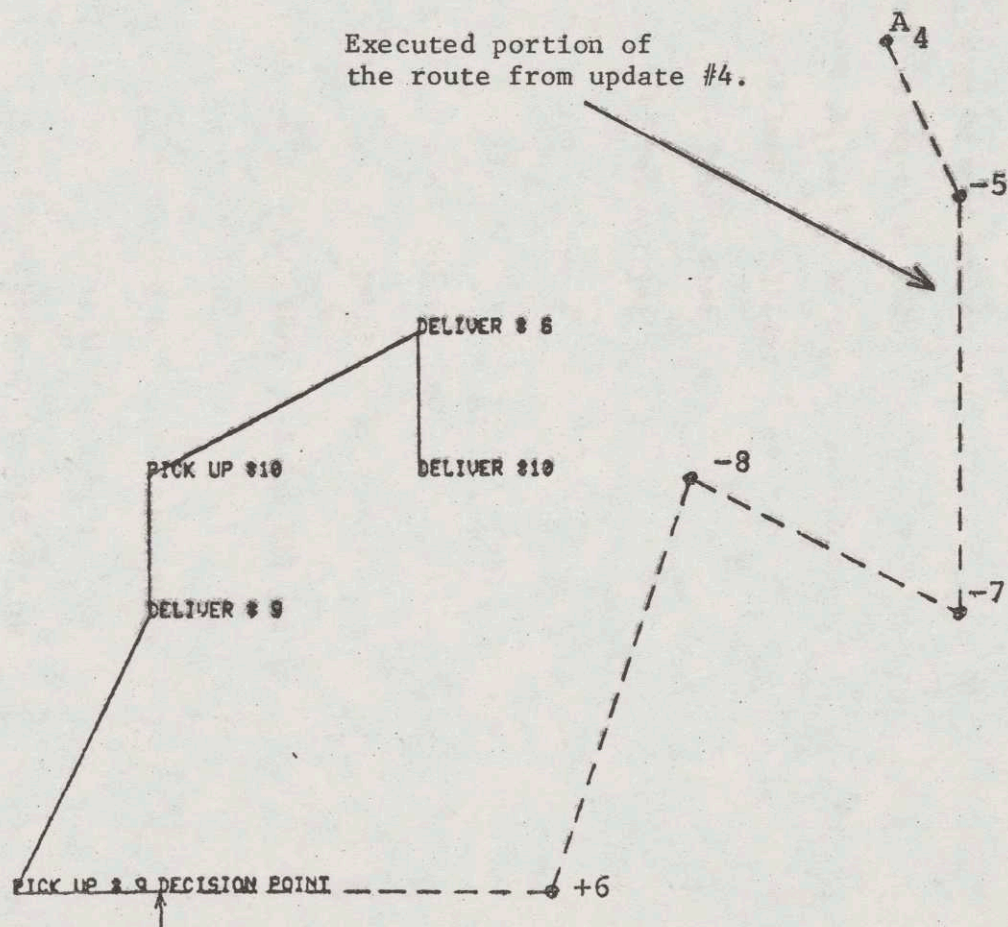


Fig. 9.11 : Case 2, update #5. No more customers will appear.

See text for role of MPS constraints on the delivery of customer 6.

CHAPTER 10

IMPROVING THE COMPUTATIONAL EFFICIENCY OF THE DIAL-A-RIDE ALGORITHM

10.0 Introduction

The purpose of this chapter is to investigate issues connected with the combinatorics and computational complexity of the "static" version of the dial-a-ride problem and to try to see if one can use the results of this investigation in order to improve upon the storage requirements or computational effort of the algorithm proposed so far.

In Chapter 8 it was seen that the computational effort of the algorithm and the corresponding storage requirements were exponential functions of the size N of the input. In fact, our proposed algorithm was seen to need storage space for $(2N+1)3^N$ states, and three pieces of information were tabulated for each state: The values of the logical variable FEASBL, for feasibility, the real variable V , for the optimal value function and finally the integer variable NEXT, for the best next stop. The growth rate of the computational effort of the algorithm was seen to be bounded from above by the above exponential function, actually depending on how tight the constraints were. The exact way the constraints influence the computational effort was not investigated in Chapter 8.

10.1 Reducing Storage Requirements by a Factor of Three

It has already been pointed out in Chapter 8 that our state representation (L, k_1, \dots, k_N) exhibits certain redundancies which create inconsistencies for various states. We saw in fact that only $2N3^{N-1} + 1$ states out of the

$(2N+1)3^N$ are consistent. The corresponding ratio is asymptotically 1 to 3 which means that if we keep the (L, k_1, \dots, k_N) representation, then we have to use three times as much storage space than we actually need. The following is a simple way of avoiding such a waste of storage space:

We replace (L, k_1, \dots, k_N) by a new non-redundant state representation (L, m_1, \dots, m_{N-1}) defined as follows:

1) if $1 \leq L \leq N$ (which means that $k_L=2$)

then $m_j = k_j$ for all $j=1$ to $L-1$

and $m_j = k_{j+1}$ for all $j=L$ to $N-1$

2) If $N+1 \leq L \leq 2N$ (which means that $k_{L-N}=1$)

then $m_j = k_j$ for all $j=1$ to $L-N-1$

and $m_j = k_{j+1}$ for all $j=L-N$ to $N-1$

3) If $L=2N+1$ (which means that $k_j=3$ for all $j=1$ to N)

then $m_j=3$ for all $j=1$ to $N-1$

(Actually we don't need to specify what the m_j 's are here, since,

if $L=2N+1$, we know that the vehicle is at the starting point A).

A very simple comparison between the old and new state representations and an illustration of the savings in storage space created is the following ($N=2$):

<u>OLD REPRESENTATION</u>			<u>NEW REPRESENTATION</u>
(L, k_1, k_2)			(L, m_1)
No. of states = 45			No. of states = 13
(1,1,1)	(2,3,1)	<u>(4,2,1)</u>	<u>(4,2)</u>
(1,1,2)	<u>(2,3,2)</u>	(4,2,2)	<u>(2,3)</u>

<u>OLD REPRESENTATION</u>			cont'd			<u>NEW REPRESENTATION</u>		
(L, k_1, k_2)						(L, m_1)		
No. of states = 45						No. of states = 13		
(1,1,3)	(2,3,3)	(4,2,3)						
<u>(1,2,1)</u>	<u>(3,1,1)</u>	<u>(4,3,1)</u>				<u>(1,1)</u>	<u>(3,1)</u>	<u>(4,3)</u>
<u>(1,2,2)</u>	<u>(3,1,2)</u>	(4,3,2)				<u>(1,2)</u>	<u>(3,2)</u>	
<u>(1,2,3)</u>	<u>(3,1,3)</u>	(4,3,3)				<u>(1,3)</u>	<u>(3,3)</u>	
(1,3,1)	(3,2,1)	(5,1,1)						
(1,3,2)	(3,2,2)	(5,1,2)						
(1,3,3)	(3,2,3)	(5,1,3)						
(2,1,1)	(3,3,1)	(5,2,1)						
<u>(2,1,2)</u>	(3,3,2)	(5,2,2)				<u>(2,1)</u>		
(2,1,3)	(3,3,3)	(5,2,3)						
(2,2,1)	<u>(4,1,1)</u>	(5,3,1)					<u>(4,1)</u>	
<u>(2,2,2)</u>	(4,2,1)	(5,3,2)				<u>(2,2)</u>		
(2,2,3)	(4,1,3)	<u>(5,3,3)</u>						<u>(5,3)</u>

We have underlined the consistent states.

We can see the savings we can realize if we use the new representation. All 13 states of the new representation are consistent, while only 13 out of 45 states of the old one were.

It should be noted at this point that the disadvantage of the representation (L, k_1, \dots, k_N) is not due to the representation itself, but is connected to the way we reserve storage space for it. In fact, if we had devised a bookkeeping procedure to store only the information connected to the consistent states of (L, k_1, \dots, k_N) , this would be exactly the same as using the new representation (L, m_1, \dots, m_{N-1}) . In that respect therefore

the two representations are exactly equivalent and reducible to one another in time proportional to N . We shall see later another way to tabulate state information, in order to reduce storage requirements even further.

10.2 Preliminary Observations Concerning the Structure of the Problem.

Attempting to shed additional light into the structure of our problem, we construct the decision graph of our previous example (Fig. 10.1) This will motivate questions that will be extensively investigated later in this Chapter.

The decision graph for our problem is a structured way to represent the consistent states as well as how these are linked to one another. The graph is directed. Any path starting from the root of the graph (which represents our starting point A) and ending to one of the terminal nodes (which represent the possible ways our route can end) is a legitimate sequence of pick-up and delivery stops. Referring to Fig. 10.1 we can make the following observations.

1) the "stratification" of the decision graph is immediately noticeable. In fact, we can classify its nodes into 5 stages, 0 to 4, as shown. At this point we recall that the dynamic programming solution procedure we developed in Chapter 8 did not use the stage concept at all.

2) In addition to the above stratification, the graph exhibits a distinct, although not yet precisely understood, branching structure. With the exception of the arcs emanating from the root, one can note a symmetry in the remaining graph, in the sense that the early stages of it exhibit similar branching characteristics with the late stages and that the same holds for intermediate stages. Also, the breadth (number of states per

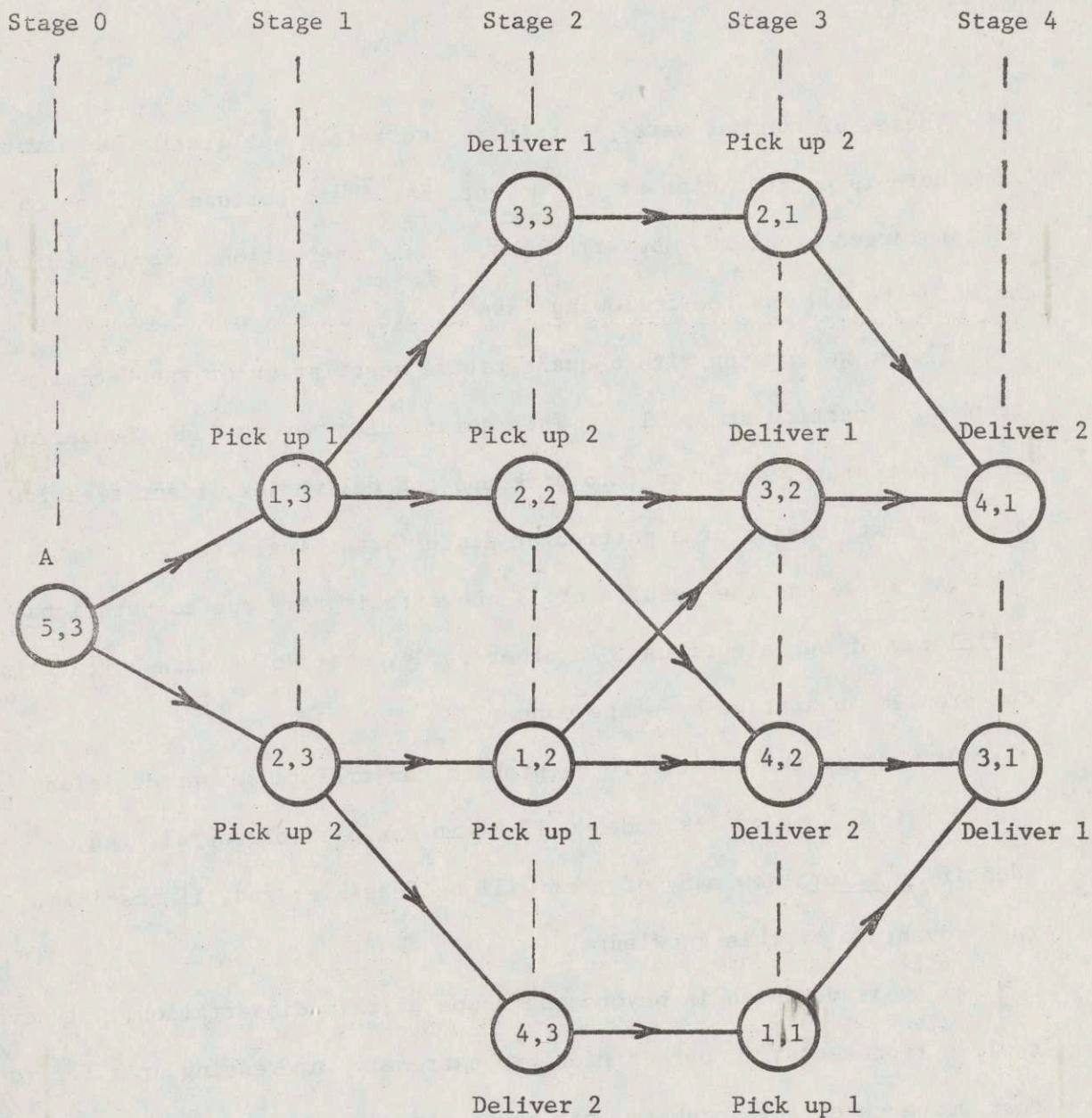


Fig. 10.1 : Decision graph of the dial-a-ride problem for $N=2$.

Numbers at nodes denote the corresponding states (L, m_1) .

stage) increases until halfway through the graph and then decreases symmetrically.

These, of course, were preliminary and mainly qualitative observations and there is no guarantee at the moment that these patterns will be observed for larger problems. Nevertheless, these observations provide sufficient stimulus to address the following issues:

1) Can we come up with a quantitative description of the decision graph of a particular problem? This description may include: Number of states per stage, division into pick-ups and deliveries, identification of the "next" states of a particular state (branching) etc.

2) Can we use the results of 1) above to improve the computational efficiency of our algorithm? In other words, what do we gain by tackling our problem in a stage-by-stage manner?

3) What is the effect of the problem constraints on our decision graph? Since some of its nodes will be infeasible in general, can we identify a priori how many of them will be feasible, and, if possible, take advantage of this knowledge?

4) An issue which is beyond the scope of this dissertation, but nevertheless represents, in our opinion, an extremely interesting area for further research on this problem, can be addressed if our knowledge on the structure of this problem reaches a certain level:

Can we increase the computational efficiency of the algorithm even more, by introducing heuristic rules in the branching process? It is understood of course that such an approach would possibly imply reaching non-optimal solutions. We shall return to this aspect of the problem in

Chapter 11.

The section that follows is devoted to a detailed investigation of the combinatorial structure and related issues for the dial-a-ride problem.

10.3 Stage-by-stage solution procedure

Throughout this section, and for convenience, we shall use our initial state representation (L, k_1, \dots, k_N) . As we mentioned earlier, this representation is as effective as any other one, provided no storage space is reserved for inconsistent states.

Let us examine the optimality recursion, as we already have presented it:

$$V(L, k_1, \dots, k_N) = \min_{x \in X} [M.t_{L,x} + V(x, k'_1, \dots, k'_N)]$$

$$\text{with } k'_j = \begin{cases} k_j - 1 & \text{if } j=x \text{ or } j=x-N \\ k_j & \text{otherwise} \end{cases}$$

for $j=1, \dots, N$

(L, k_1, \dots, k_N) and (x, k'_1, \dots, k'_N) are states corresponding to successive stages $n, n+1$ of our problem. (This property will be rigorously proven later.) Thus, we could in principle evaluate all states (L, k_1, \dots, k_N) of a certain stage n at the same step of our algorithm, taking advantage of the fact that all "next" states (x, k'_1, \dots, k'_N) belong to the same stage $n+1$ as well. Referring to Fig. 10.1 this would mean that we can for example evaluate the 4 states of stage 2 at the same step of the algorithm, because all states which are "next" to these 4 states belong to the same

stage 3. We shall show how n is connected to (L, k_1, \dots, k_N) later.

What advantages do we gain by solving the problem in a stage-by-stage manner? (As we already mentioned, the algorithm we have presented in Chapter 8 does not solve the problem in this way.)

To answer this question, we observe that if we solve the problem stage-by-stage, then the values of $V(x, k'_1, \dots, k'_N)$ appearing on the right-hand-side of our recursion are only used at one step of our procedure and become useless thereafter. At the same step of our algorithm, the values of $V(L, k_1, \dots, k_N)$ appearing on the left-hand-side of our recursion are created. These values will appear on the right-hand-side of the recursion at the subsequent step of the algorithm and will also become useless thereafter. Thus, we can see that we essentially need two arrays for V , one for the left-hand-side values which are created (say, for stage n) and one for the right-hand-side values which are used (stage $n+1$). After this particular step of our algorithm is executed, the right-hand-side values can be discarded. Moving backwards to the next step, i.e. creating now the V values of stage $n-1$, we can copy the values of V which were occupying the left-hand-side array to the current right-hand-side array and proceed similarly. Doing this we don't have to reserve space for the entire table of elements of V , elements which will essentially be utilized a limited number of times each and then become useless. To give a simple illustration, we consider our previous example for $N=2$. In our first proposed algorithm, we had to reserve 45 storage locations for V . If we do not reserve space for the inconsistent states, we can drop this number to 13. This is the total number of nodes of our decision graph (Fig. 10.1). By solving our problem stage-by-stage however, we only need two arrays of size 4 each,

namely 8 storage locations total since the maximum number of states per stage is 4 (this is the maximum breadth of our decision graph).

We see therefore that the main advantage of the stage-by-stage solution approach is the more efficient use of storage space. Several observations are important at this point:

1) By no means can we apply the above "create, copy and erase" procedure to the array NEXT as well, since, if we do that, we shall not be able to perform our identification procedure. While the nature of the array V allows us to discard the right-hand-side elements after their use in each particular step of the algorithm, no criterion exists which will enable us to tell a priori whether a particular element of the array NEXT is useless, i.e. will not be used in the identification part. Thus, we should continue to keep all relevant information on this array, either in main or in auxiliary storage.

2) Concerning the logical array FEASBL, we can totally dispose of it if we merge the "Screening" and "Optimization" procedures of the algorithm we have already proposed in Chapter 8. Thus, if a state is infeasible, we can simply put $V=-1$. A value of $V \neq -1$ will signify that the state is feasible.

3) The savings in storage requirements to be realized by the new stage-by-stage solution approach will not be without cost. We shall see that the enumeration and indexing of all consistent states inside a given stage will constitute a non-trivial problem. This is mainly due to the fact that the number of consistent states per stage (i.e. the breadth of our decision graph) is not constant. A significant effort will be made later in this chapter to overcome this difficulty.

10.4 Combinatorics of the Dial-A-Ride Problem

The purpose of this section is to discuss how we can enumerate all states (L, k_1, \dots, k_N) that belong to a given stage n and what happens when we impose capacity and priority constraints.

To begin with, we need to know how n is defined mathematically and how it is connected with the state vector (L, k_1, \dots, k_N) .

For a particular route, each stage corresponds to a particular stop of the vehicle, pick-up or delivery. We set $n=0$ when the vehicle is at its starting point A , $n=1$ when it picks up its first customer, and so on until finally $n=2N$ when it delivers its last customer.

For a particular consistent state (L, k_1, \dots, k_N) , let x_1 be the number of customers already delivered (their k is equal to 1) and x_2 the number of customers in the vehicle (their k is equal to 2).

It is easy to see that if n is the stage to which (L, k_1, \dots, k_N) belongs, then

$$n = 2x_1 + x_2 \tag{10.1}$$

In fact, if x_1 customers have been delivered, the number of stops the vehicle made for them should be $2x_1$ (x_1 pick-up stops and x_1 delivery stops). Also, if x_2 customers are in the vehicle, then x_2 additional (pick-up) stops were made. Since n can be interpreted as the total number of stops the vehicle has made to reach the state (L, k_1, \dots, k_N) , formula (10.1) follows.

Defining also x_3 as the number of customers still waiting for the vehicle (their k is equal to 3), we obviously have:

$$x_1 + x_2 + x_3 = N \quad (10.2)$$

From (10.1), (10.2) we can obtain x_2 and x_3 in terms of N , n and x_1 :

$$x_2 = n - 2x_1 \quad (10.3)$$

$$x_3 = N - n + x_1 \quad (10.4)$$

The usefulness of the last two relations will be seen later.

It is clear that not all combinations of n and x_1 will in general make sense. Since $0 \leq x_j \leq N$ for $j=1,2,3$, we shall have:

$$0 \leq x_1 \leq N \quad (10.5)$$

$$0 \leq n - 2x_1 \leq N \quad (10.6)$$

$$0 \leq N - n + x_1 \leq N \quad (10.7)$$

The above three relations establish the following range for x_1 :

Range of x_1 at stage n :

<u>Lower Bounds</u>		<u>Upper bounds</u>
0	from (10.5)	N from (10.5)
$\frac{n-N}{2}$	from (10.6)	$\frac{n}{2}$ from (10.6)
$n-N$	from (10.7)	n from (10.7)

Thus, $\text{Max}[0, \frac{n-N}{2}, n-N] \leq x_1 \leq \text{Min}[N, \frac{n}{2}, n]$

Since $\frac{n-N}{2} \leq n-N$ if $n-N \geq 0$, since $\frac{n}{2} \leq n$, and since $\frac{n}{2} \leq N$, the above is exactly equivalent to:

$$\text{Max}[0, n-N] \leq x_1 \leq \frac{n}{2} \quad (10.8)$$

The above relation is very useful in that it establishes bounds for the variable x_1 , given a particular stage n . Variables x_2 and x_3 can then be derived using (10.3) and (10.4).

Some verifications follow:

1) $n=0$ (vehicle at starting point A)

Then (10.8) gives $\text{Max}[0, -N] \leq x_1 \leq 0$, or $x_1=0$.

From (10.3) we obtain that $x_2=0$ and from (10.4) that $x_3=N$. These results are as expected.

2) $n=1$ (vehicle at first stop)

Then (10.8) gives $\text{Max}[0, 1-N] \leq x_1 \leq \frac{1}{2}$

But $1-N \leq 0$ for $N \geq 1$, thus $0 \leq x_1 \leq \frac{1}{2}$ and since x_1 is integer, it follows that $x_1=0$.

From (10.3) we obtain that $x_2=1$ and from (10.4) that $x_3=N-1$. These results can be interpreted to mean that the first stage of the route has to be a pick-up stop with no customers delivered so far.

3) $n=2N$ (vehicle at last stop)

Then (10.8) gives $\text{Max}[0, 2N-N] \leq x_1 \leq N$, or $x_1=N$

From (10.3) we obtain that $x_2=0$ and from (10.4) that $x_3=0$. The interpretation of these results is obvious.

It should be realized of course that the above were extreme cases where we could pinpoint x_1, x_2, x_3 only from the stage which we were considering. For intermediate stages (e.g. $n=N$), the bounds on x_1 are looser and there are more combinations.

A rigorous verification of the fact that the states (L, k_1, \dots, k_N) and

(x, k'_1, \dots, k'_N) , (as they appear in the optimality recursion), belong to successive states n , $n+1$ is the following:

From (10.1) we have $\delta n = 2\delta x_1 + \delta x_2$ where the operator δ signifies the change of the corresponding variable when we go from (L, k_1, \dots, k_N) to (x, k'_1, \dots, k'_N) .

Two cases are now possible:

a) x is a pickup stop. Then $\delta x_1 = 0$ and $\delta x_2 = 1$, (no additional customer is delivered but one additional customer gets into the vehicle). Thus $\delta n = 1$.

b) x is a delivery stop. Then $\delta x_1 = 1$ and $\delta x_2 = -1$ (one additional customer is delivered but one less customer gets into the vehicle). Thus $\delta n = 1$.

Since in both cases we go from n to $n + \delta n$, this means that in both cases we go from n to $n+1$.

We now proceed to find an expression for $Q_N(n)$, the number of consistent states for stage n of a problem of size N . We can initially put $Q_N(n) = P_N(n) + D_N(n)$ where $P_N(n)$ is the number of states corresponding to pick-up stops and $D_N(n)$ the number of states corresponding to delivery stops. It is conceivable that either $P_N(n)$ or $D_N(n)$ is zero. (In fact we already know that $P_N(2N) = D_N(1) = 0$).

To find $P_N(n)$ we examine all combinations in the state vector (L, k_1, \dots, k_N) with $1 \leq L \leq N$, which will produce the given value of n , according to (10.1) so that x_1, x_2, x_3 stay within their bounds.

If $1 \leq L \leq N$, then we know that there should be at least one k which is equal to 2 (k_L), thus an additional constraint is that $x_2 \geq 1$. For any value of L and for a given value of x_1 , the number of permutations of the

k-vector so that the number of k's equal to 1 is x_1 , is $\binom{N-1}{x_1}$. $N-1$ comes from the fact that only $N-1$ variables are "free", since we know that k_L is "frozen" to 2. For the remaining $N-x_1-1$ "free" variables, we know that x_2-1 of them are equal to 2. But from (10.3) we know that $x_2 = n-2x_1$, so we know that $n-2x_1-1$ of the remaining $N-x_1-1$ "free" variables take the value of 2. There are $\binom{N-x_1-1}{n-2x_1-1}$ such permutations.

Thus, for any value of L , to find the number of consistent pick-up states, we have to sum, over all possible values of x_1 , the expression

$$\binom{N-1}{x_1} \cdot \binom{N-x_1-1}{n-2x_1-1} = \frac{(N-1)!}{x_1! (n-2x_1-1)! (N-n+x_1)!}$$

The bounds on x_1 are initially given by (10.8). However, a minor modification for the upper bound is needed, since $x_2 \geq 1$. This means that $n-2x_1 \geq 1$ or that $x_1 \leq \frac{n-1}{2}$. Since also there are N possible pick-up stops, we end up with the following expression for $P_N(n)$:

$$\begin{aligned} P_N(n) &= \sum_{x_1=\text{Max}[0, n-N]}^{\frac{n-1}{2}} N \cdot \binom{N-1}{x_1} \binom{N-x_1-1}{n-2x_1-1} = \\ &= N! \cdot \sum_{x_1=\text{Max}[0, n-N]}^{\frac{n-1}{2}} \frac{1}{x_1! (n-2x_1-1)! (N-n+x_1)!} \end{aligned}$$

A similar argument for the evaluation of $D_N(n)$ produces

$$\begin{aligned} D_N(n) &= N \cdot \sum_{x_1=\text{Max}[1, n-N]}^{\frac{n}{2}} \binom{N-1}{x_1-1} \cdot \binom{N-x_1}{n-2x_1} = \\ &= N! \cdot \sum_{x_1=\text{Max}[1, n-N]}^{\frac{n}{2}} \frac{1}{(x_1-1)! (n-2x_1)! (N-n+x_1)!} \end{aligned} \quad (10.10)$$

One can recognize a similarity between (10.9) and (10.10). In fact, a straightforward algebraic manipulation can show that $P_N(n) = D_N(n-1) \equiv f(n)$. Thus, we can put $Q_N(n) = f_N(n) + f_N(n-1)$, with

$$f_N(n) \equiv \begin{cases} N! \sum_{x=\text{Max}[0, n-N]}^{\frac{n-1}{2}} \frac{1}{x!(n-2x-1)!(N-n+x)!} & \text{for } 0 < n < 2N \\ 0 & \text{otherwise} \end{cases} \quad (10.11)$$

Typical values are:

- 1) $n=1$ Then $f_N(1)=N$, $f_N(0)=0$ and $Q_N(1)=N$
- 2) $n=2N$ Then $f_N(2N)=0$, $f(2N-1)=N$ and $Q_N(2N)=N$.

It can be rather easily shown that $Q_N(n)$ is symmetric, namely that:

$$Q_N(n) = Q_N(2N-n+1)$$

Also, $P_N(n) = D_N(2N-n+1)$

and $D_N(n) = P_N(2N-n+1)$

This confirms our earlier qualitative observation of the symmetry of our decision graph.

We present a short example to see how P_N , D_N and Q_N behave.

For $N=4$, we obtain the following values:

n	$P_N(n)$	$D_N(n)$	$Q_N(n)$
1	4	0	4
2	12	4	16
3	24	12	36
4	28	24	52
5	24	28	52
6	12	24	36
7	4	12	16
8	0	4	4

A verification of these values is that, indeed $\sum_{n=1}^{2N} Q_N(n) = 2N3^{N-1}$ (=216 here), as expected.

We also observe the following:

- 1) the peak of $Q_N(n)$ occurs midway through the route, i.e. $Q_N(n)_{\max} = Q_N(N) = Q_N(N+1)$
- 2) $P_N(n) > D_N(n)$ for $n \leq N$ and
 $P_N(n) < D_N(n)$ for $n > N$.

This means that the states belonging to earlier stages of our route correspond mostly to pick-up stops while the opposite happens for states belonging to later stages of the route.

The following table displays values of $Q_N(n)_{\max} = Q_N(N)$ for several values of N :

<u>N</u>	<u>Q_N(N)</u>
1	1
2	4
3	15
4	52
5	175
6	576
7	1,869
8	6,000
9	19,107
10	60,460
11	190,333
12	596,652

Since $\sum_{n=1}^{2N} Q_N(n) = 2N3^{N-1}$ is an exponential function of N, the same should also hold for $Q_N(n)$. (Note that the issue of $Q_N(n)$ being an exponential function of n does not have any meaning, because it is N and not n that represents the size of our problem.)

Before presenting a method on how to enumerate all the states of a given stage, we shall investigate what happens when we impose capacity and MPS constraints. In particular, we shall attempt to find an expression for the number of feasible states as a function of N, MPS and the vehicle capacity C. We shall not focus on a particular stage of our problem but rather we shall examine this question from a global point of view. In addition, we should point out that since we shall not take into account the recursive nature of feasibility, it follows that the expression we shall

produce for the number of feasible states should be interpreted as an upper bound rather than as an exact figure.

We examine two cases:

1) Pick-up stop ($1 \leq L \leq N$). The corresponding MPS constraint was seen to be of the following form:

$$|L - (x_1 + x_2)| \leq \text{MPS} \quad (10.12)$$

where x_j is the number of k 's in (L, k_1, \dots, k_N) equal to j ($j=1,2$).

$x_1 + x_2$ is the number of pick-up stops up to and including that of state (L, k_1, \dots, k_N) .

Since $k_L = 2$ we already know that

$$x_2 \geq 1 \quad (10.13)$$

The vehicle capacity constraint was seen to be of the form:

$$x_2 \leq C \quad (10.14)$$

We also know that:

$$x_1 + x_2 \leq N \quad (10.15)$$

and that

$$x_1 \geq 0 \quad (10.16)$$

For a particular value of x_1 , the number of permutations in the k -vector is $\binom{N-1}{x_1}$, since k_L is already "frozen" to 2. From the remaining $N - x_1 - 1$ variables, $x_2 - 1$ of them take the value of 2, therefore the number of permutations is $\binom{N-x_1-1}{x_2-1}$.

Taking into account the constraints (10.12) through (10.16) and summing also over L we end up with the expression:

$$\sum_{L=1}^N \sum_{x_1=0}^{\min[N-1, L+\text{MPS}-1]} \sum_{x_2=\max[1, L-\text{MPS}-x_1]}^{\min[C, N-x_1, L+\text{MPS}-x_1]} \frac{(N-1)!}{(x_2-1)!(N-x_1-x_2)!x_1!} \quad (10.17)$$

This expression represents an upper bound for the number of feasible states corresponding pick-up stops.

2) Delivery stop ($N+1 \leq L \leq 2N$). The corresponding MPS constraint is of the form:

$$|L-N-x_1| \leq \text{MPS} \quad (10.18)$$

where x_1 is the number of k's equal to 1. Since $k_{L-N}=1$ we know that

$$x_1 \geq 1 \quad (10.19)$$

The vehicle capacity constraint was seen to be of the form:

$$x_2 \leq C-1 \quad (10.20)$$

with x_2 defined as before.

Also, we know that:

$$x_1 \leq N \quad (10.21)$$

For a particular value of x_1 , the number of permutations in the k-vector is $\binom{N-1}{x_1-1}$, since $k_{L-N}=1$ is already "frozen" to 1. From the remaining $N-x_1$ k's, x_2 of them are equal to 2. The corresponding number of permutations

is $\binom{N-x_1}{x_2}$.

Incorporating the constraints (10.18) through (10.21) and summing we obtain an expression similar to (10.17), the following:

$$\sum_{L=N+1}^{2N} \sum_{x_1=\text{Max}[1, L-N-MPS]}^{\text{Min}[N, L-N+MPS]} \sum_{x_2=0}^{\text{Min}[N-x_1, C-1]} \frac{(N-1)!}{(x_1-1)!(N-x_1-x_2)!x_2!} \quad (10.22)$$

This expression represents an upper bound for the number of feasible states corresponding to delivery stops.

Since both (10.17) and (10.22) seem sufficiently complicated, we examine several special cases of them:

a) $MPS=\infty$ (No priority constraints)

Then (10.17) reduces to:

$$N \cdot \sum_{x_1=0}^{N-1} \sum_{x_2=1}^{\text{Min}[C, N-x_1]} \frac{(N-1)!}{(x_2-1)!(N-x_1-x_2)!x_1!}$$

It is not difficult to see that we can interchange the summation signs and the above expression becomes equal to:

$$N \cdot \sum_{x_2=1}^{\text{Min}[N, C]} \sum_{x_1=0}^{N-x_2} \frac{(N-1)!}{(x_2-1)!(N-x_1-x_2)!x_1!} \quad (10.23)$$

But we know that:

$$2^{N-x_2} = \sum_{x_1=0}^{N-x_2} \frac{(N-x_2)!}{x_1!(N-x_2-x_1)!}$$

$$\text{Thus, rewriting } \frac{(N-1)!}{(x_2-1)!(N-x_1-x_2)!x_1!} = \frac{(N-x_2)!}{x_1!(N-x_2-x_1)!} \cdot \frac{(N-1)!}{(N-x_2)!(x_2-1)!}$$

and substituting into (10.23) we end up with the following expression:

$$N \cdot \sum_{x_2=1}^{\text{Min}[N,C]} 2^{N-x_2} \binom{N-1}{x_2-1} \quad (10.24)$$

Similarly, (10.22) reduces to:

$$N \cdot \sum_{x_1=1}^N \sum_{x_2=0}^{\text{Min}[C-1, N-x_1]} \frac{(N-1)!}{(x_1-1)!(N-x_1-x_2)!x_2!}$$

It is easy to see that the above expression is exactly equal to (10.23), or, equivalently, to (10.24). (This can be seen by a transformation of variables.)

Conclusion: If $\text{MPS} = \infty$ (no priority constraints), an upper bound for the number of feasible states is the expression:

$$F(N,C) \equiv 2N \cdot \sum_{x_2=1}^{\text{Min}[N,C]} 2^{N-x_2} \binom{N-1}{x_2-1} \quad (10.25)$$

b) The same formula holds if $\text{MPS} \geq N-1$. In fact, in (10.17) we shall have:

- i) $N-1 \leq L+\text{MPS}-1$ because $\text{MPS} \geq N-1$ and $L \geq 1$
- ii) $N-x_1 \leq L + \text{MPS} - x_1$ for the same reason
- iii) $1 \geq L-\text{MPS}-x_1$ because $L-\text{MPS}-x_1 \leq N-\text{MPS} \leq 1$.

If the above hold, we proceed as if $\text{MPS} = \infty$. Similarly, in (10.22) we shall have:

- i) $N \leq L-N+\text{MPS}$ because $L \geq N+1$ and $\text{MPS} \geq N-1$
- ii) $1 \geq L-N-\text{MPS}$ for the same reason.

If the above hold, we proceed as if $\text{MPS} = \infty$.

c) $MPS=\infty$ and $C=\infty$ (No priority and capacity constraints).

Then it is easy to see that we go back to our $2N \cdot 3^{N-1}$ states. The reason is that (10.25) reduces to:

$$F(N, \infty) = 2N \sum_{x_2=1}^N 2^{N-x_2} \binom{N-1}{x_2-1} = 2N \cdot 3^{N-1}$$

10.5 Indexing Considerations in Stage-By-Stage Solutions

The issue which arises when one considers the results of the previous section can be generally phrased as follows:

We now know how many consistent states correspond to a particular stage, how many of them concern pick up stops and how many concern deliveries. We also have bounds on the number of feasible states as functions of MPS and C. But how should we arrange these states so that they can be efficiently used in our algorithm? What indexing scheme should we provide for our data structure? Should we "compress" this structure so that it includes only (or mostly) feasible states? And if yes, what should be the corresponding indexing scheme?

Note that none of these issues had arisen in our first dial-a-ride algorithm to such an extent as to create considerable difficulties. In that algorithm, all states, even the inconsistent ones, were equally easily accessible*, because they all were kept in main storage. In the stage-by-stage solution approach however, the situation is different. Not only is

*A minor indexing problem actually involved how to map the $N+1$ -dimensional array (L, k_1, \dots, k_N) into a 1-dimensional array and vice versa (i.e., given an index of this 1-dimensional array, how to come back to the original $N+1$ -dimensional array). The need for this transformation is associated with the limitation of most computer languages on the dimensions of an array (in FORTRAN IV, for example, this dimension cannot exceed 7).

the number of states per stage variable, but all stages are not any more kept in main storage. Rather, we saw that this approach would involve a "create, copy and erase" process where we would essentially work with two state arrays, the first one involving all the consistent states of stage n and the second all the consistent states of stage $n+1$. Questions like "which is the state which is, say, fifth in the list of states of stage n ?" or, vice versa "which is the position in the list of states of stage $n+1$, of a particular state $(x, k'_1, \dots, k'_N)?"$, might seem trivial at first glance, but it turns out that they are not.

Before proceeding to an analysis of the above question, let us state that we shall not get involved with the problem of "compressing" our decision graph (which included so far all the consistent states) to take into advantage of our earlier devised upper bounds on the number of feasible states as a function not only of N , but also of C and MPS . If we did so, then a more efficient use of storage space would occur, because many infeasible states would not be considered at all and the size of our decision graph would be smaller. This is however the only advantage of this "compressing" philosophy. Its main disadvantages are: First, the size of our decision graph will now become a function not only of N , but also of MPS and C . Thus, each time we change these latter two parameters we shall have to reconstruct a new decision graph. Second, the structure of this "compressed" graph is likely to be substantially more complicated than the one of the graph containing the consistent states. Thus, a very complicated indexing scheme would have to be devised. We shall see below that even the indexing scheme of the stage-by-stage solution approach where we include all consistent states is far from trivial.

We can understand the non-triviality of the indexing problem if we attempt to answer the following question: Let $N=3$ (a very simple case). Among the states which form the list of states for stage $n=4$, which one occupies the 13th position?

It is of course understood that the answer to this question depends on how we order the states in the list. In this respect, we may proceed systematically as follows:

First, we use (10.9) and (10.10) to find that :

$$P_N(n) = 6 \quad \text{and} \quad D_N(n) = 9,$$

so that $Q_N(n) = 15$. Thus, this stage contains 15 states, 6 corresponding to pick-ups and 9 to deliveries. Assuming that our list is divided into two blocks, the first for pick-ups and the second for deliveries and since we are examining the 13th position in the list, our first conclusion is that the state we are trying to find has the 7th position in the block of deliveries. (See also Table 10.1).

We saw that the summations in (10.9) and (10.10) involved a single variable x_1 , the number of k 's in (L, k_1, \dots, k_N) equal to 1. Also, the range of x_1 in both cases is known. For deliveries, $\text{Max}[1, n-N] \leq x_1 \leq \frac{n}{2}$. In our example, this means that $1 \leq x_1 \leq 2$. We assume that the next subdivision inside a block of states concerning pick-ups or deliveries, is done according to the values of x_1 . In other words, we assume that our delivery block of 9 states is further subdivided into two parts, one for $x_1=1$ and one for $x_1=2$. The length of each part is the corresponding number of permutations, which is equal (for deliveries) to

$$\frac{N!}{(x_1-1)!(n-2x_1)!(N-n+x_1)!} \quad \text{In our case the corresponding lengths are 3}$$

TABLE 10.1

LIST OF ALL CONSISTENT STATES OF STAGE $n=4$ for $N=3$

MAIN INFORMATION

Index	x_1	L	k_1	k_2	k_3
1	1	1	2	1	2
2			2	2	1
3		2	1	2	2
4			2	2	1
5		3	1	2	2
6			2	1	2
7	1	4	1	2	2
8		5	2	1	2
9		6	2	2	1
10	2	4	1	1	3
11			1	3	1
12		5	1	1	3
13			3	1	1
14		6	1	3	1
15			3	1	1

AUXILIARY INFORMATION

x_2	x_3	"FROZEN VARIABLE"
2	0	$k_1 = 2$
		$k_2 = 2$
		$k_3 = 2$
2	0	$k_1 = 1$
		$k_2 = 1$
		$k_3 = 1$
0	1	$k_1 = 1$
		$k_2 = 1$
		$k_3 = 1$

for $x_1=1$ and 6 for $x_1=2$. Since we are looking for the 7th state in the delivery block, our second conclusion is that $x_1=2$ and that the state we are looking for is the 4th in the subdivision (of length 6) of the delivery block that has $x_1=2$.

At this point we may say that we know more than x_1 , for (10.3) and (10.4) allow us to estimate x_2 and x_3 as well. In our case $x_2=0$ and $x_3=1$. Thus, we know that (L, k_1, k_2, k_3) will have 2 k's equal to 1, no k equal to 2 and one k equal to 3.

To find L, we assume that the subdivision of length 6 that has $x_1=2$ is further subdivided into $N=3$ parts of equal length, one for each of the N values L can take. Since we are looking for the 4th state in that block of length 6, it is easy to see that we are into the 2nd of the 3 parts of length 2 that constitute that block, thus $L=5$ (since it is a delivery) and our state is the 2nd of the two states of that part.

At this point, another factor comes in: Since we know L, we know also the value of one of the k's. In our case, we know that $k_{L-N}=k_2$ is "frozen" to 1.

Thus, we are left with only one question unresolved: which permutation of the remaining k's corresponds to the 2nd state in the block that has 2 states, both of which have one remaining k equal to 1 and another one equal to 3? The answer to this question can be given by ordering these states lexicographically. In our case, these are (5,1,1,3) and (5,3,1,1). Clearly the second is the state which we are looking for, namely (5,3,1,1).

The above example hopefully illustrates the non-triviality, yet feasibility of the indexing problem one has to face in solving the dial-

a-ride problem stage-by-stage. In addition to the above, it is clear that for each state (L, k_1, \dots, k_N) of stage n we examine, we shall also be faced with the opposite problem as well: What is the index of a particular "next" state (x, k'_1, \dots, k'_N) in its corresponding next stage $n+1$? This latter problem is solvable by a procedure which is essentially the inverse of the one we have just applied above. Both procedures exhibit computational efforts which are linear functions of N , so the combined computational effort per stage will be of the order of N^2 . This effort, although a polynomial function of N , is by no means a negligible quantity.

As a matter of fact, the computer program which we have created to implement this solution algorithm and which spends this amount of time just for coding and decoding, proved very slow and cumbersome.

It turns out that a very simple observation can help overcome the above difficulty. This is the fact that the indexing structure of all problems of the same size N is exactly the same. So we can perform our coding and decoding calculations only once for each N . Doing this, we can create a kind of "directory" for every size N that we are interested in. Our algorithm then can use the information of the "directory" as a guide in solving the problem. Thus, the algorithm will completely skip the step of finding from scratch the state that has a particular index and vice versa. Instead, this information will be externally furnished to the algorithm by the "directory" already created. A typical directory will consist of $2N$ ordered segments, each segment containing all states of a particular stage n . For each state (L, k_1, \dots, k_N) the indices of all its "next" states (x, k'_1, \dots, k'_N) , will be included. These states can be located in the segment of stage $n+1$. The directories will be kept in

auxiliary storage. The algorithm will have in main storage two segments at a time, one for stage n and one for stage $n+1$. In this way, the process of identifying a certain state and all its next states is greatly simplified.

We show how a typical "directory" is organized in Table 10.2 ($N=3$). The directory segments are arranged by descending order of stage number, n . Each segment has length $Q_N(n)$. We include also a $2N+1^{\text{th}}$ segment of unit length for the single state of stage 0 (vehicle at starting point). To illustrate how this directory works, suppose we want to find which is the 5^{th} state of stage 2 and which are its "next" states. We do the following:

- a) We locate the segment of stage $n=2$: We see that this segment has size 9.
- b) We read the information of the 5^{th} line of this segment: We see that the corresponding state is $(L, k_1, k_2, k_3) = (3, 2, 3, 2)$. We also obtain the set of next-state indices $(j_1, j_2, j_3) = (2, 11, 14)$.
- c) We locate the segment of stage $n+1=3$: We see that this segment has size 15.
- d) We look at lines 2, 11 and 14 of this segment. This we see that the next states are: $(2, 2, 2, 2)$, $(4, 1, 3, 2)$ and $(6, 2, 3, 1)$ respectively.

Computational experience with this directory scheme has been satisfactory, although not particularly sensational. With a 300K core limitation*, our first "all-states-in-core" algorithm can handle problems of up to $N=7$ customers. If we eliminate the inconsistent states and use the

*This is the limitation of the IBM-370/168 TSO interactive system at M.I.T.

state representation (L, m_1, \dots, m_{N-1}) , this capability can increase to $N=8$. (It should be remembered that, as a rule of thumb, the storage requirements and computational effort increase three times when we increase N by one). By contrast, the stage-by-stage/directory algorithm requires only 120K of core storage for $N=9$ customers and thus can, in this respect, be handled with the 300K core limitation. It should be noted however that tests of problems of this size displayed intolerably high execution times.

We shall discuss computer implementation issues together with other suggestions in Part IV of the thesis (Chapter 11).

10.6 Summary of the D.P. Approach to the Dial-A-Ride Problem.

The above considerations conclude our investigation of the dial-a-ride problem. An exact, dynamic programming approach was developed in order to solve the single-vehicle, many-to-many version of this problem. Both the "static" and the "dynamic" cases were solved. In both cases a linear objective function, consisting of a weighted combination of the time to service all customers not serviced so far and their corresponding total disutility, was assumed. Vehicle capacity constraints, as well as priority constraints to prevent indefinite deferment of customer service, were incorporated into the problem. These constraints were seen to fit into the dynamic programming approach that was used. Two algorithms were presented, based on that approach. Their main difference was that the second algorithm examined the problem in a stage-by-stage way in order to make more efficient use of storage space. The combinatorics of this latter approach were finally examined.

TABLE 10.2
Directory for N=3

Stage	Size	Index		State			Indices Of Next States		
n	$Q_N(n)$	j(n)	L	k_1	k_2	k_3	$j_1(n+1)$	$j_2(n+1)$	$j_3(n+1)$
6	3	1	4	1	1	1	-	-	-
		2	5	1	1	1	-	-	-
		3	6	1	1	1	-	-	-
5	9	1	1	2	1	1	1	-	-
		2	2	1	2	1	2	-	-
		3	3	1	1	2	3	-	-
		4	4	1	1	2	3	-	-
		5	4	1	2	1	2	-	-
		6	5	1	1	2	3	-	-
		7	5	2	1	1	1	-	-
		8	6	1	2	1	2	-	-
		9	6	2	1	1	1	-	-
4	15	1	1	2	1	2	4	9	-
		2	1	2	2	1	5	7	-
		3	2	1	2	2	6	8	-
		4	2	2	2	1	5	7	-
		5	3	1	2	2	6	8	-
		6	3	2	1	2	4	9	-
		7	4	1	2	2	6	8	-
		8	5	2	1	2	4	9	-
		9	6	2	2	1	5	7	-
		10	4	1	1	3	3	-	-
		11	4	1	3	1	2	-	-
		12	5	1	1	3	3	-	-
		13	5	3	1	1	1	-	-
		14	6	1	3	1	2	-	-
		15	6	3	1	1	1	-	-

Table 10.2 (cont'd)

Directory for N=3

Stage	Size	Index		State			Indices of next states		
n	$Q_N(n)$	j(n)	L	k_1	k_2	k_3	$j_1(n+1)$	$j_2(n+1)$	$j_3(n+1)$
3	15	1	1	2	2	2	7	8	9
		2	2	2	2	2	7	8	9
		3	3	2	2	2	7	8	9
		4	1	2	1	3	6	10	-
		5	1	2	3	1	4	11	-
		6	2	1	2	3	5	12	-
		7	2	3	2	1	2	13	-
		8	3	1	3	2	3	14	-
		9	3	3	1	2	1	15	-
		10	4	1	2	3	5	12	-
		11	4	1	3	2	3	14	-
		12	5	2	1	3	6	10	-
		13	5	3	1	2	1	15	-
		14	6	2	3	1	4	11	-
		15	6	3	2	1	2	13	-
2	9	1	1	2	2	3	3	10	2
		2	1	2	3	2	2	11	14
		3	2	2	2	3	3	10	12
		4	2	3	2	2	1	13	15
		5	3	2	3	2	2	11	14
		6	3	3	2	2	1	13	15
		7	4	1	3	3	6	8	-
		8	5	3	1	3	4	9	-
		9	6	3	3	1	5	7	-
1	3	1	1	2	3	3	3	5	7
		2	2	3	2	3	1	6	8
		3	3	3	3	2	2	4	9
0	1	1	7	3	3	3	1	2	3

PART IV

FINAL REMARKS

CHAPTER 11

CONCLUSIONS AND DIRECTIONS FOR FURTHER RESEARCH

11.0 Introduction

In the single Chapter of this last part of the thesis, 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.

11.1 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 thesis and what we think are the main problems associated with that.

11.2 ASP: "Dynamic" Case

The reader familiar with both Parts II and III of this dissertation may have noticed that we have avoided extending the "static" CPS algorithm for the ASP into its equivalent "dynamic" algorithm, in the same fashion that did for the two equivalent dial-a-ride algorithms. Does a fundamental conceptual difference exist therefore between the sequencing of aircraft landings and the dispatching of a vehicle? It turns out that it does. We shall see below that the "dynamic" sequencing of aircraft landings is the equivalent of the advance-request version of the "dynamic" dial-a-ride problem, and in particular the version where the new customer requests to be serviced not earlier than a presented time in the future.

Leaving the discussion of this version of the dial-a-ride problem for an appropriate later section of this Chapter, let us examine the characteristics of the "dynamic" version of the ASP:

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 11.1.

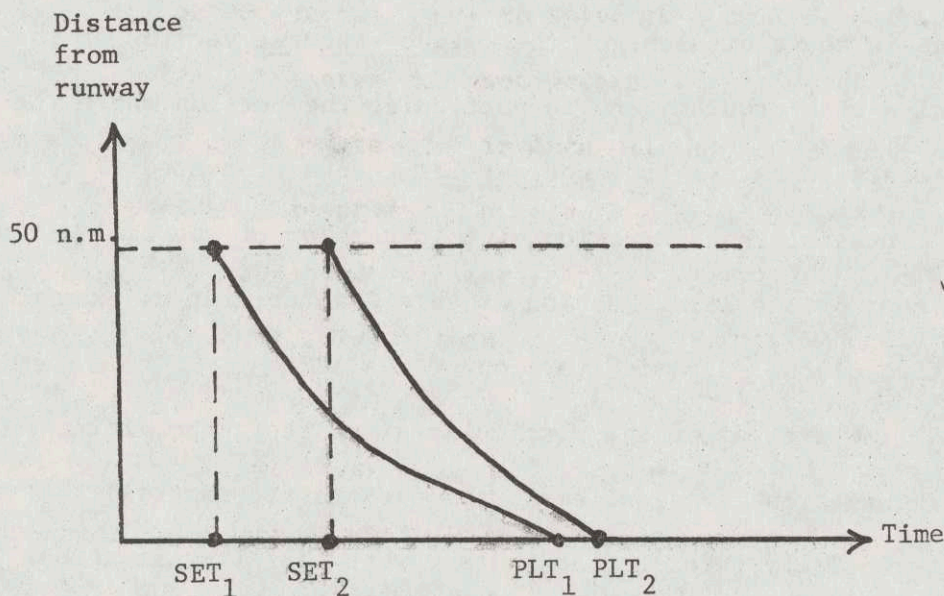


Fig. 11.1.

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 cannot 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 earlier than its corresponding PLT, is infeasible. 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) Defer 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 when 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 temporarily 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" ASD 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, as in the equivalent dial-a-ride algorithm, 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.

11.3 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. 5th, 10th, 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, 10th on the first runway, lands actually later than an airplane which lands, say 12th 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.

11.4 ASP - Usefulness of the "Steepest Descent" Heuristic for TPD Minimization.

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-optimal 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 insensitive 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 thesis.

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.

11.5 ASP - Implementation Issues

It is beyond the scope of this dissertation 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, if implemented, will result in the reduction of delays, but they must also be convinced that this strategy can in fact be implemented at all. 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-assisted, 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.

11.6 Dial-A-Ride: Review of the Results of this Work.

In this work, the "static" and "dynamic" versions of the single-vehicle,

immediate-request, many-to-many dial-a-ride problem were investigated and solution algorithms based on dynamic programming were developed.

In the "static" case, the task of the vehicle operator is to pick up a specified number of customers from distinct points and deliver them to similarly distinct points. New customer requests occurring during the execution of the route are not considered. The problem's objective is the minimization of a weighted combination of the following two measures of performance:

- 1) Time to service all customers
- 2) Total customer disutility

This last measure of performance is assumed to be a linear combination of the time each passenger waits for the vehicle and of the time he spends inside the vehicle until his delivery.

Vehicle capacity constraints and priority rules similar to CPS were also considered and were seen to be compatible with the dynamic programming formulation.

The "dynamic" case was formulated as an extension of the "static" case in the following sense: Each time a new (immediate) request appears, it is incorporated into the problem's input and an update of the optimal solution is made. No new customer requests are anticipated (in a probabilistic or any other way) so that new requests become part of the input only when they appear. The priority rules of the problem serve in this "dynamic" case to prevent the undesirable phenomenon of indefinite deferment of a customer's request.

In addition to the above, an investigation of the combinatorics of the problem was made and an alternative stage-by-stage algorithm was

developed. This last algorithm takes advantage of the information of the structure of the problem to reduce the storage requirement.

All algorithms presented are exponential in terms of computational effort and this fact limits the problem size to about 8 or 9 customers (depending on the facility used), i.e. to graphs of about 18 nodes. It is felt that for real-world data (concerning customer request rates, size of geographical region, vehicle speed and vehicle fleet size) the above graph size represents a reasonable upper bound for the scheduling operations of a single vehicle. Additional customers will either have to wait, or be assigned to other vehicles as we shall discuss below.

Let us now examine several directions towards which the work in this thesis on the dial-a-ride problem can be extended.

11.7. Dial-A-Ride: The Multi-Vehicle Problem

The most natural direction for extending the single-vehicle problem is obviously to its equivalent multi-vehicle version. As a matter of fact, since there can be no such thing as a single-vehicle operating agency, the practical importance of the multi-vehicle problem is overwhelming. The following considerations may suggest that the multi-vehicle dial-a-ride problem is likely to be substantially more difficult than its single-vehicle counterpart:

- 1) The problem of how to partition a given set of customers among a given set of vehicles seems to be the crux of the difficulty of the multi-vehicle dial-a-ride problem. It should be noted at this point that the classification of vehicle stops into origins and destinations tends to increase this difficulty even more. Thus, various algorithms developed

for the scheduling of vehicles from a central depot to a number of delivery points, do not fit the nature of the multi-vehicle dial-a-ride algorithm*. Thus, procedures that just partition the geographical region of the problem to subregions of points clustered together and that subsequently assign one vehicle to each region, are not adequate for our problem.

2) In the "dynamic" case, the appearance of a new customer will, in general, render some already made customer-to-vehicle assignments, sub-optimal. In other words, it is conceivable that it may no longer be optimal to keep customer i assigned to vehicle j after the appearance of a new customer k . This "re-shuffling" of customer-to-vehicle assignments, in order to avoid sub-optimal solutions, will certainly add to the computational effort of a "dynamic" multi-vehicle algorithm.

3) It may be necessary to redefine our priority constraints in order to avoid complications in the feasibility considerations of our problem. In fact, a customer picked up, say, 4th by one vehicle, may have been picked up later than another customer, picked-up 5th by another vehicle, but it will certainly be easier for our algorithm to examine priorities on a single-vehicle basis.

4) Concerning our problem's objectives, one can certainly see that the problem of servicing all customers as quickly as possible is a minimax problem. The problem of minimizing total customer disutility is certainly

*Among these algorithms, we can cite the exact branch-and-bound approach of Christofides and Eilon [CHRI 69] and various heuristic algorithms, such as the "savings" algorithm of Clarke and Wright [CLAR 64] and the "sweep" algorithm of Gillett and Miller [GILL 74]. For a comprehensive survey of vehicle routing algorithms, see [GOLD 76].

different. It is not clear at this moment whether the partitioning subproblem is easier in the minimax version or not.

Speculating on how one might go about "solving" the multi-vehicle problem, we can only suggest at this point that a way to proceed may be to try to develop a procedure which "solves" the partitioning problem using our exact, single vehicle algorithm as a subroutine. Concerning that procedure itself, it seems that the additional complexity of the problem may discourage further attempts to proceed through an exact approach. A possible suggestion with respect to this may be to develop a "hybrid" procedure, where the partitioning problem is solved heuristically while the corresponding single-vehicle problems exactly using our D.P. algorithm.

11.8 Dial-A-Ride: Incorporating Time Constraints

Another direction in which our single-vehicle algorithm could be extended is by incorporating time constraints. Specifically, each customer requesting service, is now allowed to request also specific time bounds for his pick-up or delivery time, or for both. For example, he may request to be picked up not earlier than 10 a.m., or to be delivered not later than 11 a.m., etc.

An investigation to see if we can fit these constraints into our dynamic programming algorithm in a way similar to the one we used for our priority constraints, showed that, by contrast to our existing "Screening" and "Optimization" procedures (in which feasibility is treated independently from optimality), time constraints involve an interaction of feasibility with optimality. This interaction means that the feasibility

of a particular state depends on what will be our optimal decision regarding the "next" state. Hence, the idea of directly incorporating time constraints into our dynamic programming algorithm was not further pursued.

An alternative idea concerning these constraints could be to try to "translate" them, in some fashion, to "equivalent" priority constraints, which can be handled by our algorithm. It should be noted at this point that this "translation" may not be necessarily exact.

Another (heuristic) idea would be to defer the inclusion of a customer who specifies an earliest pick-up time, into the problem's inputs, until some time before that prescribed instant. Actually, this is how these constraints are taken into account in [WILS 77b]. In the case when the algorithm produces a schedule in which the vehicle reaches the customer's pick-up point before the corresponding pick-up time, the vehicle remains idle at that point till the customer is ready.

11.9 Dial-A-Ride: Probabilistic Extension of the "Dynamic" Case

A most general probabilistic extension of the "Dynamic" case of the dial-a-ride problem could involve the taking into account information concerning the probabilistic spatial distribution of customer pick-up and delivery points into the region in which the vehicle operates, as well as information on the probabilistic law through which customer requests are generated in time. In that sense, the vehicle operator may conceivably prefer to drive through an area with high probability of customer requests, rather than to go directly to pick-up a specific customer who is situated at a low density area. The objective function for the probabilistic problem can be the minimization of the long-run (or average)

time to service all customers or of their average disutility.

A special case of this general probabilistic problem may be the problem in which we can anticipate with certainty and ahead of time the time of customer's request and his equivalent origin and destination points. How will one use this information to improve the performance of our algorithm? Recall (Chapter 9) that the inherent "myopia" of our current "dynamic" algorithm, (as far as the anticipation of future demands is concerned) will in general result in a posteriori suboptimal solutions.

11.10 Dial-A-Ride: Incorporating Heuristics into the D.P. Algorithm

This may sound like a strange idea, but if one is willing to sacrifice optimality in favor of solution speed, the idea may perhaps prove itself useful. Thus, instead of investigating all "next" states of a particular state, one might use a heuristic criterion to examine some of them. This criterion may involve, for example, the elimination of all next states which correspond to points "sufficiently distant" for the point of the current state. Other, more sophisticated criteria may be devised. In this way, the computational effort of the algorithm may be reduced. (It should be noted of course that the above suggestions concerning the form of the heuristic criteria were made mainly for motivation purposes rather than as recommendations of a specific heuristic.)

11.11 Dial-A-Ride: Implementation Issues

Several dial-a-ride systems are operational in one form or another in the United States and the implementation experience from these systems has been considerable. While it is not our intention to cover such issues

in detail here*, we can at least go over several important points:

1) While the structure of a dial-a-ride software package should be sufficiently general and versatile to be adapted to various urban configurations, it should be pointed out that several modules of that package should be specifically tailored to the particular environment for which the package is designed to operate. For example, a special data base should maintain all the available information on the geographical characteristics of the region: Streets, routes, distances etc. In addition, there should exist a module to generate the matrix $[t_{ij}]$ for all pairs of points of our problem. Recall that this matrix was easily calculated in our algorithms from the Cartesian coordinates of the specified points. In the real world situation however this simplification is no longer valid and going from a point of a map to another point will involve the use of the street grid of the region. The module should therefore be able to produce quickly the shortest path between any two specified points on the map, taking into account not only distances, but also traffic information such as one-way streets, allowable speeds etc.

2) The computer software should be flexible enough so that unpredictable events could be taken care of. These include emergency requests, vehicle breakdowns, request cancellations etc. Vehicle refuelling and overhaul should also be allowed.

3) Feedback from vehicle drivers should be allowed and invited in several circumstances. While it is understood that the whole concept of computer-assisted vehicle-dispatching is lost if drivers totally ignore

*Details concerning the implementation of the Rochester dial-a-ride project can be found in [WILS 77a].

the instructions of the computer and make their own decisions instead, on the other hand psychological and other reasons dictate that drivers should have an opportunity to examine what the computer has decided for their vehicles ahead of time.

4) A final issue concerns the nature of computer facilities available. Should there be a dedicated computer or a timeshared facility? In the Rochester project, the latter alternative was chosen and this involved a complex communications network between Rochester, Washington D.C., M.I.T. and Waltham, Mass., where the computer facility was situated at [WILS 77a]. If a timeshared facility is chosen, this would imply that the system response time will also depend on how many other users are on the system and this may result in run-time delays, which may be important for a real-time algorithm. On the other hand, the creation of a dedicated computer system from scratch will certainly imply other problems, as far as cost, debugging and maintenance is concerned, so that it may be advantageous to use the extensive facilities, software and usually increased back-up capability of a large timeshared system.

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 69] Christofides, N., S. Eilon, "An Algorithm for the Vehicle Dispatching Problem," Operational Research Quarterly, 20 (1969).
- [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).
- [CLAR 64] Clarke, C., I. Wright, "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points," Operations Research Vol. 12, No. 4 (1964).
- [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).

REFERENCES (CONT'D)

- [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).
- [GILL 74] Gillett, B., L. Miller, "A Heuristic Algorithm for the Vehicle Dispatch Problem," Operations Research, Vol. 22 (1974).
- [GOLD 76] Golden, B.L., "Large Scale Vehicle Routing and Related Combinatorial Problems," Ph.D. Thesis, Operations Research Center, M.I.T., Cambridge, Mass. (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).
- [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).
- [WILS 76] Wilson, N.H.M., Weissberg, Hauser, "Advanced Dial-A-Ride Algorithms Research Project: Final Report," Department of Civil Engineering, M.I.T. Report R76-20(1976).

- [WILS 77a] Wilson, N.H.M., N.J. Colvin, "Computer Control of the Rochester Dial-A-Ride System," M.I.T. Report, (1977).
- [WILS 77b] Wilson, N.H.M., E. Miller, "Advanced Dial-A-Ride Algorithms Research Project, Phase II: Interim Report," M.I.T. Report, (1977).
- [ZADE 73] Zadeh, N., "A Bad Network Problem for the Simplex Method and Other Minimum Cost Flow Algorithms," Mathematical Programming, 5, (1973).

PART V
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 :

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

Proof: Let $\Delta \equiv \text{Cost (A)} - \text{Cost (B)}$ 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.

$$\begin{aligned} \text{Then } \Delta &= Q \cdot t_{ii} + (Q - P_i) t_{ij} + (Q - P_i - P_j) t_{ji} \\ &\quad - Q \cdot t_{ij} - (Q - P_j) t_{ji} - (Q - P_i - P_j) t_{ii} \end{aligned}$$

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

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

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

RESULT 2 (Fig. A.1):

If $[t_{ij}]$ satisfies

$$t_{ii} + t_{jj} \leq t_{ij} + t_{ji} \quad (A.2)$$

$$\text{and if } \frac{P_i}{t_{ii}} > \frac{P_j}{t_{jj}}, \quad (A.3)$$

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:

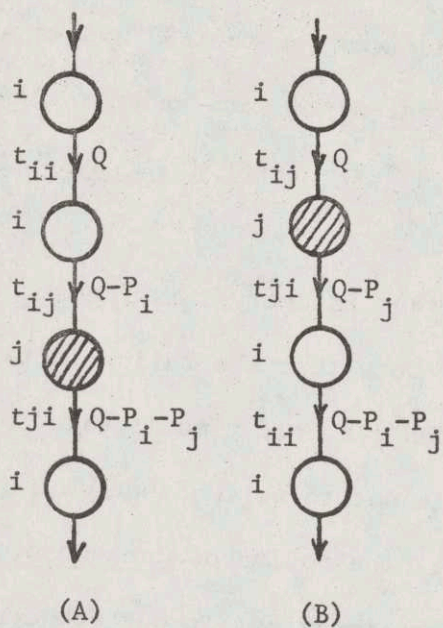


Fig. A.1.

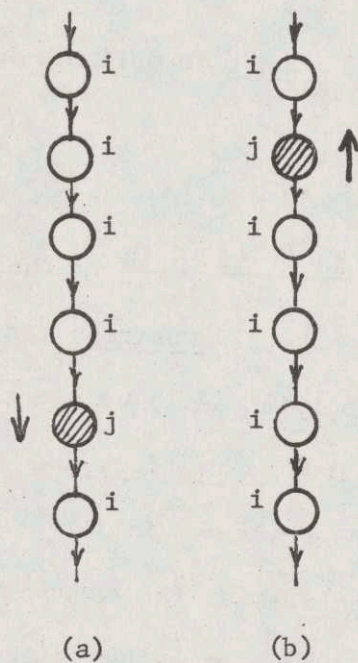


Fig. A.3.

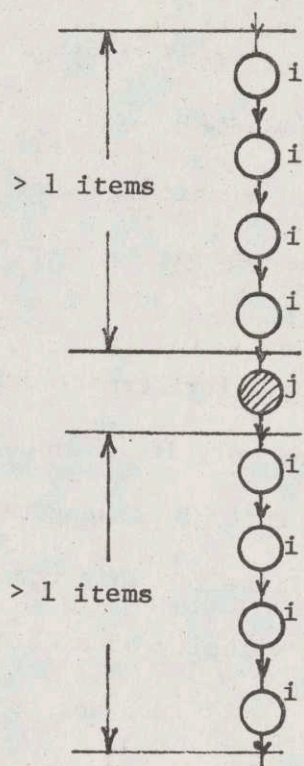


Fig. A.2.

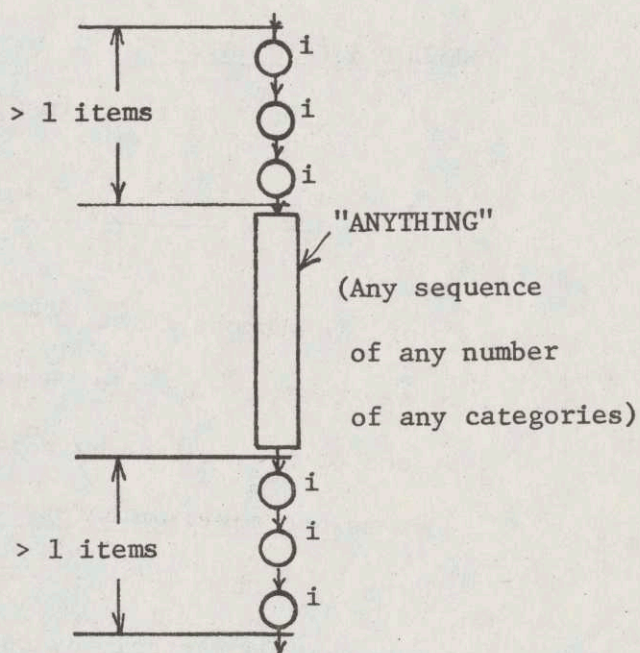


Fig. A.4.

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:

$$\frac{t_{ii}}{p_i} = \frac{t_{ij} + t_{ji} - t_{ii}}{p_j} \quad (\text{A.4})$$

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 separated (A) or overlap (B):

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

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 $(P_i + 2P_j)(t_{ii} + t_{jj} - t_{ij} - t_{ji}) + P_j(t_{ji} + t_{ij} - t_{jj}) - P_i t_{jj} < 0$ 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.

$$\frac{t_{ii} + t_{jj}}{P_i + P_j} < \frac{t_{ij} + t_{ji} - t_{ii}}{P_j} \quad (\text{A.6})$$

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

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

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

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

which is true since i is "denser" than j .

Second we show that:

$$\frac{t_{jj}}{P_j} < \frac{t_{ij} + t_{ji} - t_{ii}}{P_j} \quad (\text{A.8})$$

which is obvious if $[t_{ij}]$ is (i,j)-reasonable.

Then (A.6) follows directly from (A.7) and (A.8). But (A.6) is a sufficient 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 correctly at its "downmost" position, Result 3 cannot resolve the question of whether it should go further downstream, 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.

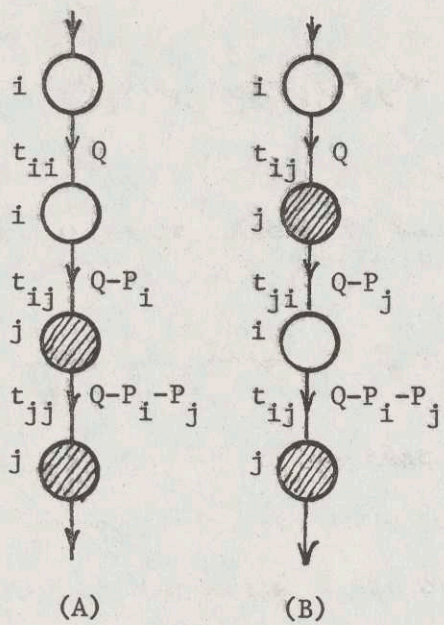


Fig. A.5.

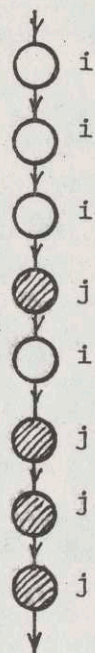


Fig. A.6.

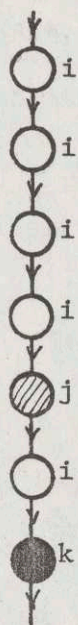


Fig. A.7.

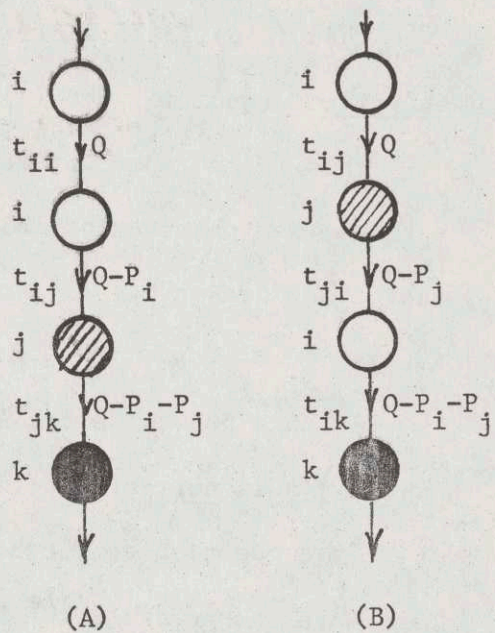


Fig. A.8.

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:

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

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:

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

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 $\Delta = \text{Cost}(A) - \text{Cost}(B)$. 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 $\Delta > 0$ #.

RESULT 12 (Fig. A.10)

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

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

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

RESULT 13 (End of the queue as above but with $k=i$)

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

Proof: From (A.11), setting $k=i$ #.

RESULT 14

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

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

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

We next present three results concerning the comparison of groups of categories.

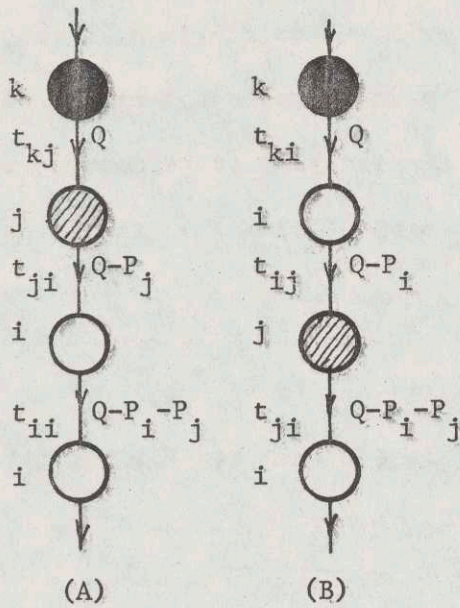


Fig. A.9.

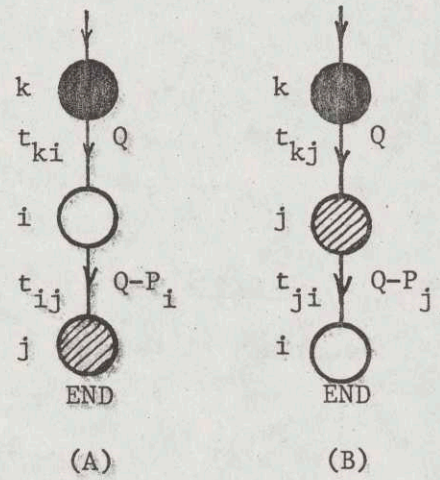


Fig. A.10.

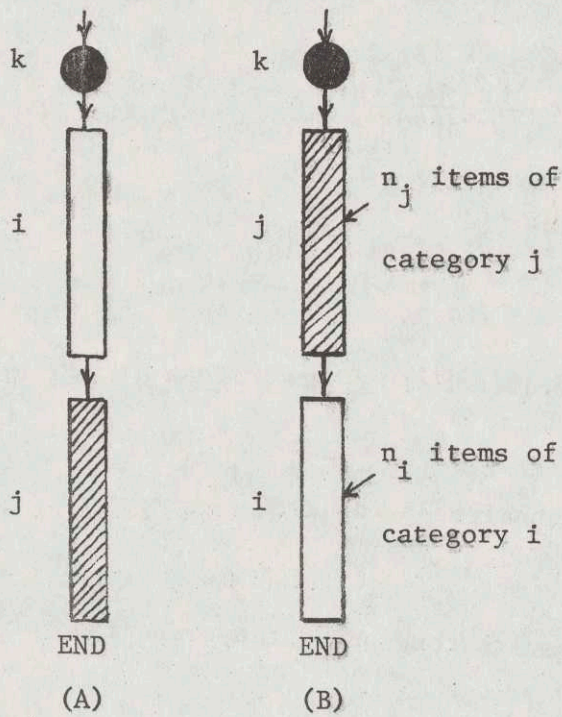


Fig. A.11.

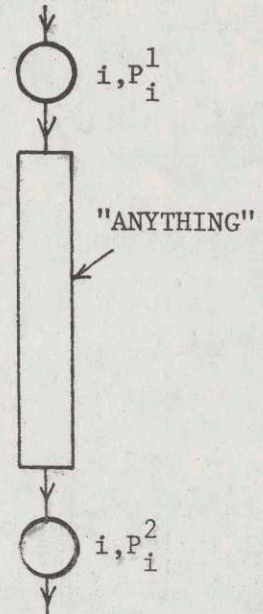


Fig. A.12.

RESULT 15 (Fig. A.11)

$$(A) > (B) \Leftrightarrow n_i n_j (P_j t_{ii} - P_i t_{jj}) + (n_i P_i + n_j P_j) (t_{ki} - t_{kj}) + \\ + n_i P_i (t_{jj} - t_{ji}) - n_j P_j (t_{ii} - t_{ij}) < 0 \quad (A.13)$$

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

RESULT 17:

If $[t_{ij}]$ is "(i,j,k)-reasonable", and if the relation $[(A) > (B) \text{ for } (n_i, n_j)]$ is true, the relation $[(A) > (B) \text{ for } (n_i+1, n_j)]$ 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 B, 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 horizontal length of the common final approach, 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.

$$(v_i < v_j \Leftrightarrow i > j.)$$

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_0 , 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 ($v_i > v_j \Leftrightarrow i < j$).

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_0 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_0 to the landing of airplane j) and $\frac{F}{v_i}$ (Time interval from T_0 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)

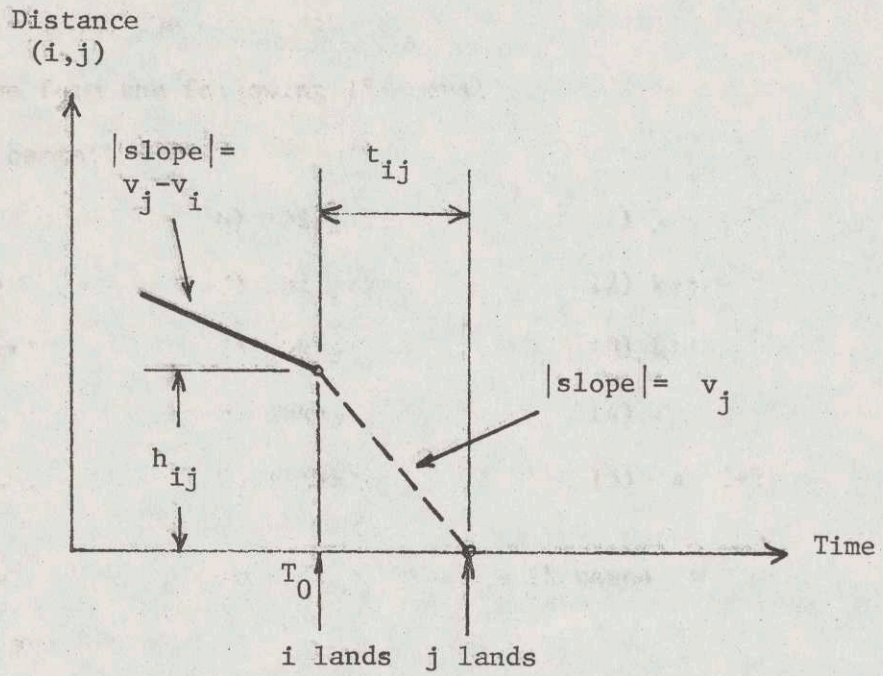


Fig. B.1 : "Overtaking" case ($v_j > v_i$).

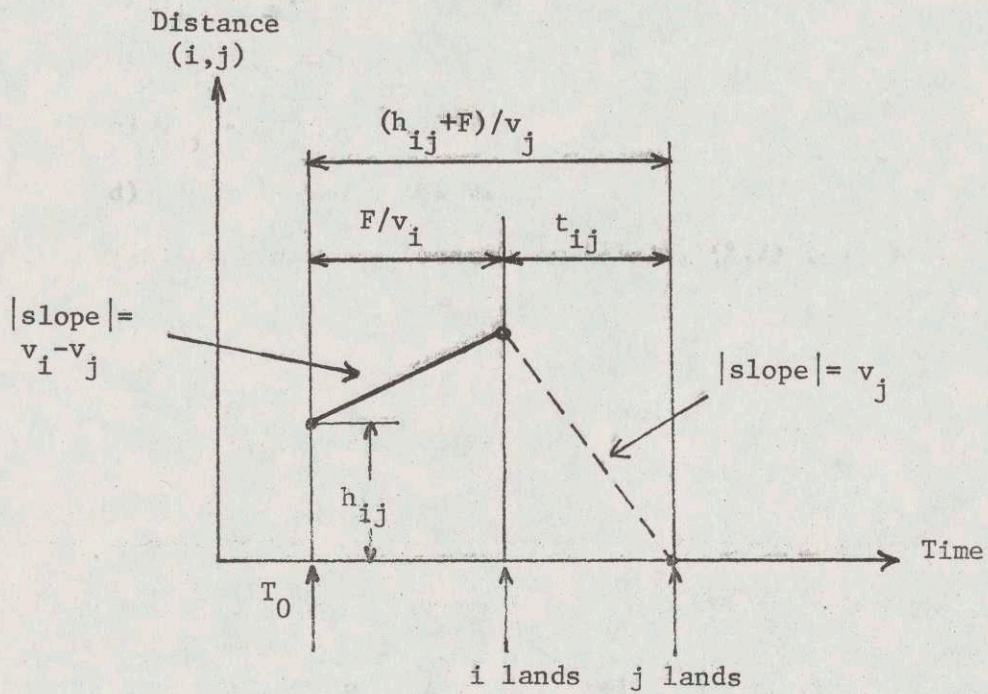


Fig. B.2 : "Opening" case ($v_i > v_j$).

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 (i,j,k)-reasonableness:

A time separation matrix $[t_{ij}]$ is (i,j,k)-reasonable 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 (i,j)-reasonable 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, except 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) $i=j$ | 6) $k>i>j>1$ | 11) $j>i>k=1$ |
| 2) $k=i \neq j$ | 7) $k>i>j=1$ | 12) $k>j>i>1$ |
| 3) $i>j>k=1$ | 8) $i>k>j>1$ | 13) $k>j>i=1$ |
| 4) $i>j \gg k=1$ | 9) $i>k>j=1$ | 14) $k \gg j>i>1$ |
| 5) $i>j=k=1$ | 10) $j>i>k>1$ | 15) $j \gg k>i=1$ |

We shall present here only one 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.

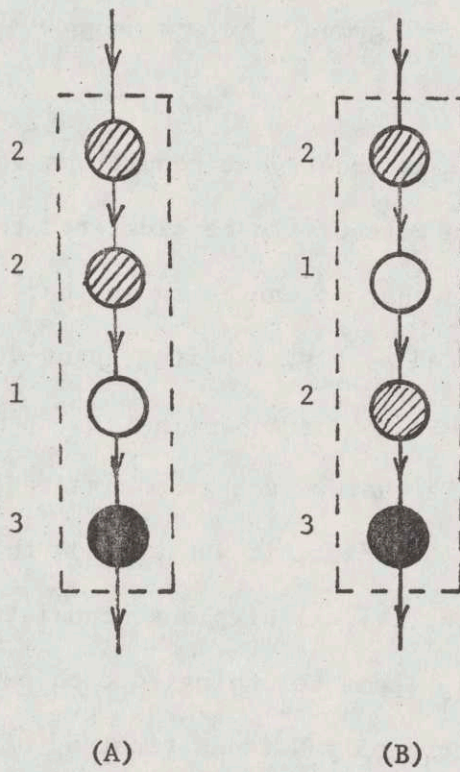


Fig. C.1.

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, \dots, 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

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

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

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

Rewriting we have:

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

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

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

From (D.2) we find also that:

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

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

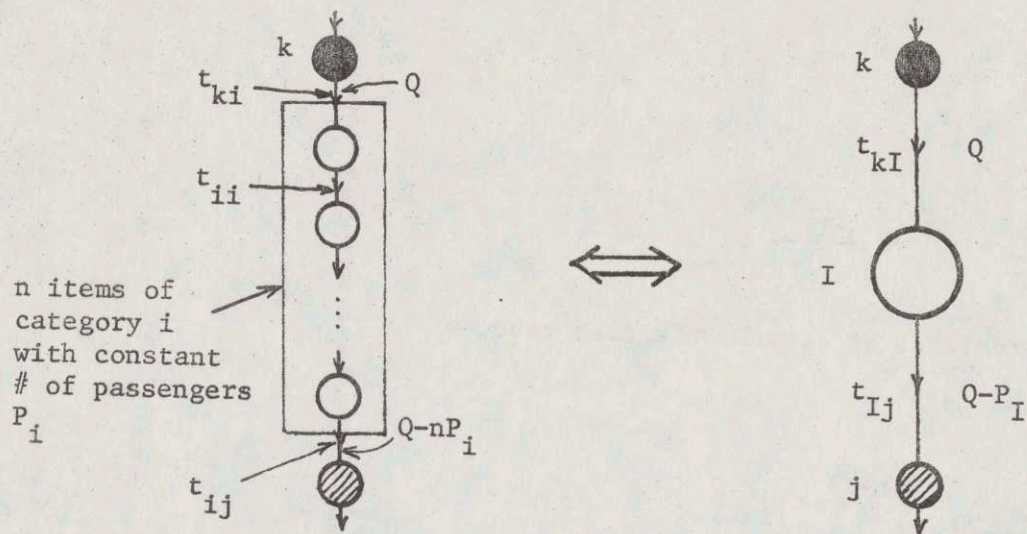


Fig. D.1.

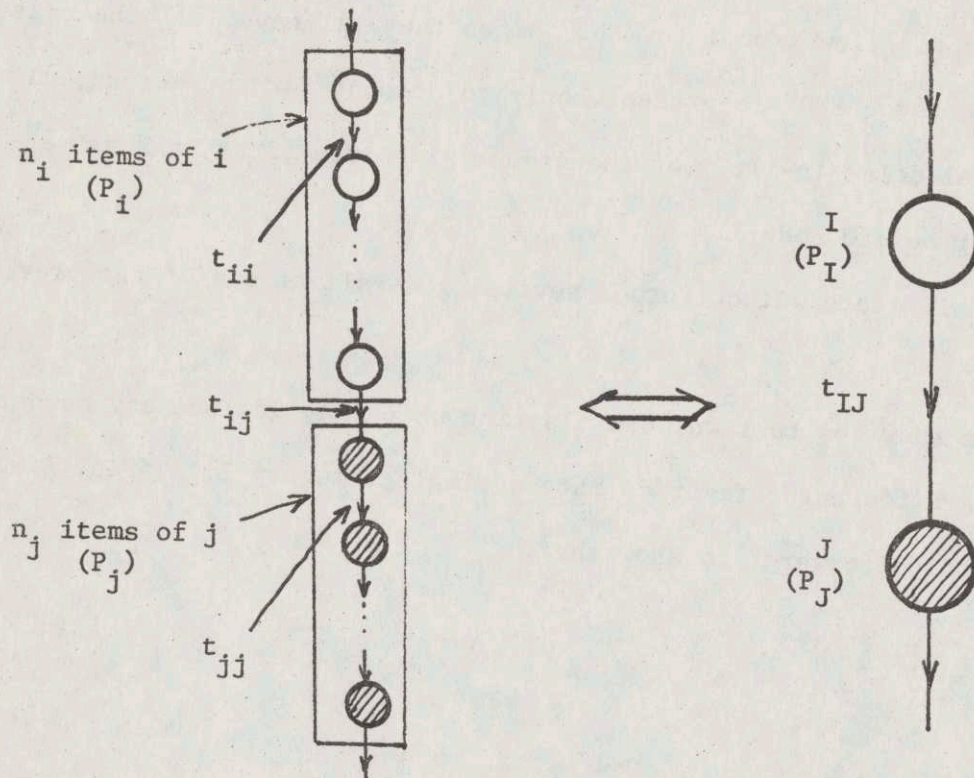


Fig. D.2.

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:

$$P_I = n P_i$$

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

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:

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

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

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

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

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

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

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

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

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

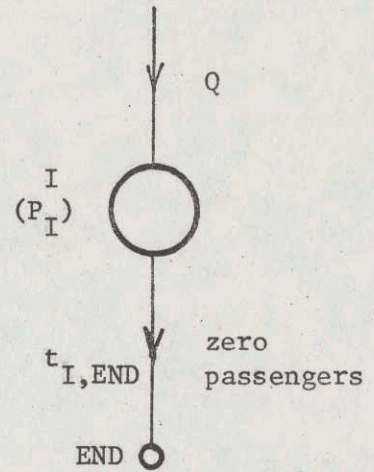
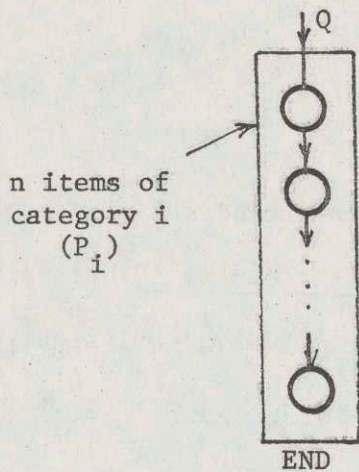


Fig. D.3.

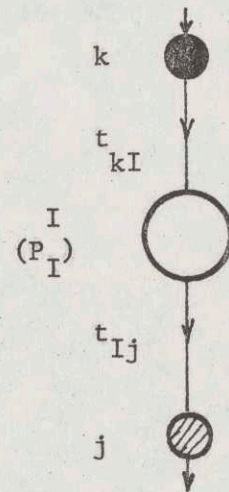
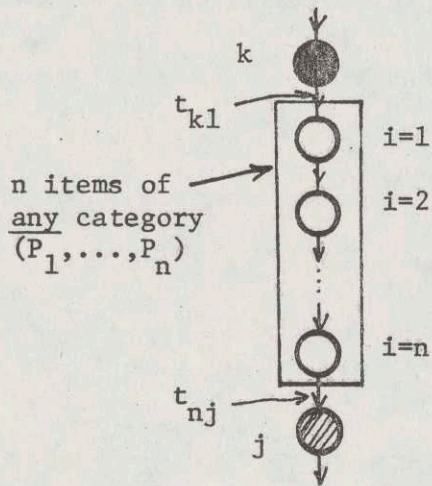


Fig. D.4.

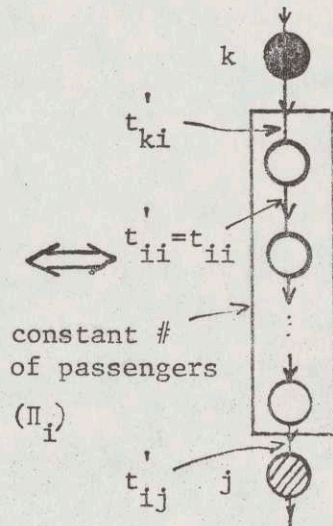
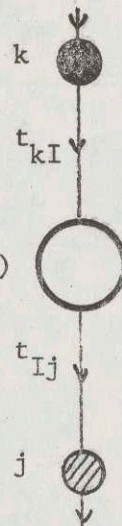
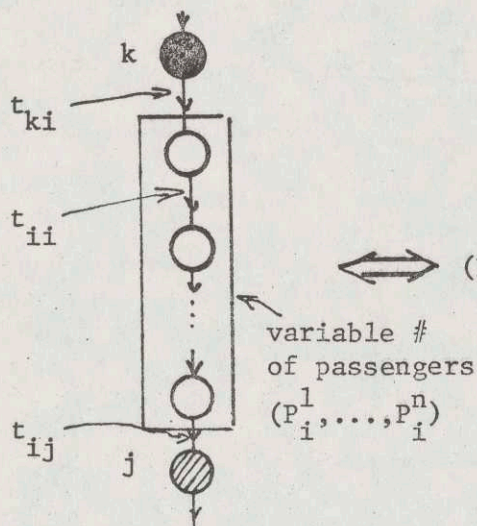


Fig. D.5.

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

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 $C = Q \cdot t_{ki} + (Q - P_i) t_{ij}$ 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 descending order of number of passengers, so that finally $\delta_i^1 \leq \delta_i^2 \leq \dots \leq \delta_i^n$.

This confirms the observation we have made in Appendix A when examining 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 two strings 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:

$$\delta_i^m = P_i^m - \Pi_i \quad (m=1, \dots, n_i)$$

$$\delta_j^m = P_j^m - \Pi_j \quad (m=1, \dots, n_j)$$

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

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

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

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

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

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

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

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:

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

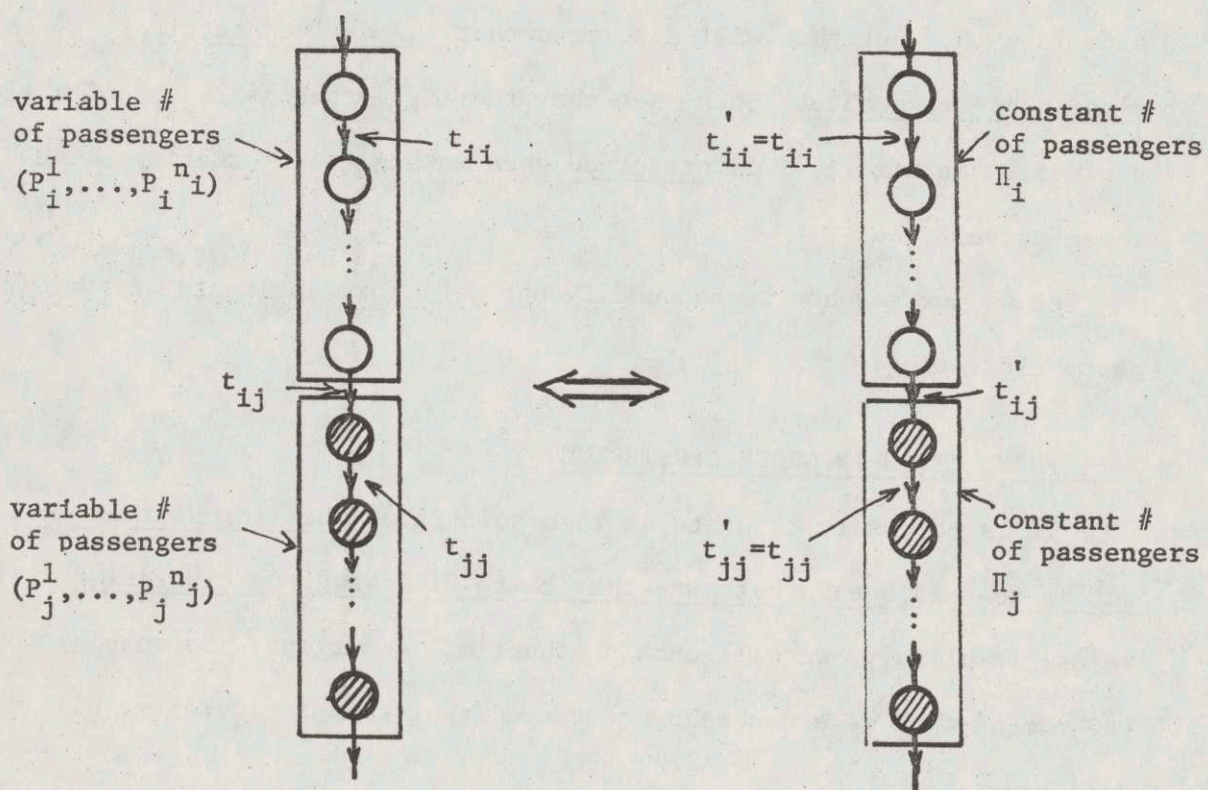


Fig. D.6.

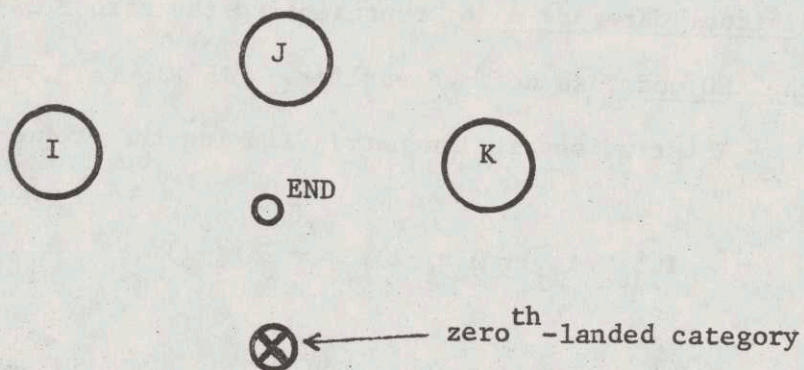


Fig. D.7.

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:

$$P_I = n_i P_i \quad (i=1, \dots, N)$$

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

$$V_Z(L, k_1, \dots, k_N) = \min_{x \in X} [W_Z \cdot \tau + V_Z(x, k'_1, \dots, k'_N)]$$

where L is the currently landed item (which may be either a whole group, if we are at stage $n \leq N$, or a single plane, 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:

$$k'_j = \begin{cases} k_j - 1 & \text{if } j=x \text{ and } n \leq N \\ k_j & \text{otherwise} \end{cases}$$

Also:

$$\tau = \begin{cases} T_{Lx} & \text{if } n \leq N \\ T'_{Lx} & \text{if } n=N+1 \end{cases}$$

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 $V_Z(L,0,0\dots)=0$ for any $L=1,\dots,N$ and for $Z=1,2$.

This condition applies only for $Z=2$ here.

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:

$$V_1(L,0,\dots,0) = \frac{(n_L-1)}{2} t_{II}$$

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

THE COMPUTER PROGRAMS

All computer programs of this thesis 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 (except the one developed to create the directories described in Chapter 10) are interactive, prompting the user to enter the input, select the options, modify the data, etc. Specifically, the following programs were developed:

1) ASP-singlerunway-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 $(i_0, k_1^0, \dots, k_N^0)$ 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(413) TO TERMINATE TYPE -1 OR -2 OR -3

3 1 5 5

INIT. COND:

START FROM CAT.

STAGE	FROM	TO	1	5	5	OBJECTIVE- 2	TIME	SUBT1-	80	COST-	124000	SUBT2-	124000	REST-	634550	OPT.COST-	758550
STAGE 11	FROM 3	TO 2	---	1	4	5	PASS- 1550	TIME- 80	SUBT1-	160	COST-	112000	SUBT2-	236000	REST-	522550	
STAGE 10	FROM 2	TO 2	---	1	3	5	PASS- 1400	TIME- 80	SUBT1-	240	COST-	100000	SUBT2-	336000	REST-	422550	
STAGE 9	FROM 2	TO 2	---	1	2	5	PASS- 1250	TIME- 80	SUBT1-	320	COST-	88000	SUBT2-	424000	REST-	334550	
STAGE 8	FROM 2	TO 2	---	0	1	5	PASS- 1100	TIME- 80	SUBT1-	392	COST-	68400	SUBT2-	492400	REST-	266150	
STAGE 7	FROM 2	TO 1	---	0	0	5	PASS- 950	TIME- 72	SUBT1-	573	COST-	117650	SUBT2-	610050	REST-	148500	
STAGE 6	FROM 1	TO 2	---	0	0	4	PASS- 650	TIME-181	SUBT1-	690	COST-	58500	SUBT2-	668550	REST-	90000	
STAGE 5	FROM 2	TO 3	---	0	0	3	PASS- 500	TIME-117	SUBT1-	780	COST-	36000	SUBT2-	704550	REST-	54000	
STAGE 4	FROM 3	TO 3	---	0	0	2	PASS- 400	TIME- 90	SUBT1-	870	COST-	27000	SUBT2-	731550	REST-	27000	
STAGE 3	FROM 3	TO 3	---	0	0	1	PASS- 300	TIME- 90	SUBT1-	960	COST-	18000	SUBT2-	749550	REST-	9000	
STAGE 2	FROM 3	TO 3	---	0	0	0	PASS- 200	TIME- 90	SUBT1-	1050	COST-	9000	SUBT2-	758550	REST-	0	
STAGE 1	FROM 3	TO 3	---	0	0	0	PASS- 100	TIME- 90	SUBT1-		COST-		SUBT2-		REST-		

ENTER 0TH LAND. CAT. & 3 NOS. OF PLANES FROM 0 TO 5,

FORMAT(413) TO TERMINATE TYPE -1 OR -2 OR -3

3 2 5 5

INIT. COND:

START FROM CAT.

STAGE	FROM	TO	1	5	5	OBJECTIVE- 2	TIME	SUBT1-	72	COST-	133200	SUBT2-	133200	REST-	803550	OPT.COST-	936750
STAGE 12	FROM 3	TO 1	---	1	5	5	PASS- 1850	TIME- 72	SUBT1-	168	COST-	148800	SUBT2-	282000	REST-	654750	
STAGE 11	FROM 1	TO 1	---	0	5	5	PASS- 1550	TIME- 96	SUBT1-	349	COST-	226250	SUBT2-	508250	REST-	428500	
STAGE 10	FROM 1	TO 2	---	0	4	5	PASS- 1250	TIME-181	SUBT1-	429	COST-	88000	SUBT2-	596250	REST-	340500	
STAGE 9	FROM 2	TO 2	---	0	3	5	PASS- 1100	TIME- 80	SUBT1-	509	COST-	76000	SUBT2-	672250	REST-	264500	
STAGE 8	FROM 2	TO 2	---	0	2	5	PASS- 950	TIME- 80	SUBT1-	589	COST-	64000	SUBT2-	736250	REST-	200500	
STAGE 7	FROM 2	TO 2	---	0	1	5	PASS- 800	TIME- 80	SUBT1-	669	COST-	52000	SUBT2-	788250	REST-	148500	
STAGE 6	FROM 2	TO 2	---	0	0	4	PASS- 650	TIME- 80	SUBT1-	786	COST-	58500	SUBT2-	846750	REST-	90000	
STAGE 5	FROM 2	TO 3	---	0	0	3	PASS- 500	TIME-117	SUBT1-	876	COST-	36000	SUBT2-	882750	REST-	54000	
STAGE 4	FROM 3	TO 3	---	0	0	2	PASS- 400	TIME- 90	SUBT1-	966	COST-	27000	SUBT2-	909750	REST-	27000	
STAGE 3	FROM 3	TO 3	---	0	0	1	PASS- 300	TIME- 90	SUBT1-	1056	COST-	18000	SUBT2-	927750	REST-	9000	
STAGE 2	FROM 3	TO 3	---	0	0	0	PASS- 200	TIME- 90	SUBT1-		COST-		SUBT2-		REST-		
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(413) 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.)

<u>"TIME"</u>	<u>"COST"</u>
LLT _o (FCFS)	TPD _o (FCFS)
LLT	TPD
LLT _o - LLT	TPD _o - TPD
$100(LLT_o - LLT) / LLT_o$	$100(TPD_o - TPD) / TPD_o$

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 8 4 MPS- 5

ENTER L0 OR -1,-2,-3,-4,-5,-6.FOR HELP HIT RETURN

2

INIT. COND:

STAGE	FROM	CAT.	TO	COND.	5	6	4	OBJECTIVE- 2	TIME- 80	SUBT1-	COST-	224000	SUBT2-	224000	REST-	1659250
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	5	4	PASS- 2650	TIME- 72	SUBT1- 152	COST- 198800	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
 INITIAL SEQUENCE : 2 1 1 3 2 2 3 2 1 2 1 3 3 2 1 2
 OPTIMAL SEQUENCE : 2 2 1 1 1 1 2 2 3 3 2 2 1 2 3 3
 POSITION SHIFTING : 0 3 -1 -1 4 5 -1 0 -5 -3 -1 2 2 2 -3 -3
 PERCENT GAINS : 1

TIME COST
 1729 2383800
 1520 1883250
 201 500550
 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
STAGE 6 FROM 1 TO 1 --- 1 2 2 PASS= 1100 TIME= 86 SUBT1= 96 COST= 105600 SUBT2= 105600 OPT.COST= 636
STAGE 5 FROM 1 TO 2 --- 1 1 2 PASS= 800 TIME= 181 SUBT1= 277 COST= 144800 SUBT2= 250400 REST= 548
STAGE 4 FROM 2 TO 2 --- 1 0 2 PASS= 650 TIME= 80 SUBT1= 357 COST= 52000 SUBT2= 302400 REST= 359
STAGE 3 FROM 2 TO 3 --- 1 0 1 PASS= 500 TIME= 117 SUBT1= 474 COST= 58500 SUBT2= 360900 REST= 279
STAGE 2 FROM 3 TO 3 --- 1 0 0 PASS= 400 TIME= 90 SUBT1= 564 COST= 36000 SUBT2= 396900 REST= 162
STAGE 1 FROM 3 TO 1 --- 0 0 0 PASS= 300 TIME= 72 SUBT1= 636 COST= 21600 SUBT2= 418500 REST= 72
0

RUNWAY 2
START FROM CAT. 1 --- 2 2 2 OBJECTIVE= 1
STAGE 6 FROM 1 TO 1 --- 1 2 2 PASS= 1100 TIME= 86 SUBT1= 96 COST= 105600 SUBT2= 105600 OPT.COST= 636
STAGE 5 FROM 1 TO 2 --- 1 1 2 PASS= 800 TIME= 181 SUBT1= 277 COST= 144800 SUBT2= 250400 REST= 548
STAGE 4 FROM 2 TO 2 --- 1 0 2 PASS= 650 TIME= 80 SUBT1= 357 COST= 52000 SUBT2= 302400 REST= 359
STAGE 3 FROM 2 TO 3 --- 1 0 1 PASS= 500 TIME= 117 SUBT1= 474 COST= 58500 SUBT2= 360900 REST= 279
STAGE 2 FROM 3 TO 3 --- 1 0 0 PASS= 400 TIME= 90 SUBT1= 564 COST= 36000 SUBT2= 396900 REST= 162
STAGE 1 FROM 3 TO 1 --- 0 0 0 PASS= 300 TIME= 72 SUBT1= 636 COST= 21600 SUBT2= 418500 REST= 72
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
STAGE 7 FROM 2 TO 3 --- 2 0 4 PASS= 1100 TIME= 117 SUBT1= 117 COST= 128700 SUBT2= 128700 OPT.COST= 645
STAGE 6 FROM 3 TO 3 --- 2 0 3 PASS= 1000 TIME= 90 SUBT1= 207 COST= 90000 SUBT2= 218700 REST= 528
STAGE 5 FROM 3 TO 3 --- 2 0 2 PASS= 900 TIME= 90 SUBT1= 297 COST= 81000 SUBT2= 299700 REST= 438
STAGE 4 FROM 3 TO 3 --- 2 0 1 PASS= 800 TIME= 90 SUBT1= 387 COST= 72000 SUBT2= 371700 REST= 348
STAGE 3 FROM 3 TO 3 --- 2 0 0 PASS= 700 TIME= 90 SUBT1= 477 COST= 63000 SUBT2= 434700 REST= 258
STAGE 2 FROM 3 TO 1 --- 1 0 0 PASS= 600 TIME= 72 SUBT1= 549 COST= 43200 SUBT2= 477900 REST= 168
STAGE 1 FROM 1 TO 1 --- 0 0 0 PASS= 300 TIME= 96 SUBT1= 645 COST= 28800 SUBT2= 506700 REST= 96
0

RUNWAY 2
START FROM CAT. 2 --- 3 5 0 OBJECTIVE= 1
STAGE 8 FROM 2 TO 2 --- 3 4 0 PASS= 1650 TIME= 80 SUBT1= 80 COST= 132000 SUBT2= 132000 OPT.COST= 664
STAGE 7 FROM 2 TO 2 --- 3 3 0 PASS= 1500 TIME= 80 SUBT1= 160 COST= 120000 SUBT2= 252000 REST= 584
STAGE 6 FROM 2 TO 2 --- 3 2 0 PASS= 1350 TIME= 80 SUBT1= 240 COST= 108000 SUBT2= 360000 REST= 504
STAGE 5 FROM 2 TO 2 --- 3 1 0 PASS= 1200 TIME= 80 SUBT1= 320 COST= 96000 SUBT2= 456000 REST= 424
STAGE 4 FROM 2 TO 2 --- 3 0 0 PASS= 1050 TIME= 80 SUBT1= 400 COST= 84000 SUBT2= 540000 REST= 344
STAGE 3 FROM 2 TO 1 --- 2 0 0 PASS= 900 TIME= 72 SUBT1= 472 COST= 64800 SUBT2= 604800 REST= 264
STAGE 2 FROM 1 TO 1 --- 1 0 0 PASS= 600 TIME= 96 SUBT1= 568 COST= 57600 SUBT2= 662400 REST= 192
STAGE 1 FROM 1 TO 1 --- 0 0 0 PASS= 300 TIME= 96 SUBT1= 664 COST= 28800 SUBT2= 691200 REST= 96
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.

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 "GROUING" 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."

5) Dial-A-Ride-"Static" case; This program implements the algorithm

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 1 110 110 130
 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
 STAGE 3 FROM 3 TO 3 --- 1 1 0 PASS- 1750 TIME-270 SUBT1- 270 COST- 472500 SUBT2- 472500 OPT.COST- 1121500
 STAGE 2 FROM 3 TO 1 --- 0 1 0 PASS- 1100 TIME-300 SUBT1- 660 COST- 420000 SUBT2- 901500 REST- 649000
 STAGE 1 FROM 1 TO 2 --- 0 0 0 PASS- 550 TIME-400 SUBT1- 1060 COST- 220000 SUBT2- 1121500 REST- 220000
 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 1 110 110 120
 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
 STAGE 3 FROM 3 TO 1 --- 0 1 1 PASS- 1700 TIME-210 SUBT1- 210 COST- 357000 SUBT2- 357000 OPT.COST- 1087000
 STAGE 2 FROM 1 TO 2 --- 0 0 1 PASS- 1150 TIME-400 SUBT1- 610 COST- 460000 SUBT2- 817000 REST- 730000
 STAGE 1 FROM 2 TO 3 --- 0 0 0 PASS- 600 TIME-450 SUBT1- 1060 COST- 270000 SUBT2- 1087000 REST- 270000
 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.

developed in Chapter 8. Table E.5 shows a typical output of this program.

Inputs to the program are (see symbols in Table E.5):

"N": Number of Customers

"SPEED": Vehicle Speed in mph.

"MPS": Maximum Position Shift

"CAP": Vehicle Capacity

"OBJ": Objective function of the problem. Two options are allowed: OBJ=1 to minimize time to service all customers and OBJ=2 to minimize total customer disutility.

"ALFA": Parameter α of customers' disutility (see Chapter 8).

The rest of the input consists of the Cartesian coordinates of the initial position of the vehicle (here named "DEPOT") and of the origins and destinations of the customers (in miles).

The program provides as outputs two "SUMMARIES" (see Table E.5). One for the vehicle, summarizing its schedule in time and one for each individual customer, displaying such information as pick-up time, delivery time, pick-up order, pick-up shift, delivery order, delivery shift and disutility (called here "WEIGHTED WAITING TIME").

In addition to the above, the program plots the vehicle route, as already shown in Figures 8.6 to 8.12. The plotting capability is provided through the ADVANCED GRAPHING feature of the M.I.T. Information Processing Services, developed for use with TEKTRONIX Graphics display terminals.

6) Dial-A-Ride-"Dynamic" Case: This program implements the algorithm developed in Chapter 9. A typical session of this program is shown in Table E.6. The program has essentially the same form of output with that

Table E.5: Typical output of the computer program for the dial-a-ride-"static" case (Chapter 8). This particular example corresponds to Case 1 of Chapter 8.

N= 7 SPEED=30.00 MPS= 6 CAP= 7 OBJ= 2 ALFA= 1.00

COORDINATES IN MILES

DEPOT LOCATED AT		3.000		3.000	
CUSTOMER	PICK-UP	DELIVERY			
1	2.000	8.000	11.000	8.000	
2	6.000	7.000	5.000	0.0	
3	10.000	2.000	7.000	1.000	
4	5.000	4.000	3.000	6.000	
5	1.000	4.000	9.000	6.000	
6	10.000	4.000	5.000	3.000	
7	2.000	1.000	7.000	5.000	

TIME MATRIX IN MINUTES

0.0	3.2	20.0	10.0	8.2	17.9	14.0	18.0	17.1	17.2	4.5	14.6	11.7	11.7	10.2
3.2	0.0	12.8	6.3	11.7	10.0	14.4	10.2	14.1	12.2	6.3	6.3	8.2	4.5	10.0
20.0	12.8	0.0	10.8	18.4	4.0	16.1	12.2	10.8	6.3	16.1	8.2	10.2	8.5	14.1
10.0	6.3	10.8	0.0	8.0	10.0	8.5	14.4	8.0	7.2	5.7	8.9	2.0	4.5	4.5
8.2	11.7	18.4	8.0	0.0	18.0	6.3	21.5	11.3	13.4	5.7	16.5	8.2	12.2	4.5
17.9	10.0	4.0	10.0	18.0	0.0	17.1	8.2	12.8	8.5	14.6	4.5	10.2	6.3	14.1
14.0	14.4	16.1	8.5	6.3	17.1	0.0	22.8	6.3	10.0	10.2	17.2	7.2	12.8	4.5
18.0	10.2	12.2	14.4	21.5	8.2	22.8	0.0	20.0	16.1	16.5	5.7	15.6	10.0	18.9
17.1	14.1	10.8	8.0	11.3	12.8	6.3	20.0	0.0	4.5	12.6	14.4	6.0	10.8	7.2
17.2	12.2	6.3	7.2	13.4	8.5	10.0	16.1	4.5	0.0	12.8	10.8	5.7	8.0	8.9
4.5	6.3	16.1	5.7	5.7	14.6	10.2	16.5	12.6	12.8	0.0	12.0	7.2	8.2	6.0
14.6	6.3	8.2	8.9	16.5	4.5	17.2	5.7	14.4	10.8	12.0	0.0	10.0	4.5	13.4
11.7	8.2	10.2	2.0	8.2	10.2	7.2	15.6	6.0	5.7	7.2	10.0	0.0	5.7	4.0
11.7	4.5	8.5	4.5	12.2	6.3	12.8	10.0	10.8	8.0	8.2	4.5	5.7	0.0	8.9
10.2	10.0	14.1	4.5	4.5	14.1	4.5	18.9	7.2	8.9	6.0	13.4	4.0	8.9	0.0

OF STATES=

32805

OF FEASIBLE STATES =

10207

OPTIMAL VALUE FOR OBJ= 2 : 363.005

SUMMARY FOR VEHICLE : SPEED= 30.00 MPH, CAPACITY= 7 PERSONS

TIME (MINS)	LOCATION (MILES)		CUSTOMER SERVED	CUSTOMERS IN VEHICLE	CUSTOMERS TO BE PICKED UP
	X	Y			
0.0	3.000	3.000	VEHICLE AT DEPOT	0	7
4.472	2.000	1.000	# 7(PICKED UP)	1	6
10.797	1.000	4.000	# 5(PICKED UP)	2	5
22.459	6.000	7.000	# 2(PICKED UP)	3	4
26.931	7.000	5.000	# 7(DELIVERED)	2	4
31.403	9.000	6.000	# 5(DELIVERED)	1	4
35.875	10.000	4.000	# 6(PICKED UP)	2	3
39.875	10.000	2.000	# 3(PICKED UP)	3	2
46.199	7.000	1.000	# 3(DELIVERED)	2	2
50.672	5.000	0.0	# 2(DELIVERED)	1	2
56.672	5.000	3.000	# 6(DELIVERED)	0	2
58.672	5.000	4.000	# 4(PICKED UP)	1	1
64.328	3.000	6.000	# 4(DELIVERED)	0	1
68.801	2.000	8.000	# 1(PICKED UP)	1	0
86.801	11.000	8.000	# 1(DELIVERED)	0	0

TOTAL DISTANCE TRAVELLED : 43.40024 MILES

SUMMARY FOR EACH INDIVIDUAL CUSTOMER : MPS= 6

CUSTOMER	PICK-UP TIME	TIME IN VEHICLE	DELIVERY TIME	PICK-UP ORDER-SHIFT	DELIVERY ORDER-SHIFT	WEIGHTED WAITING TIME (ALFA=1.00)
1	68.801	18.000	86.801	7	-6	86.801
2	22.459	28.213	50.672	3	-1	50.672
3	39.875	6.325	46.199	5	-2	46.199
4	58.672	5.657	64.328	6	-2	64.328
5	10.797	20.606	31.403	2	3	31.403
6	35.875	20.797	56.672	4	2	56.672
7	4.472	22.459	26.931	1	6	26.931
TOTAL WEIGHTED WAITING TIME						363.005

of the "static" case, but now the "update" capability is added. Table E.6a shows how the program is started. "NMAX" is the maximum allowable number of customers to be processed simultaneously and is a user input. "DONE" counts how many customers have been fully serviced so far. "DECISION POINT" displays the Cartesian coordinates of the point of the current update. "RANDOM" is a user option, equal to 0 when the Cartesian coordinates as well as the times of the customer requests are deterministic, namely provided by the user. When RANDOM=1, the program generates itself the above data (locations and time of next request). The origins and destinations of the customers under this option are assumed to be uniformly distributed over a rectangular region with user-supplied boundaries and the times of customer requests are assumed to follow a negative exponential distribution, with a user-supplied rate of requests.

Table E.6b shows what happens when a new customer requests service. At that point in time, a new update is performed and a new tentative schedule is produced. This may continue indefinitely, as long as new customer requests keep appearing.

The program also plots the tentative routes as already shown in Figs. 9.2 to 9.11.

7) Dial-A-Ride-"Static" case/stage-by-stage version: This program implements the algorithm described in Chapter 10. As the program uses the "directories" described in that Chapter, auxiliary storage space must be reserved for them. The array NEXT is also kept in auxiliary storage. The output of this program is essentially the same as that of Table E.5.

8) Dial-A-Ride-"Directory" generating program. This program is not

interactive, in contrast to all the other programs described here. It creates the "directories," as described in Chapter 10, for any given size of problem. The directories are written on a user-specified device, such as magnetic tape or disk.

Table E.6(a): Typical output of the computer program for the dial-a-ride-"dynamic" case (Chapter 9).
This particular example corresponds to Case 2 of Chapter 9. This table depicts update #1.

```

TIME= 0.0 DECISION POINT: 1.000 4.000 RANDOM= 0
NMAX= 6 DONE= 0 N= 6
SPEED= 30.000 MPS= 3 CAP= 4 OBJ= 2 ALFA=1.00
BOUNDARIES OF REGION: 0.0 12.000 0.0 9.000
COORDINATES
CUSTOMER PICK-UP DELIVERY
1 1.000 2.000 5.000 2.000
2 3.000 1.000 1.000 8.000
3 5.000 4.000 2.000 6.000
4 6.000 3.000 4.000 7.000
5 10.000 8.000 11.000 6.000
6 8.000 1.000 7.000 5.000
TIME MATRIX:
0.0 4.47 8.94 10.20 21.63 14.14 8.00 12.00 8.25 11.66 21.54 13.42 4.00
4.47 0.0 7.21 19.80 10.00 4.47 14.56 10.20 12.17 18.87 11.31 7.21
8.94 7.21 0.0 2.83 12.81 8.49 4.00 11.31 7.21 6.32 12.65 4.47 8.00
10.20 7.21 2.83 0.0 12.81 5.66 2.83 14.14 10.00 8.94 11.66 4.47 10.20
21.63 19.80 12.81 12.81 0.0 14.56 15.62 18.00 16.49 12.17 4.47 8.49 19.70
14.14 10.00 8.49 5.66 14.56 0.0 6.32 19.80 15.62 14.42 11.66 8.25 15.23
8.00 4.47 4.00 2.83 15.62 6.32 0.0 14.42 10.00 10.20 14.42 7.21 8.94
12.00 14.56 11.31 14.14 18.00 19.80 14.42 0.0 4.47 6.32 20.40 13.42 8.00
8.25 10.20 7.21 10.00 16.49 15.62 10.00 4.47 0.0 4.47 18.00 10.20 4.47
11.66 12.17 6.32 8.94 12.17 14.42 10.20 6.32 4.47 0.0 14.14 7.21 8.49
21.54 18.87 12.65 11.66 4.47 11.66 14.42 20.40 18.00 14.14 0.0 8.25 20.40
13.42 11.31 4.47 4.47 8.49 8.25 7.21 13.42 10.20 7.21 8.25 0.0 12.17
4.00 7.21 8.00 10.20 19.70 15.23 8.94 8.00 4.47 8.49 20.40 12.17 0.0
# OF FEASIBLE STATES = 1882

```

OPTIMAL VALUE FOR OBJ= 2 : 232.051

```

TIME= 0.0 DECISION POINT: 1.000 4.000
TENTATIVE SCHEDULE FOR VEHICLE: SPEED= 30.00 MPH, CAPACITY= 4 PERSONS
TIME LOCATION (MILES) CUSTOMER CUSTOMERS CUSTOMERS TO
(MINS) X Y SERVED IN VEHICLE BE PICKED UP
0.0 1.000 4.000 DECISION POINT 0 6
4.000 1.000 2.000 # 1(PICKED UP) 1 5
8.472 3.000 1.000 # 2(PICKED UP) 2 4
12.944 5.000 2.000 # 1(DELIVERED) 1 4
15.773 6.000 3.000 # 4(PICKED UP) 2 3
18.601 5.000 4.000 # 3(PICKED UP) 3 2
25.812 2.000 6.000 # 3(DELIVERED) 2 2
30.284 1.000 8.000 # 2(DELIVERED) 1 2
36.609 4.000 7.000 # 4(DELIVERED) 0 2
48.774 10.000 8.000 # 5(PICKED UP) 1 1
53.246 11.000 6.000 # 5(DELIVERED) 0 1
64.908 8.000 1.000 # 6(PICKED UP) 1 0
73.155 7.000 5.000 # 6(DELIVERED) 0 0
TOTAL DISTANCE TRAVELLED: 36.57729 MILES

```

```

TENTATIVE SCHEDULE FOR EACH INDIVIDUAL CUSTOMER: MPS= 3
CUSTOMER PICK-UP TIME IN DELIVERY PICK-UP DELIVERY WEIGHTED WAITING
ORDER-SHIFT ORDER-SHIFT TIME (ALFA=1.00)
1 4.000 8.944 12.944 1 0 1 0 12.944
2 8.472 21.812 30.284 2 0 3 -1 30.284
3 18.601 7.211 25.812 4 -1 2 1 25.812
4 15.773 20.836 36.609 3 1 4 0 36.609
5 48.774 4.472 53.246 5 0 5 0 53.246
6 64.908 8.246 73.155 6 0 6 0 73.155
TOTAL WEIGHTED WAITING TIME 232.051
TO PLOT: TYPE 1, ERASE & RETURN. AFTER PLOT: MAKE HDCOPY, ERASE & RETURN
ELSE SIMPLY RETURN

```


Table E.6(b). Same as table E.6(a) but this depicts update #2.

ENTIRE SCHEDULE WILL BE OPTIMAL ONLY IF NO NEW CUSTOMER
APPEARS BEFORE TIME- 73.155
TYPE 1 IF NEW CUSTOMER APPEARS ELSE RETURN

1
ENTER TIME NEXT CUSTOMER APPEARS AND X-Y COORDS.
OF PICK-UP & DELIVERY POINTS

?

20 4 6 11 3

AT TIME- 20.000 NEW CUSTOMER 8 7 REQUESTS PICK-UP FROM 4.00 6.00 AND DELIVERY TO 11.00 3.00

ONLY THE PORTION OF THE ROUTE OF THE TENTATIVE SCHEDULE

UP TO TIME- 20.000 IS EXECUTED. AT THAT TIME A NEW DECISION IS TO BE TAKEN. DECISION POINT- 4.418 4.388

TIME- 20.000 DECISION POINT: 4.418 4.388 RANDOM- 0

NMAX- 6 DONE- 1 N- 6

SPEED- 30.000 MPS- 3 CAP- 4 OBJ- 2 ALFA-1.00

BOUNDARIES OF REGION: 0.0 12.000 0.0 9.000

COORDINATES

CUSTOMER	PICK-UP	DELIVERY
2	4.418 4.388	1.000 8.000
4	4.418 4.388	4.000 7.000
3	4.418 4.388	2.000 6.000
5	10.000 8.000	11.000 6.000
6	8.000 1.000	7.000 5.000
7	4.000 6.000	11.000 3.000

TIME MATRIX:

0.0	0.0	0.0	13.30	9.86	3.33	9.95	5.29	5.81	13.55	5.31	13.45	0.0
0.0	0.0	0.0	13.30	9.86	3.33	9.95	5.29	5.81	13.55	5.31	13.45	0.0
0.0	0.0	0.0	13.30	9.86	3.33	9.95	5.29	5.81	13.55	5.31	13.45	0.0
13.30	13.30	13.30	0.0	14.56	12.65	18.00	12.17	16.49	4.47	8.49	10.20	13.30
9.86	9.86	9.86	14.56	0.0	12.81	19.80	14.42	15.62	11.66	8.25	7.21	9.86
3.33	3.33	3.33	12.65	12.81	0.0	7.21	2.00	4.00	14.00	6.32	15.23	3.33
9.95	9.95	9.95	18.00	19.80	7.21	0.0	6.32	4.47	20.40	13.42	22.36	9.95
5.29	5.29	5.29	12.17	14.42	2.00	6.32	0.0	4.47	14.14	7.21	16.12	5.29
5.81	5.81	5.81	16.49	15.62	4.00	4.47	4.47	0.0	18.00	10.20	18.97	5.81
13.55	13.55	13.55	4.47	11.66	14.00	20.40	14.14	18.00	0.0	8.25	6.09	13.55
5.31	5.31	5.31	8.49	8.25	6.32	13.42	7.21	10.20	8.25	0.0	8.94	5.31
13.45	13.45	13.45	10.20	7.21	15.23	22.36	16.12	18.97	6.09	8.94	0.0	13.45
0.0	0.0	0.0	13.30	9.86	3.33	9.95	5.29	5.81	13.55	5.31	13.45	0.0

OF FEASIBLE STATES = 1882

OPTIMAL VALUE FOR OBJ- 2 : 167.107

TIME- 20.000 DECISION POINT: 4.418 4.388

TENTATIVE SCHEDULE FOR VEHICLE: SPEED- 30.00 MPH, CAPACITY- 4 PERSONS

TIME (MINS)	LOCATION (MILES) X	Y	CUSTOMER SERVED	CUSTOMERS IN VEHICLE	CUSTOMERS TO BE PICKED UP
20.000	4.418	4.388	DECISION POINT	3	3
23.331	4.000	6.000	# 7(PICKED UP)	4	2
25.331	4.000	7.000	# 4(DELIVERED)	3	2
29.803	2.000	6.000	# 3(DELIVERED)	2	2
34.275	1.000	8.000	# 2(DELIVERED)	1	2
52.275	10.000	8.000	# 5(PICKED UP)	2	1
56.747	11.000	6.000	# 5(DELIVERED)	1	1
62.747	11.000	3.000	# 7(DELIVERED)	0	1
69.958	8.000	1.000	# 6(PICKED UP)	1	0
78.204	7.000	5.000	# 6(DELIVERED)	0	0

TOTAL DISTANCE TRAVELLED : 39.10213 MILES

Table E.6(b) cont'd.

TENTATIVE SCHEDULE FOR EACH INDIVIDUAL CUSTOMER 1 MPS= 3							
CUSTOMER	PICK-UP TIME	TIME IN VEHICLE	DELIVERY TIME	PICK-UP ORDER-SHIFT	DELIVERY ORDER-SHIFT	WEIGHTED WAITING TIME (ALFA=1.00)	
2	20.000	14.275	34.275	1 0	3 -2	34.275	
4	20.000	5.331	25.331	2 1	1 2	25.331	
3	20.000	9.803	29.803	3 -1	2 0	29.803	
5	52.275	4.472	56.747	5 -1	4 0	56.747	
6	69.958	8.246	78.204	6 -1	6 -1	78.204	
7	23.331	39.416	62.747	4 2	5 1	62.747	
TOTAL WEIGHTED WAITING TIME						287.106	
TO PLOT: TYPE 1, ERASE & RETURN. AFTER PLOT: MAKE HDCOPY, ERASE & RETURN							
ELSE SIMPLY RETURN							