ALGORITHMS AND COMPUTER PROGRAMS IN DETERMINISTIC NETWORK OPTIMIZATION APPLIED TO PUBLIC SYSTEMS

by

Christoph E. MANDL

Research Memorandum No.126
February 1978

Abstract

This book reports on the state of the art in application of deterministic network optimization to public systems. After general remarks on problems of applying mathematical methods to decision problems in public systems in Chapter 1, the fundamental definitions for graphs and networks are presented in Chapter 2. In Chapter 3 the network flow problems are discussed: After presenting algorithms for the well known shortest path and maximum flow problem, the traffic assignment problem is discussed. In case of constant arc costs a minimum cost flow algorithm is presented. For the general case of arc costs, depending on the arc flow together with multiple origin-destination flow, a new algorithmic approach is presented that is based on Klein's minimum cost flow algorithm. The relationships between descriptive and normative assignment are shown. Chapter 4 deals with algorithms to find optimal subnetworks on a given network for the following problems:
Optimal waste water canal system and optimal filter plant location to minimize construction and operating costs; optimal location of emergency service facilities to minimize number of locations; optimal routes for airplanes to maximize number of people who fly non-stop from their origin to their destination; optimal spanning tree for an offshore oil-pipeline system to minimize construction and operating costs; optimal investment into a rail network to maximize transportation time reduction and optimal investment into a road network. Knowing the optimal network, it might also be of interest how this network should be constructed sequentially. In Chapter 5, an algorithm for finding the optimal construction sequence of a waste water canal network is presented, which maximizes the amount of purified water during construction period. The similar problem in case of a railway network is treated too.

Chapter 6 deals with the problem of finding routes with minimum length that either pass through all arcs (Chinese postman problem) or pass through all vertices (travelling salesman problem). In case the routes may not exceed a given length different heuristic algorithms are proposed. The applications to street cleaning, garbage collection as well as school bus routing are discussed. In the final Chapter 7 the problem of computing optimal routes for an urban public transportation system is discussed in detail, stating a new algorithm for solving it. To most of the presented algorithms computer programs are listed which are able to solve small up to medium sized problems.

Zusammenfassung

In diesem Buch wird der aktuelle Stand der Forschung bei der Anwendung deterministischer Netzwerk-Optimierung in öffentlichen Systemen dargestellt. Nach einleitenden Bemerkungen über die Probleme, welche bei der Anwendung mathematischer Methoden auf Entscheidungsfragen auftreten, werden in Kapitel 2 die notwendigen graphentheoretischen Begriffe eingeführt. Kapitel 3 befaßt sich mit Netzwerkfluß-Problemen: Nach der Darstellung der bekannten Algorithmen für kürzeste Wege und maximale Fluß-Probleme wird das Verkehrszuordnungs (traffic assignment) problem diskutiert. Im Fall konstanter Kantenkosten wird ein minimaler Kosten-Fluß (minimal cost flow) Algorithmus präsentiert. Für den allgemeinen Fall mit Kantenkosten, welche vom Kantenfluß abhängen, und mehrfachen Ursprungs-Ziel Flüssen wird ein neuer Algorithmus dargelegt, welcher eine Verallgemeinerung des minimalen Kosten-Fluß Algorithmus von Klein darstellt. Die Zusammenhänge zwischen deskriptiven und normativen Verkehrszuordnungen werden dargestellt. Kapitel 4 behandelt das Auffinden optimaler Subnetzwerke von gegebenen Netzwerken für die folgenden Fragestellungen: Optimales Abwasserkanalsystem und optimale Standorte von Kläranlagen zur Minimierung der Bau- und Betriebskosten; optimale Standorte von Rettungsautos zur Minimierung der Anzahl Standorte; optimale Flugzeugrouten zur Maximierung der Anzahl Fluggäste, welche non-stop reisen können; optimales Ölleitungsnetz im Meer zur Minimierung der Bau- und Betriebskosten; optimaler Ausbau von Schienen- und Straßennetzen zur Minimierung der Transportzeiten. Kennt man das optimale Netzwerk so stellt sich die Frage, in welcher Reihenfolge die einzelnen Kanten im Netz gebaut werden sollen. In Kapitel 5 wird dieses Problem im Falle des Abwasserkanalsystems (zur Maximierung des geklärten Wassers) und im Falle eines Schienennetzes (zur Minimierung der Transportzeiten) behandelt. Kapitel 6 beschäftigt sich mit der Frage optimaler Routen mit minimaler Länge, welche entweder jede Kante passieren (chinesisches Briefträgerproblem) oder jeden Knoten passieren

(Handelsreisendenproblem). Ist die Routenlänge indes beschränkt, so müssen heuristische Algorithmen verwendet werden. Die Anwendungen in der Straßenreinigung, Müllabfuhr und Schulbus-Routenplanung werden aufgezeigt. Im letzten Kapitel 7 geht es um die Frage, wie optimale Linienführung in Bus- oder Straßenbahnnetzen gefunden werden können. Ein neuer Algorithmus wird erläutert und seine Anwendung diskutiert.

Zu den meisten Algorithmen sind auch Computerprogramme beigefügt, welche kleine bis mittel große Probleme zu lösen imstande sind.

Acknowledgements

The origin of this work goes back to 1972, when my colleagues H.-J.Lüthi, A.Polyméris and myself, all working at the Department of Operations Research at the Swiss Federal Institute of Technology, Zürich, started to try to apply OR-methods to public systems: Therefore I want to thank especially H.-J.Lüthi and A.Polyméris for the many fruitful discussions on problems of OR-applications to public systems, as well as Prof.F.Weinberg, chairman of the department, for his strong support of our work. The final motivation to write this book was an invitation for giving a lecture on network optimization at the Universidade Catolica in Rio de Janeiro in 1977. Therefore, I also like to thank Prof.Paulo Dalcol, chairman of the Department of Industrial Engineering at Universidade Catolica, for his invitation, and all students attending my class for being such an interested audience. Also I have to thank the city of Rio de Janeiro itself, not only for the lovely beaches for recovering, but also for the stimulating atmosphere for working.

K.Plasser, Department of Mathematical Methods and Computer Science at the Institute for Advanced Studies, Vienna, gave me strong support when implementing the programs at the Institute for Advanced Studies. Without him, the programming phase would have taken more time trying to find Fortran-errors. Finally I want to thank Mrs.I.Hafner for carefully typing the manuscript and Mrs.E.Reischl for reading the draft and improving my English style substantially.

Vienna, February 1978

Contents

1. OR for public systems: an overview

Compared to the applications of OR to military problems and to industries, the applications of OR in governmental decision making are rather new and few. It was not until 1965 when an OR consulting firm was founded in Great Britain - the "Local Government OR Unit". From 1969 to 1975 the "New York City Rand Institute" operated also mainly on this purpose. A lot of articles and some books, e.g. Byrd (1975), Beltrami (1976), Drake (1972), Gass (1975), Greenberger (1976) and Weinberg et.al. (1976) appeared. However, the usefulness of an OR approach to problems in public systems has not been shown in all fields of public services so far. Following Byrd (1975), public decision making can be categorized into decisions on the

- operational level
- strategic level and
- political level.

Although there are differences between the public administration of different countries, it seems that in all hierarchies of public administration the people who hold a higher position are more interested in political decisions, less in strategic decisions and least of all in operational decision. However, it happens that OR applications have proved to be very successful on the operational level and little on the two other levels. It's probably partly because of this reason that people in public administration are not too enthusiastic about OR and that OR has not been really established in this field.

As OR techniques are a tool for making "better" decisions, quite naturally the costs of an OR study should be less than its benefits. Although cost-benefit analysis can be a difficult job in industries, it has been done for OR studies and for standard applications, it showed quite good results. Cost-benefit analysis for public systems is sometimes impossible. Although the costs of an OR study can be easily evaluated in terms of money, its benefits cannot for many applications be measured in terms of money. Or, how should one measure the savings for an improved transportation system that reduces transportation time, or an improved ambulance system that reduces the number of people killed by accidents or an improved waste water management system that results in cleaner water? Therefore the cost-benefit of OR applications in public systems in many cases depends on an individual and not always very rational trade-off between the costs in terms of money and the benefits measured in some other quantity. Thus, for the successful application of OR in government, it is necessary to find a person within public administration that somehow believes in quantitative methods like OR.

Still, besides all these difficulties, the number of OR applications in public systems is growing. One methodological area has proved especially useful for planning and decisions at the operational level, namely network optimization. In the last decade one has realised that transportation flows, road- and rail-systems, pipeline systems and other public systems can be modelled as networks and if such systems are to be planned or improved quite naturally lead to optimization problems on networks. It is the aim of this book to present results in this field, which were published in many different journals, in a compact and ordered form and to explore new application areas to which less attention has been given so far.

Therefore the approach chosen in this book will be problem oriented presenting the "easier" problems first and finally leading to the more complex problems. As a matter of fact transportation problems are slightly dominating, reflecting the great efforts currently undertaken in this area. It is the hope that this book will stimulate applications of network optimization in public systems, as well as research in this field as there are still enough unsolved urgent questions.

Besides presenting algorithms, including some research results of my own, and their applications to computer programs, nearly all algorithms are listed as well. This is done to enable the reader to solve smaller problems on his own and to give precise descriptions of the algorithms. I'm quite aware of the fact that the programs are not optimal as far as necessary storage and computing time are concerned. Therefore all programs are only applicable to problems on networks with about 5o vertices, some,like the traffic assignment program,only apply to networks with 14 vertices. All programs have been tested on a UNIVAC 11o6 using the ASCII-Fortran Compiler. No Input-Output Routines are presented in this book as they are of no importance and no computational results are given in general, because they depend too much on the structure of the problem. For any reports on the performance of the programs I shall be grateful.

2. Graphs, networks and combinatorial problems

In this section we want to introduce the basic definitions of graphs and networks, namely following Christofides (1975) and Steenbrink (1974). Besides we will be introducing the ideas of computational complexity of problems on graphs, which was presented in a paper by Karp (1975).

A graph is a collection of points or vertices $x_1, x_2, \ldots, x_n$ (denoted by the set X), and a collection of lines $a_1, a_2, \ldots, a_m$ (denoted by the set A) joining all or some of these points. The graph is then fully described and denoted by the doublet (X,A). If the lines in A have a direction - which is usually shown by an arrow - they are called arcs and the resulting graph is called a directed graph. If the lines have no orientation they are called links and the graph is non-directed. In Fig.2.1. an example of a directed and a non-directed graph is given.

An alternative way to describe a directed graph G , is by specifying the set X of vertices and a correspondence Γ which shows how the vertices are related to each other. Γ is called a mapping of the set X in X and the graph is denoted by the doublet $G = (X, \Gamma)$.

In the example of Fig.2.1.(a) we have

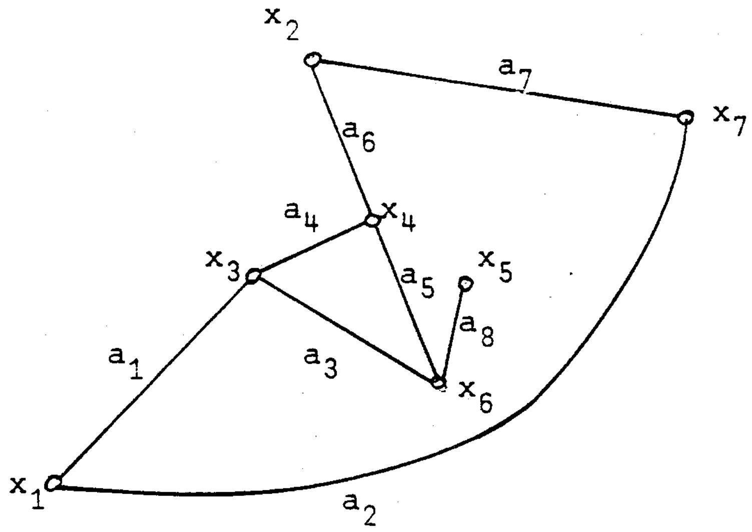
$$\Gamma(x_1) = \{x_3, x_7\}$$

$$\Gamma(x_4) = \{x_2, x_3, x_6\}$$

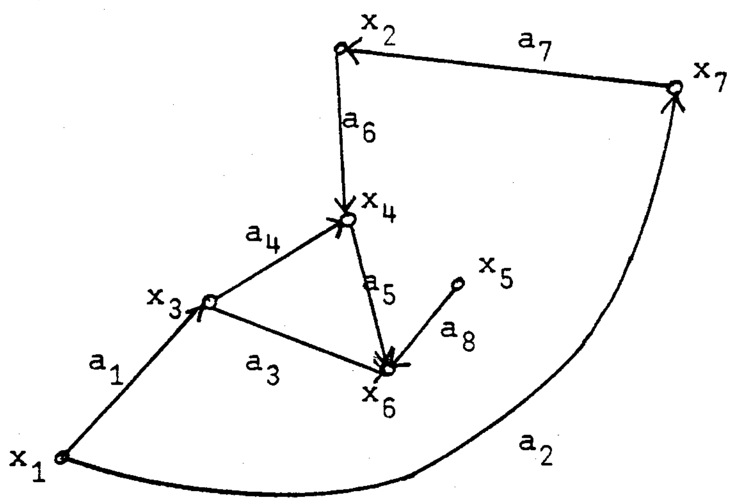
$$\Gamma(x_5) = \{x_6\}$$

$$\Gamma(x_6) = \{x_3, x_4, x_5\}$$

and of Fig.2.1.(b)



(a) Nondirected graph



(b) Directed graph

Fig. 2.1.

$$\Gamma(x_1) = \{x_3, x_7\}$$

$$\Gamma(x_4) = \{x_6\}$$

$$\Gamma(x_5) = \{x_6\}$$

$$\Gamma(x_6) = \{\emptyset\} .$$

A path in a directed graph is any sequence of arcs where the final vertex of one is the initial vertex of the next one. Thus in Fig. 2.1.(b) the sequence of arcs

a_2, a_7, a_6, a_5

a_1, a_4, a_5

a_1, a_3

are all paths.

Arcs $a=(x_i, x_j)$, $x_i \neq x_j$ which have a common terminal vertex are called adjacent. Also, two vertices x_i and x_j are called adjacent if either arc (x_i, x_j) or arc (x_j, x_i) or both exist in the graph. Thus in Fig.2.1.(b) arcs a_1, a_3 and a_4 are adjacent and so are the vertices x_1 and x_7.

A simple path is a path,which does not use the same arc more than once. Thus all paths in Fig.2.1.(b) are simple.

An elementary path is a path,which does not use the same vertex more than once. Thus all paths in Fig.2.1.(b) are elementary. Obviously an elementary path is also simple but the reverse is not necessarily true.

A chain is the nondirected counterpart of the path and applies to nondirected graphs like the one in Fig.2.1.(a). The definitions for simple and elementary chains are analogical to the definitions for paths.

A number c_{ij} may sometimes be assiocated with an arc (x_i,x_j). These numbers can be weights, lengths, costs, capacities or flows. Also a weight v_i may sometimes be associated with a vertex x_i. Following Steenbrink (1974), such weighted graphs are called networks.

Considering a path μ represented by the sequence of arcs $(a_1,a_2,...,a_9)$, the length (or cost) of the path $l(\mu)$ is taken to be the sum of the weights of the arcs appearing in μ, i.e.

$$l(\mu) = \sum_{(x_i,x_j) \text{ in } \mu} c_{ij} .$$

The cardinality of the path μ is the number of arcs appearing in the path.

A loop is an arc whose initial and final vertices are the same.

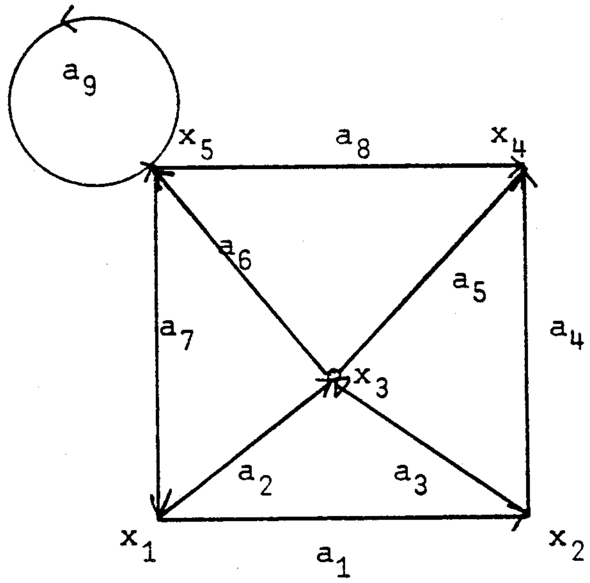


Fig.2.2.

In Fig.2.2.,for example,arc a_9 is a loop. A circuit is a path in which the initial vertex of the path coincides with the final vertex. Thus in Fig.2.2. the sequences

a_7, a_2, a_6

a_7, a_1, a_3, a_6

are circuits.

An elementary circuit does not use the same vertex more than once. In Fig.2.2. all circuits are elementary.

An elementary circuit which passes through all vertices of a given graph is called a Hamiltonian circuit. In Fig. 2.2. no Hamiltonian circuit exists.

A cycle is the counterpart of a circuit in a nondirect graph.

The number of arcs which have a vertex x_i as their initial vertex is called the outdegree of vertex x_i and similarly the number of arcs which have x_i as their final vertex is called the indegree of vertex x_i.

It is quite obvious that the sum of the outdegrees or indegrees of all the vertices in a graph is equal to the total number of arcs.

For a nondirected graph the degree of a vertex is similarly defined.

Given a graph $G = (X, A)$, a partial graph G_p of G is the graph (X,A_p) with $A_p \subset A$. Thus, a partial graph is a graph with the same number of vertices but with only a subset of the arcs of the original graph.

Given a graph $G = (X,\Gamma)$ a subgraph G_S is the graph (X_S,Γ_S) with $X_S \subset X$ and for every $x_i \varepsilon X_S$, $\Gamma_S(x_i) = \Gamma(x_i) \cap X_S$. Thus, a subgraph has only a subset X_S of the set of vertices of the original graph but contains all the arcs, whose initial and final vertices are both within this subset. The two definitions can be combined to define the partial subgraph. As an example see Fig.2.3.

If a graph represents a railway system with the vertices representing railway stations and the arcs representing the rails, then the graph representing only the main connections is a partial graph, the graph which represents only the railway system of a special region is a subgraph; and the graph which represents the main connections of the special region is a partial subgraph.

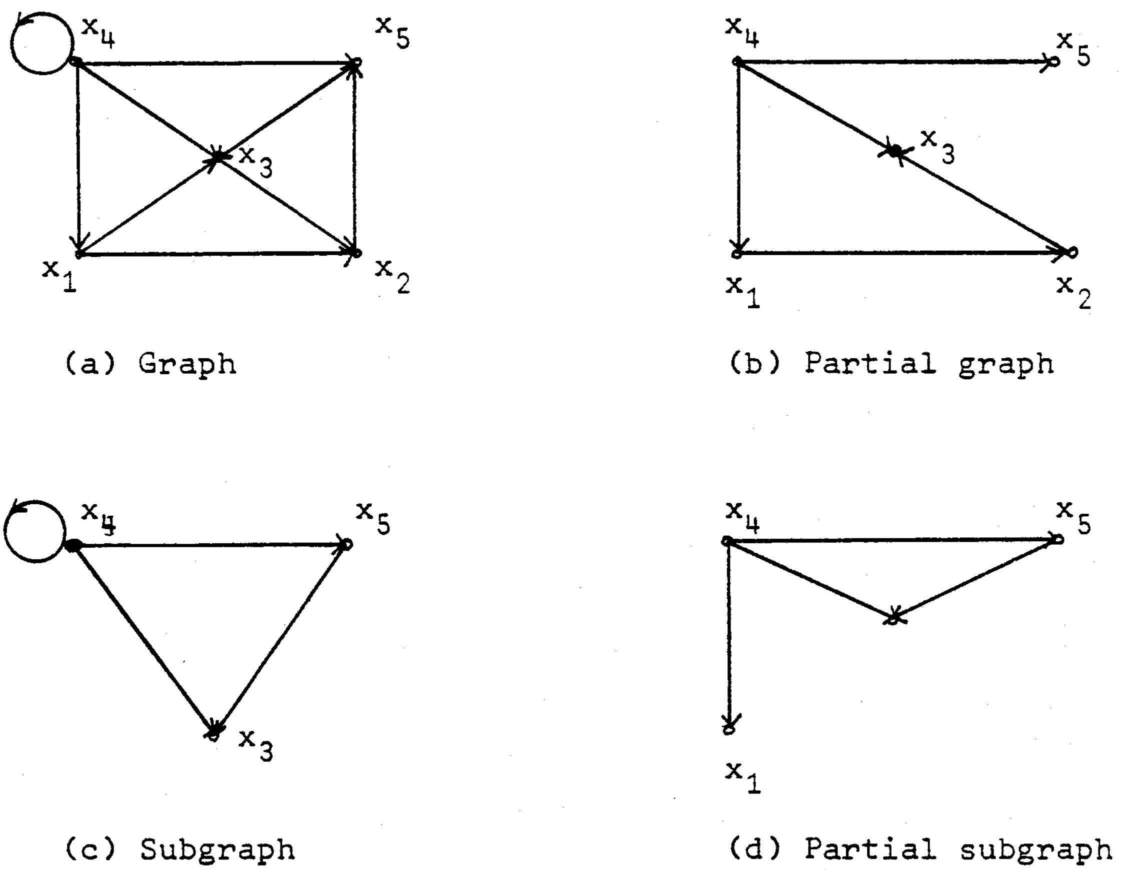


Fig.2.3.

A graph is said to be complete if all pairs of vertices are connected by at least one arc. A graph (X,A) is said to be symmetric if, whenever an arc (x_i,x_j) is in the set A of arcs, the opposite arc (x_j,x_i) also belongs to the set A.

An antisymmetric graph is one in which whenever an arc $(x_i,x_j)\in A$, the opposite arc $(x_j,x_i)\notin A$. Obviously an antisymmetric graph cannot contain loops.

A nondirected graph is bipartite, if the set of vertices X can be partitioned into two subsets X^a and X^b, so that all arcs have one terminal vertex in X^a and the other in X^b.

A graph is called connected (or strong) if for any two distinct vertices x_i and x_j, there is at least one path going from x_i to x_j. This definition implies that any two vertices of a strong graph are mutually reachable.

A graph is unilateral if for any two distinct vertices x_i and x_j, there is at least one path going either from x_i to x_j or from x_j to x_i.

A graph is called weak, if there is at least one chain joining every pair of distinct vertices. A graph that is not weak is called disconnected.

After this rather tiring number of definitions we come to the problem of specifying a graph. So far, we have been describing a graph in form of a picture. The other possibility, which is especially useful with graphs that have many vertices and few arcs, is by defining Γ, such that a graph $G=(X,\Gamma)$. This representation usually is the most compact form for handling a graph in a computer. Still, there are two other possibilities left:

Given a graph G, its adjacency matrix is denoted by $A=[a_{ij}]$ and is given by

$$a_{ij} = \begin{cases} 1 & \text{if arc } (x_i,x_j) \text{ exists in G} \\ 0 & \text{if arc } (x_i,x_j) \text{ does not exist in G} \end{cases}$$

Thus the adjacency matrix of the graph shown in Fig.2.3.(a) is

$$A = \begin{array}{c|ccccc|} & x_1 & x_2 & x_3 & x_4 & x_5 \\ \hline x_1 & 0 & 1 & 1 & 0 & 0 \\ x_2 & 0 & 0 & 0 & 0 & 1 \\ x_3 & 0 & 1 & 0 & 0 & 0 \\ x_4 & 1 & 0 & 1 & 1 & 1 \\ x_5 & 0 & 0 & 1 & 0 & 0 \\ \hline \end{array}$$

If one computes from a given adjaceney matrix A its square A^2 ,this gives all connections between pairs of vertices that exist via paths of cardinality 2 i.e. paths consisting of 2 arcs.

Given a graph G of n vertices and m arcs, the incidence matrix of G is denoted by $B= [b_{ij}]$ which is an n x m matrix and is defined by

$$b_{ij} = \begin{cases} 1 & \text{if } x_i \text{ is the initial vertex of arc } a_j \\ 0 & \text{if } x_i \text{ is not a terminal vertex of arc } a_j \text{ or} \\ & \text{if } a_j \text{ is a loop} \\ -1 & \text{if } x_i \text{ is the final vertex of arc } a_j \end{cases}$$

Therefore, the incidence matrix of the graph shown in Figur.2.3.(a), is

$$B = \begin{array}{c|ccccccccc|} & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & a_9 \\ \hline x_1 & 1 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ x_2 & -1 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 \\ x_3 & 0 & -1 & 1 & 0 & -1 & -1 & 0 & 0 & 0 \\ x_4 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ x_5 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 \\ \hline \end{array}$$

Since each arc is adjacent to exactly two vertices, each column of the incidence matrix contains one 1 and one -1 entry, except when the arc forms a loop. In that case it contains only zero entries.

If G is a nondirected graph, the incidence matrix is defined as above, except that all entries of -1 are changed to +1.

Now many combinatorial problems that are defined on graphs can with the help of the adjacency matrix be formulated as a linear integer optimization problem. Therefore, graph and network problems are strongly connected with integer programming. As is well known such problems tend to be difficult compared to say linear programming. Karp (1975) analyzed in detail the difficulty or computational complexity of combinatorial problems. He claims that in general such problems can be divided into two classes. In the first class are those problems which can be solved in polynomial time. That means that if a graph has n vertices, the computational time to solve the problem defined on this graph is in the worst case growing with $O(n^k)$ where k is some fixed integer number 1,2,... depending on the problem. Such problems, which are rather easy, are said to belong to the P-class. Problems of that kind are the matching problem, shortest paths and some network-flow problems.

In the second class, the so-called NP-class, are those problems which generally can only be solved in exponential time, i.e. the computational time is in the worst case growing with $O(k^n)$, where n is the number of vertices in the graph and k is some fixed integer. Clearly, NP-problems cause lot of troubles in trying to solve them, but as we shall see in this book many practical problems like general integer programming, travelling salesman, setcovering and others arc of the NP-type.

Thus, for large but difficult problems, approximation techniques and especially heuristic algorithms are of great importance and often run with great success, although the reason for this is still unknown and as Karp (1975) believes: "The ultimate explanation of this phenomenon will undoubtedly have to be probabilistic".

Besides, branch-and-bound methods and dynamic programming have to be recognized as useful tools for solving combinatorial problems.

An informative bibliography of recent literature on combinatorial problems in connection with network optimization can be found in Golden and Magnanti (1977).

3. Network flow problems

In this somehow preliminary chapter we will discuss problems that arise within a given network in which there are flows from vertex to vertex. Such flows can be cars within a road network, where the roads are represented by the arcs and the vertices represent cities. The same model applies to public tranportation systems, like railway systems or bus systems. But one can also think of flows in connection with a waste water canal system or an oil pipeline system. But the following questions are really important in transportation systems, especially on road systems.

3.1. Shortest path

Given a road network, each driver is confronted with the problem of finding the best way from his present location to the one he wishes to go. As an intuitively quite obvious objective he wishes to reach this point in the shortest time possible. Of course, other objectives might influence him as well, like travelling rather through beautiful countryside than through dirty industrial zones. Thus, the stated objective of minimizing travel time is a simplification, but one which in fact has proved to model drivers behaviour pretty well. Therefore, the first step of modelling drivers behaviour within a given road network is to find the shortest path between

(i) a given vertex s and another vertex t in the graph

(ii) a given vertex s and all other vertices $x_i \in X$, where X is the set of vertices and

(iii) all pairs of vertices.

The weight or length of an arc represents in this case the travel time needed to travel along this specific arc. Naturally we can assume the length of all arcs $c_{ij} \geq 0$, as no negative travel time is possible.

The most efficient algorithm for the solution of the under (i) and (ii) stated shortest path problems was given initially by E.Dijkstra.

Dijkstra's Algorithm:

Let $l(x_i)$ be the label on vertex x_i.

Step_1 (Initialization):

Set $l(s) = 0$ and mark the label as permanent. Set $l(x_i) = \infty$ for all $x_i \neq s$ and mark these labels temporarily. Set $p=s$.

Step_2 (Updating of labels):

For all $x_i \in \Gamma(p)$ and which have temporary labels, update the labels according to

$$l(x_i) = \min\left[l(x_i),\ l(p) + c(p,x_i)\right] \quad , \tag{3.1}$$

where $c(p,x_i)$ is the travel time on the arc (p,x_i).

Step_3 (Fixing a label as permanent):

Of all temporarily labelled vertices find x_i^* for which $l(x_i^*) = \min\left[l(x_i)\right]$.
Mark the label of x_i^* permanent and set $p=x_i^*$.

Step_4 (Termination):

(i) (If only the path from s to t is desired)
If $p=t$, $l(p)$ is the required shortest path length. STOP.
If $p \neq t$, go to Step 2.

(ii) (If the path from s to every other vertex is required)
If all the vertices are permanently labelled, then the labels are the lengths of the shortest paths. STOP.
If some labels are temporary, go to Step 2.

Once the shortest path lengths from s are obtained as the final values of the vertex labels, the paths themselves can be obtained by a recursive application of (3.2) below. Thus if x_i' is the vertex just before x_i in the shortest path from s to x_i, then for any given vertex x_i, x_i' can be found as the one of the remaining vertices for which

$$l(x_i') + c(x_i', x_i) = l(x_i) \qquad (3.2)$$

The proof that the above algorithm indeed produces the shortest paths is quite simple and well known. Thus we do not state the proof here; one may read it for example in Christofides (1975).

For the case of a n-vertex completely connected graph, where the shortest paths between s and all other vertices are required, the algorithm involves $n(n-1)/2$ additions and comparisons at Step 2 and another $n(n-1)/2$ comparisons at Step 3. Additionally, at Steps 2 and 3, it is necessary to determine which vertices are temporarily labelled, which requires an extra $n(n-1)/2$ comparison . These figures are also upper bounds on the number of operations necessary to find the shortest path from s to a specified t, and can in fact be realized if t happens to be the last vertex to be permanently labelled.

```
C ... *** DIJKSTRA'S ALGORITHM FOR SHORTEST PATH(S)
C ... ***
C ...
C ... INPUT
C ...
C ... N     NUMBER OF VERTICES
C ... IG    VECTOR DENOTING THE LENGTHS OF THE ARCS.
C           THE VECTOR HAS LENGTH N*N.
C ... B     B = TRUE MEANS THAT SHORTEST PATH FROM S
C           TO T IS WANTED ONLY. B = FALSE DENOTES THAT
C           SHORTEST PATH FROM S TO ALL OTHER VERTICES
C           IS WANTED
C ... S     ORIGIN VERTEX
C ... T     DESTINATION VERTEX (ONLY NEEDED IF B = FALSE)
C ...
C ... OUTPUT
C ...
C ... L(I)  LENGTH OF SHORTEST PATH FROM S TO I
C
      SUBROUTINE SPI(N.IG,L.B,S,T)
      INTEGER N,IG(1),L(1),S,T,P
      LOGICAL B
C ...
C ... STEP 1
C ...
      L(1)=-2**15
      DO 5 I=2,N
      J=I-1
    5 L(I)=L(J)
      L(S)=0
      P=S
C ...
C ... STEP 2
C ...
    1 DO 10 I=1,N
      IF(L(I) .GE. 0) GO TO 10
      J=IND(P,I,N)
      IF(IG(J) .LE. 0) GO TO 10
      M=-L(P)-IG(J)
      L(I)=MAX0(L(I),M)
   10 CONTINUE
C ...
C ... STEP 3
C ...
      M=-2**15
      DO 15 I=1,N
      IF(L(I) .GE. 0) GO TO 15
      IF(L(I) .LE. M) GO TO 15
      M=L(I)
      P=I
   15 CONTINUE
      L(P)=-M
      IF(.NOT.B) GO TO 30
C ...
C ... STEP 4 (I)
C ...
```

```
      IF(P .NE. T) GO TO 1
      GO TO 25
C ...
C ... STEP 4 (II)
C ...
   30 DO 20 I=1,N
      IF(L(I) .LT. 0) GO TO 1
   20 CONTINUE
   25 CONTINUE
      RETURN
      END
```

```
      FUNCTION IND(I,J,N)
      INTEGER I,J,N
      IND=(I-1)*N+J
      RETURN
      END
```

When the shortest paths between all pairs of vertices of a graph are required, an obvious way for obtaining the answer is to apply Dijkstra's algorithm n times, each time with a different starting vertex s. In the case of a complete graph, the resulting calculation time would be proportional to n^3. We now describe a different approach to the problem. The following method requires computation time proportional to n^3, but is in general about 5o % faster than the application of Dijkstra's algorithm n times. This algorithm was first described by R.W.Floyd.

Floyd's Algorithm:

It is assumed that the matrix of the arc lengths c_{ij} has been initialized so that $c_{ii} = 0$ for all $i=1,2,\ldots,n$, and $c_{ij}=\infty$, whenever arc (x_i,x_j) is not in the graph G.

Step 1 (Initialization):

Set $k = 0$

Step 2 (Iteration):

Set $k = k+1$.
For all $i \neq k$ such that $c_{ik} \neq \infty$ and all $j \neq k$ such that $c_{kj} \neq \infty$, perform

$$c_{ij} = \min\,[c_{ij},\ (c_{ik}+c_{kj})] \tag{3.3}$$

Step 3 (Termination):

(a) If $k=n$, the solution has been reached and $[c_{ij}]$ gives the lengths of all shortest paths. Stop.

(b) If $k < n$, return to Step 2.

The shortest paths themselves can once more be obtained from the shortest path lengths using a recursive relation similar to (3.2) .

Alternatively, a bookkeeping mechanism can be used to record (concurrently with the shortest path lengths) information about the paths themselves. The technique involves the storage and updating of a second n x n - matrix $D=(d_{ij})$ in addition to the cost matrix C. The entry d_{ij} implies that d_{ij} is the vertex just before vertex x_j on the shortest path from x_i to x_j. The matrix D is initialized so that $d_{ij} = x_i$ for all x_i and x_j.

Following (3.3) in Step 2 of the algorithm one would then introduce the updating of matrix D as follows

$$d_{ij} = \begin{cases} d_{kj}, \text{ if } (c_{ik} + c_{kj}) < c_{ij} \text{ in } (3.3) \\ \text{unchanged, if } c_{ij} \leq (c_{ik} + c_{kj}) \end{cases}$$

At the end of the algorithm the shortest path can be obtained immediately from the final D matrix. Thus, if the shortest path between any two vertices x_i and x_j is required, this path is given in the vertex sequence

$$x_i, x_\nu, \ldots, x_\gamma, x_\beta, x_\alpha, x_j$$

where $x_\alpha = d_{ij}$, $x_\beta = d_{i\alpha}$, $x_\gamma = d_{i\beta}$ etc. until finally $x_i = d_{i\nu}$.

It should perhaps be pointed out here that had all c_{ii} been initialized to ∞ (instead of at 0), at the start of the algorithm, then the final values of c_{ii} would be the cost of the shortest circuit through vertex x_i.

```
C ... *** FLOYD'S ALGORITHM FOR SHORTEST PATHS
C ... ***
C
C ... INPUT
C
C ... N      NUMBER OF VERTICES
C ... IG(L) LENGTH OF ARC FROM I TO J, WHERE L=(I-1)*N+J
C            IF IG(L)=0 THEN ARC(I,J) DOES NOT EXIST
C
C ... OUTPUT
C
C ... IG     LENGTH OF SHORTEST PATHS
C ... D      VECTOR FOR FINDING THE VERTICES BELONGING
C            TO THE SHORTEST PATHS
C ... C      IF C = TRUE A CIRCUIT WITH NEGATIVE LENGTH
C            EXISTS
C
      SUBROUTINE SPII(N,IG,D,C)
      INTEGER N,IG(1),D(1)
      LOGICAL C
      C=.FALSE.
C
C ... STEP 1
C
      DO 20 I=1,N
      DO 25 J=1,N
      K=IND(I,J,N)
   25 D(K)=I
   20 CONTINUE
      M=N*N
      DO 22 I=1,M
      IF(IG(I) .EQ. 0) IG(I)=2**34
   22 CONTINUE
C
C ... STEP 2 AND 3
C
      DO 5 K=1,N
      DO 10 I=1,N
      L=IND(I,K,N)
      IF(I.EQ.K .OR. IG(L).EQ.2**34) GO TO 10
      DO 15 J=1,N
      L1=IND(K,J,N)
      IF(J.EQ.K .OR. IG(L1).EQ.2**34) GO TO 15
      M=IG(L)+IG(L1)
      L2=IND(I,J,N)
      IF(M .LT. IG(L2)) D(L2)=D(L1)
      IG(L2)=MIN0(IG(L2),M)
      IF(IG(L2) .LT. 0 .AND. I.EQ.J ) GO TO 30
   15 CONTINUE
   10 CONTINUE
    5 CONTINUE
      GO TO 35
   30 C=.TRUE.
   35 CONTINUE
      RETURN
      END
```

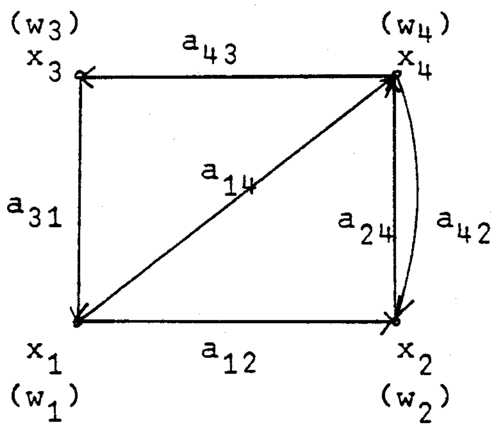
3.2. Maximum flow:

In designing a future road network, the traffic engineer has to determine the capacities of streets and intersections as a function of road widths, number of lanes, shoulder widths, gradients, traffic signalization, etc. Depending on the level of service provided, capacities are chosen and the future network tested by checking whether the estimated traffic exceeds any capacities. If so, more capacity can be provided by designing improved facilities. In this context a capacity q_{ij} is associated with every arc (x_i, x_j) of a given network G, and this capacity represents the largest amount of flow that can be transmitted along the arc, where flow here means the number of vehicles per hour. It then raises the major question how many vehicles per hour can travel from a vertex s to a different vertex t, which is the so called maximal flow problem. A solution to this problem also indicates the parts of the road network, which are saturated and form a bottleneck as far as the flow between two given locations is concerned.

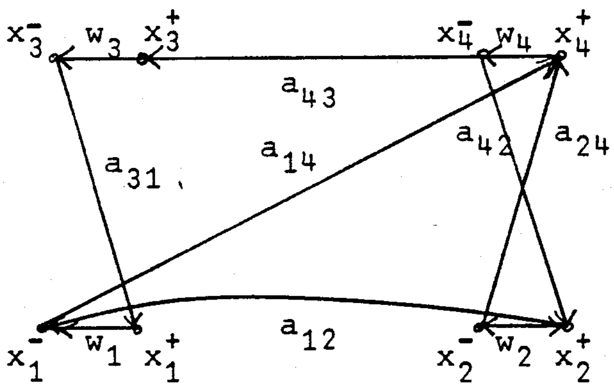
An interesting point worth noting is that it is the intersections rather than the streets, which are potential bottlenecks in a city street network. The emphasis on link capacities in flow theory is more appropriate to a main road network where the vertices are not of direct traffic significance. But it is not difficult to include vertex capacities - they are easily reduced by representing a capacitated vertex by two vertices joined by one capacitated dummy arc in the following way:

Let the maximum flow between vertices s and t of a network G be required. Define a network G_o, so that every vertex x_j of network G corresponds to two vertices x_j^+ and x_j^- in the network G_o, in such a way that for every arc (x_i, x_j) of G incident to x_j corresponds an arc (x_i^-, x_j^+) of G_o incident to x_j^+ and for every arc (x_j, x_k) of G emanating from

x_j corresponds an arc (x_j^-, x_k^+) of G_o emanating from x_j^- . Moreover, an arc between x_j^+ and x_j^- of capacity w_j (the capacity of vertex x_j) is introduced. As an example see Fig.3.1.



(a) Graph with vertex and arc capacities



(b) Equivalent graph with arc capacities only

Fig. 3.1.

Sometimes it is of interest not only to know the maximum flow between vertex s and t but between n_s source vertices to n_t sink vertices where flow can go from any source to any sink. This problem can be converted to the simple (s to t) maximum flow problem by adding a new artificial source vertex s and sink vertex t with added arcs leading from s to all n_s source vertices and from every sink to t as given in Fig.3.2.

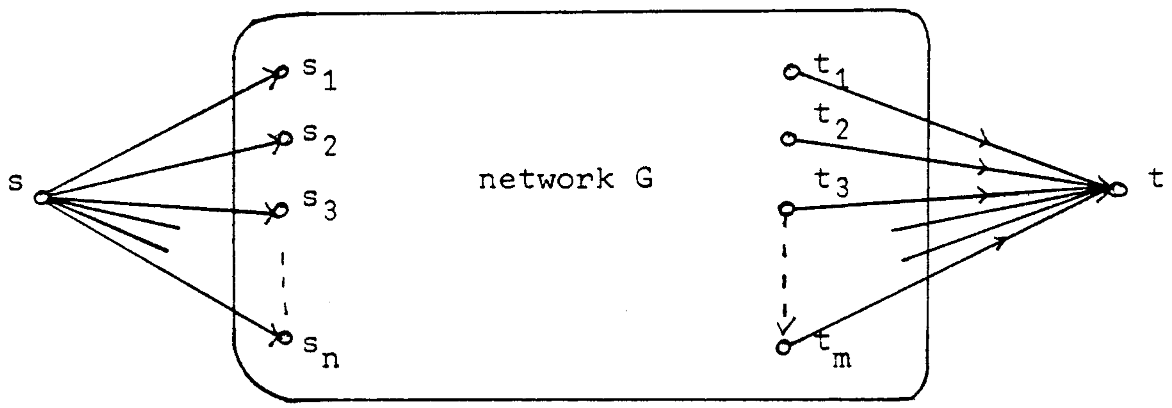


Fig.3.2.

We shall now state the maximum flow problem mathematically.

Consider the network G=(X,A) with integer arc capacities q_{ij}, a source vertex s and a terminal vertex t (called sink): (s and tεX) . A set of numbers f_{ij} defined on the arcs $(x_i,x_j)\varepsilon A$ are called flows in the arcs if they satisfy the following conditions:

$$\sum_{x_j \varepsilon \Gamma(x_i)} f_{ij} - \sum_{x_k \varepsilon \Gamma^{-1}(x_i)} f_{ki} = \begin{cases} v & \text{if } x_i = s \\ -v & \text{if } x_i = t \\ 0 & \text{if } x_i \neq s \text{ or } t \end{cases} \qquad (3.4)$$

and

$$0 \leq f_{ij} \leq q_{ij} \qquad \text{for all } (x_i, x_j) \in A \qquad (3.5)$$

Equation (3.4) is an equation of conservation of flow (also known as Kirchhoff's law) and states that the flow into a vertex x_i is equal to the flow out of the same vertex, except for the source and sink vertices s and t, for which there is a net out flow and inflow of value v respectively. Equation (3.5) simply states the capacity constraint for each arc of the network G. The objective is to find a set of arc flows, so that

$$v = \sum_{x_j \in \Gamma(s)} f_{sj} = \sum_{x_k \in \Gamma^{-1}(t)} f_{kt} \qquad (3.6)$$

is maximized, where f_{sj} and f_{kt} are written for the flows from vertex s to x_j and from x_k to t respectively.

Before now presenting the algorithm, we have first to introduce the definition of a cut-set, which we shall need for a theorem stating the similarity between maximum flow and minimum cut. This theorem will be the basis of the algorithm.

If the set of vertices X of a graph G=(X,A) is partitioned into two complementary sets X_o, $\tilde{X}_o$, then the subset of A defined by

$$(X_o, \tilde{X}_o) = \{(x_i, x_j) | (x_i, x_j) \in A, \ x_i \in X_o, \ x_j \in \tilde{X}_o\}$$

is called a cut-set. We emphasize the fact that a cut-set is a subset of directed links. For the graph illustrated in Fig.3.3 .

$$(X_o, \tilde{X}_o) = \{a_2, a_5, a_6, a_8\}$$

and

$$(\tilde{X}_o, X_o) = \{a_3\}$$

are cut-sets with $X_o = \{x_1, x_4\}$ and $\tilde{X}_o = \{x_2, x_3\}$.

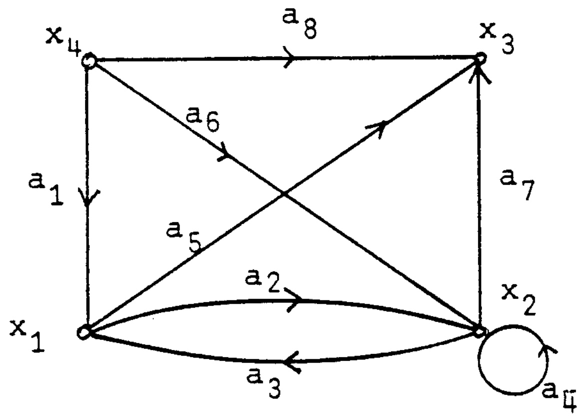


Fig.3.3.

Maximum-flow minimum-cut theorem:

The value of the maximum flow from s to t is equal to the value of the minimum cut-set $(X_m, \tilde{X}_m)$ separating s from t.

A cut-set $(X_o, \tilde{X}_o)$ separates s from t if $s \varepsilon X_o$ and $t \varepsilon \tilde{X}_o$. The value of such a cut-set is the sum of the capacities of all arcs belonging to the cut-set; i.e.

$$v(X_o, \tilde{X}_o) = \sum_{(x_i, x_j) \varepsilon (X_o, \tilde{X}_o)} q_{ij}$$

The minimum cut-set $(X_m, \tilde{X}_m)$ is then the cut-set with the smallest such value.

Proof:
A constructive proof of the theorem is given and the method of construction immediately suggests the labelling algorithm which follows.

Obviously, the maximum flow from s to t cannot be greater than $v(X_m,\tilde{X}_m)$, since all paths leading from s to t use one of the arcs of this cut-set. The aim of the proof is therefore, to show that a flow exists which attains this value. Let us now assume a flow given by the m-dimensional vector f with the elements all nonnegative integers and define a cut set $(X_o,\tilde{X}_o)$ by recursively applying step (b) below:

(a) Start by setting $X_o = \{s\}$

(b) If $x_i \varepsilon X_o$, and either $f_{ij} < q_{ij}$, or $f_{ji} > 0$ place x_j in the set X_o and repeat the step until X_o cannot be increased further

Then two cases can occur, either $t \varepsilon X_o$ or $t \not\varepsilon X_o$.

Case 1 ($t \varepsilon X_o$):
According to step (b) above $t \varepsilon X_o$ implies that a chain of arcs from vertex s to vertex t exists, so that for every arc (x_i,x_j) used by the chain in the forward direction (forward arcs), $f_{ij} < q_{ij}$; and for every arc (x_k,x_l) used by the chain in the backward direction i.e. in the direction from x_l to x_k (backward arcs), $f_{kl} > 0$. (This chain of arcs will be called a flow-augmenting chain).

Let

$$d_f = \min_{(x_i,x_j)} (q_{ij}-f_{ij}), \quad (x_i,x_j) \text{ forward}$$

$$d_b = \min_{(x_k,x_l)} (f_{kl}), \quad (x_k,x_l) \text{ backward}$$

$$d = \min (d_f,d_b) = \text{positive integer}$$

If now d is added to the flow in all forward arcs and subtracted from the flow in all backward arcs of the chain, the net result is a new feasible flow with a value d units greater than the previous one. This is apparent, since the addition of d to the flow in the forward arcs cannot violate any of the arc capacities of these arcs (since $d \leq d_f$) and the substraction of d from the flow in the backward arcs cannot make the flow in these arcs negative (since $d \leq d_b$).

Using the new improved flow, one can then reapply steps (a) and (b) above to define a new cut set $(X_o, \tilde{X}_o)$ and repeat the argument.

Case 2 $(t \notin X_o)$:

If $t \in \tilde{X}_o$ then according to step (b) $f_{ij} = q_{ij}$ for all $(x_i, x_j) \in (X_o, \tilde{X}_o)$ and $f_{kl} = 0$ for all $(x_k, x_l) \in (\tilde{X}_o, X_o)$.

Hence

$$\sum_{(x_i,x_j)\in(X_o,\tilde{X}_o)} f_{ij} = \sum_{(x_i,x_j)\in(X_o,\tilde{X}_o)} q_{ij}$$

and

$$\sum_{(x_k,x_l)\in(\tilde{X}_o,X_o)} f_{kl} = 0.$$

Therefore the value of the flow which is

$$\sum_{(x_i,x_j)\in(X_o,\tilde{X}_o)} f_{ij} - \sum_{(x_k,x_l)\in(\tilde{X}_o,X_o)} f_{kl}$$

is equal to the value of the cut $(X_o, \tilde{X}_o)$.

Since in case 1 the flow is continuously increased by at least one unit, then assuming all q_{ij} are finite integers, the maximum flow must be obtained in a finite number of steps when case 2 occurs. That flow then equals the value of the current cut (X_o,X_o) which must therefore be the minimium cut. As a result of this proof the following algorithm can now be stated.

<u>Labelling algorithm for the (s to t) maximum flow problem:</u>

The algorithm starts with an arbitrary feasible flow (zero flow may be used) and then tries to increase the flow value systematically, searching all possible flow-augmenting chains from s to t. The search for a flow-augmenting chain is carried out by attaching labels to vertices indicating the arc along which the flow may be increased and by how much. Once such a chain is found, the flow along it is increased to its maximum value, all vertex labels are erased and the new flow is used as a basis for relabelling. When no flow-augmenting chain can be found the algorithm terminates with the maximal flow.

A. The labelling process:

A vertex can only be in one of three possible states; labelled and scanned (i.e. it has a label and all adjacent vertices have been processed), labelled and unscanned (i.e. it has a label but not all its adjacent vertices have been processed) and unlabelled. A label on a vertex x_i is composed of two parts and takes one of the two forms $(+x_j,d)$ or $(-x_j,d)$. The part $+x_j$ of the first type of label implies that the flow along arc (x_j,x_i) can be increased. The part $-x_j$ of the alternative type of label implies that the flow along arc (x_i,x_j) can be decreased. d represents in both cases the maximum amount of extra flow that can be sent from s to x_i along the augmenting chain being constructed. The labelling of vertex x_i corresponds to finding a flow-augmenting chain from s to x_i.

Initially all vertices are unlabelled.

Step_1: Label s by $(+s,\ d_s = \infty)$. s is now labelled and unscanned and all other vertices are unlabelled.

Step_2: Choose any labelled unscanned vertex x_i and suppose its label is $(\pm\ x_k, d_i)$.

(i) To all vertices $x_j \varepsilon \Gamma(x_i)$ that are unlabelled for which $f_{ij} < q_{ij}$ attach the label $(+x_i, d_j)$ where

$$d_j = \min\ (d_i, q_{ij} - f_{ij})$$

and

(ii) To all vertices $x_j \varepsilon \Gamma^{-1}(x_i)$ that are unlabelled and for which $f_{ji} > 0$ attach the label $(-x_i, d_j)$ where

$$d_j = \min(d_i, f_{ji}).$$

(The vertex x_i is now labelled and scanned and the vertices x_j labelled by (i) and (ii) are labelled and unscanned). Indicate that x_i is now scanned by marking it in some way.

Step_3: Repeat Step 2 until either t is labelled,in which case proceed to Step 4 ,or t is unlabelled and no more labels can be placed,in which case the algorithm terminates with f as the maximum flow vector. It should be noted here that if X_o is the set of labelled vertices and $\tilde{X}_o$,the set of unlabelled ones then $(X_o, \tilde{X}_o)$ is the minimum cut.

B. Flow augmenting process:

Step_4: Let x=t and got to Step 5.

Step_5:

(i) If the label on x is of the form $(+z, d_x)$, change the flow along the arc (z,x) from f_{zx} to $f_{zx} + d_t$.

(ii) If the label on x is of the form $(-z,d_x)$ change the flow along the arc (x,z) from f_{xz} to $f_{xz} - d_t$.

Step 6:

If z=s, erase all labels and go to Step 1 to repeat the labelling process starting from the new improved flow calculated in Step 5.

If $z \neq s$ set x=z and go to Step 5.

```
C ... *** MAXIMUM FLOW ALGORITHM
C ... ***
C
C ... INPUT
C
C ... N      NUMBER OF VERTICES
C ... Q(L)  CAPACITY OF ARC(I,J), WHERE L=(I-1)*N+J
C ... S      ORIGIN VERTEX OF FLOW
C ... T      DESTINATION VERTEX OF FLOW
C ... V      IF V>0, FLOW OF VALUE V IS FOUND
C
C ... OUTPUT
C
C ... F(L)  FLOW ON ARC(I,J), WHERE L=(I-1)*N+J
C
      SUBROUTINE MAXFLO(N,Q,F,S,T,V)
      INTEGER Q(1),F(1),S,T,N,L1(52),L2(52),X,V,VV
C
C ... STEP 1
C
       VV=0
      M=N*N
      DO 10 I=1,M
   10 F(I)=0
   15 DO 5 I=1,N
      L1(I)=0
    5 L2(I)=-1
      L1(S)=S
      L2(S)=-2**18
C
C ... STEP 2
C
   20 DO 25 J=1,N
      IF(L1(J).EQ.0 .OR. L2(J).GT.0) GO TO 25
      I=J
      GO TO 30
   25 CONTINUE
      RETURN
   30 L2(I)=-L2(I)
C
C ... STEP 2 (I)
C
      DO 35 J=1,N
      IF(I.EQ.J) GO TO 35
      L=IND(I,J,N)
      IF(Q(L).LE.F(L) .OR. L1(J).NE.0) GO TO 35
      L1(J)=I
      K=Q(L)-F(L)
      L2(J)=-MIN0(L2(I),K)
   35 CONTINUE
C
C ... STEP 2 (II)
C
      DO 40 J=1,N
      IF(I.EQ.J) GO TO 40
      L=IND(J,I,N)
```

```
      IF(F(L).EQ.0 .OR. (Q(L).EQ.0 .OR.L1(J).NE.0)) GO TO 40
      L1(J)=-I
      L2(J)=-MINO(L2(I),F(L))
   40 CONTINUE
C
C ... STEP 3
C
      IF(L1(T).EQ.0) GO TO 20
C
C ... STEP 4
C
      X=T
       IF(V .LE. 0) GO TO 45
       VV=VV+IABS(L2(T))
       IF(VV .LE. V) GO TO 45
       VV=VV-IABS(L2(T))
       L2(T)=V-VV
       VV=V
   45 IF(L1(X).LT.0) GO TO 50
C
C ... STEP 5 (I)
C
      M=L1(X)
      L=IND(M,X,N)
      F(L)=F(L)+IABS(L2(T))
      GO TO 55
C
C ... STEP 5 (II)
C
   50 M=-L1(X)
      L=IND(X,M,N)
      F(L)=F(L)-IABS(L2(T))
C
C ... STEP 6
C
55     IF(V.GT.0 .AND. (VV.EQ.V .AND. M.EQ.S)) RETURN
      IF(M.EQ.S) GO TO 15
      X=M
      GO TO 45
      END
```

3.3. Traffic assignment

Given a road network with many source and sink vertices, the problem of finding out how much flow to assign to each arc of the network, such that the conservation equations of (3.4.) are satisfied, is called traffic assignment. It is obvious that there are many solutions to this general assignment or, to put it another way, that it is possible to apply further criteria to the assignment. Two approaches seem particularly interesting for practical problems, namely the descriptive and the normative assignment. Descriptive assignment tries to model the flows, the way car drivers would behave in real road networks. Normative assignment tries to model the flows, such that it would be best for all drivers. In this sense descriptive also means optimal to the individual driver and normative means optimal to the drivers as a society. These two approaches are stated in the Wardrop principles, which give the following criteria for determining the distribution of traffic

(i) Descriptive assignment:
The journey time on all routes actually used are equal and less than those which would be experienced by a single vehicle on any unused route.

(ii) Normative assignment:
The average journey time is a minimum. Of course, as we already discussed earlier, travel time will not be the only decision variable to be considered. But if we interpret the weights on arcs not just as journey time but more general as journey costs, these parameters could be not only a measure for time but also implicitely include other important factors, like quality of the road, scenery, noise

and others. Thus we shall rather talk about travel costs than time.

In the next chapters we shall first discuss the application of Wardrop's principles (or extremal principles as they are called too) to a single source-sink network with capacity constraints on all arcs and constant arc costs. This will also lead to some insight on the relation between descriptive and normative assignment and result in the presentation of an algorithm to solve this problem. As a more realistic model we shall then discuss a multiple source-sink network with arc costs increasing with the flow, but no capacity constraints. This model can also be generalized to a model where the created flow in the source vertices depends on the travel costs, i.e. if travel costs are high, less people will travel to such a destination than if travel costs are low. Such models, also called trip distribution models, have the disadvantage that only heuristic algorithms are known which are not very satisfying, thus real world applications seem to be very limited. A presentation of such models can be found in Oliver & Potts (1972) and also in Florian et.al. (1975) and Florian (1976).

3.3.1. Network with constant arc costs

We shall first discuss the normative assignment. On this purpose we formulate our problem as a linear program in the following way:

In (3.4) we defined Kirchhoff's law in terms of flows through arcs. Another equivalent approach is to define the conservation equations in terms of flows through elementary paths, connecting the source with the terminal vertex. In this formulation of network flow, it is convenient to denote the arcs by $i=1,2,\ldots,l$; the arc flows by f_i, the source-sink elementary paths by m_j, $j=1,2,\ldots,m$ and the path flows by h_j. Then the flow value v is given by the conservation equation

$$v = \sum_j h_j \tag{3.7}$$

The arc flows f_j resulting from the path flows h_j can be obtained by letting

$$a_{ij} = \begin{cases} 1, & \text{if arc } i \text{ is on path } m_j, \\ 0, & \text{otherwise} \end{cases} \tag{3.8}$$

so that

$$f_i = \sum_j a_{ij}h_j. \tag{3.9}$$

We now can state the normative assignment as the linear program

$$\begin{aligned} & h_j \geq 0 && j=1,\ldots,m \\ & \sum_j h_j = v && \end{aligned} \tag{3.10}$$

$$f_i = \sum_j a_{ij} \leq q_i \qquad i=1,\ldots,l$$

$$\min \sum_j h_j c_j = C \tag{3.11}$$

where q_i is the capacity constraint on arc i
and c_j is the travel cost on path j (which is the sum of all costs of arcs belonging to j).

The objective (3.11) is exactly the formulation of Wardrop's second principle, if c_j represents the journey time on the jth path. In this case C is the total journey time.

From theory of linear programming it is well known that the dual program of (3.1o) and (3.11) (with dual variables ν and $-\mu_i$) can be written as

$$\nu - \sum_i a_{ij}\,\mu_i \leq c_j \qquad j=1,2,\ldots,m \tag{3.12}$$

$$\nu \qquad \text{unrestricted in sign} \tag{3.13}$$

$$\mu_i \geq 0 \qquad i = 1,2,\ldots,l \tag{3.14}$$

$$\max:\ \nu v - \sum_i \mu_i q_i = V \tag{3.15}$$

For optimal solutions h_j^* of the primal problem (with corresponding optimal arc flows f_i^*) and the optimal solutions ν^*, μ_i^* of the dual, the duality theory implies

$$C^* = V^* ,$$

that is

$$\sum_j h_j^*\, c_j = \nu^* v - \sum_i \mu_i^*\, q_i \tag{3.16}$$

as well as the complementary slackness inferences:

$$\text{if} \quad h_j^* > 0, \qquad \text{then} \quad \nu^* - \sum_i a_{ij}\,\mu_i^* = c_j \tag{3.17}$$

$$\text{if} \quad \nu^* - \sum_i a_{ij}\mu_i^* < c_{ij}, \ \text{then} \quad h_j^* = 0 \tag{3.18}$$

$$\text{if} \quad f_i^* = \sum_j a_{ij}\, h_j^* < q_i, \ \text{then} \quad \mu_i^* = 0 \tag{3.19}$$

$$\text{if} \quad \mu_i^* > 0, \qquad \text{then} \quad f_i^* = \sum_j a_{ij}h_j^* = q_i \tag{3.2o}$$

It is now possible to analyse the relation between normative assignement (as above stated) and descriptive assignment. On this purpose we introduce the following terminology. An arc is called saturated if $f_i = q_i$ and unsatured if $f_i < q_i$. For a given network flow, some or all of the flow on a particular (s to t) path can be diverted to another path, provided that all those arcs on the second path that do not belong to the first one are unsaturated. Any such path is said to be available for flow from the first path; otherwise, the path is said to be unavailable. A path may be available for flow from one path but unavailable for flow from another.

If we reformulate Wardrop's first principle in the following way

(i') The journey time (route cost) on all paths is less than or equal to the journey time on any path available for flow from it

We shall show now that a normative assignment, as stated in (3.1o) and (3.11), also is a descriptive assignment in its extended form of the first principle, as stated in (i'), but that there exist descriptive assignments that are not normative.

Suppose that path j has positive flow at the optimal solution of the normative assignment (i.e. $h_j^* > 0$). Then (3.17) implies

$$\nu^* - \sum_i a_{ij} \mu_i^* = c_j. \tag{3.21}$$

If k is a path available for flow from j, then the arcs of k, that do not belong to j, are unsaturated and hence by (3.19), the corresponding values of μ_i^* are zero, giving

$$\sum_i a_{ij} \mu_i^* \geq \sum_i a_{ik} \mu_i^* . \tag{3.22}$$

From (3.12), (3.21) and (3.22), it therefore follows that

$$c_j = \nu^* - \sum_i a_{ij}\mu_i^* \leq \nu^* - \sum_i a_{ik}\mu_i^* \leq c_k$$

as required for a descriptive assignment.

That the contrary (i.e. descriptive assignment is also normative) need not be true can be proven by a simple counterexample:

Example 1:
Given the network of Fig.3.4., with a flow value v=9 from vertex s to vertex t and arc capacities and costs given in Fig.3.5.

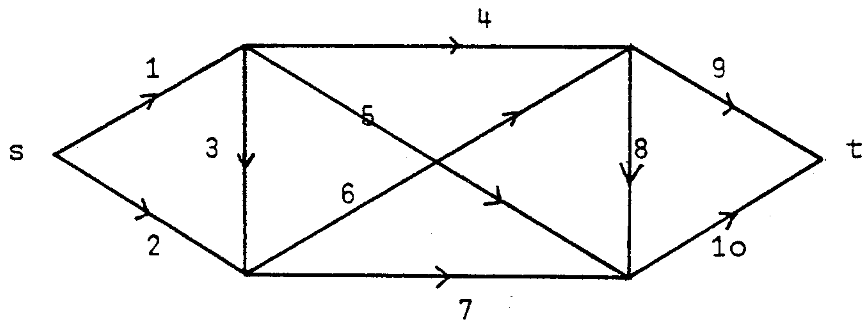


Fig.3.4.

arc number i	arc capacity q_i	arc cost σ_i
1	6	1
2	4	5
3	4	1
4	6	3
5	1	7
6	3	1
7	4	6
8	6	3
9	4	6
1o	6	3

Fig.3.5.

From Fig.3.4 it is clear that there exist 9 elementary paths m_j for which the route costs c_j can be computed as the sum of the arc costs σ_i over those arcs which belong to m_j. The problem can then be formulated like (3.1o) and (3.11) and solved with, for example, the simplex-method (we shall present a more efficient algorithm for this special linear program later). The optimal solution is given in Fig.3.6 Therefore the minimal total

path m_j	arcs i	route cost c_j	path flow h_j^*
m_1	1,3,6,8,1o	9	0
m_2	1,3,6,9	9	0
m_3	1,3,7,1o	11	0
m_4	1,4,8,1o	1o	4
m_5	1,4,9	1o	2
m_6	1,5,1o	11	0
m_7	2,6,8,1o	12	2
m_8	2,6,9	12	1
m_9	2,7,1o	14	0

arc flow f_i^*	arc number i
6	1
3	2
0	3
6	4
0	5
3	6
0	7
6	8
3	9
6	10

Fig.3.6

journey cost is $C^*=96$.

In Fig.3.7 a descriptive assignment is given that has a total cost of $C'=99$ and therefore is not a normative assignment.

path m_j	path flow h'_j
m_1	2
m_2	1
m_3	0
m_4	1
m_5	2
m_6	0
m_7	0
m_8	0
m_9	3

arc flow f'_i	arc number i
6	1
3	2
3	3
3	4
0	5
3	6
3	7
3	8
3	9
6	10

Fig.3.7.

To show that the path flow of Fig.3.7 is optimal to the individual drivers,we shall look at the available paths for those paths which have a nonzero flow.

paths m_j with flow	available paths m_k
1	2,3,4,5,6,7,8,9
2	5
4	3,5,6,9
5	-
9	-

It can now easily be checked in Fig.3.6. that all paths with flow do not have greater costs than the available paths for the particular flows. This completes the counter-example. Note however, that chains m_3, m_6, m_7, m_8 with costs less than the cost of chain m_9, have no flow (and m_9 has).

We will now consider the problem of finding a flow for a given value v from s to t so that the total cost of the flow is minimized. Although this problem could be solved with linear programming, as shown in (3.1o) and (3.11), this is not a very efficient way and the linear program formulation was given only for the theoretical considerations. Obviously, the minimum cost flow problem is only meaningful if the given flow value v is not greater than the maximum flow from s to t. The best known method for the minimum cost flow problem is the so-called "out-of-kilter" algorithm of Ford and Fulkerson. Here we will describe a method, due to M.Klein, which is conceptually simpler than the out-of-kilter method and use techniques already presented in this book. Computationally the methods are comparable. A more detailed description of the following can be found in Christofides (1975).

Let us suppose that a feasible flow f of value v exists in the graph and that this flow pattern is known. Such a flow pattern can be obtained by applying the (s to t) maximum flow algorithm and performing Steps 4 to 6 of this algorithm not until the maximum flow is reached, but until the flow f_{st} from s to t reaches the given flow value v. With this feasible flow define a so-called incremental network $G^{\mu}(f)=(X^{\mu}, A^{\mu})$ on the given network $G=(X,A)$ in the following way:

$$X^{\mu} = X$$
$$A^{\mu} = A_1^{\mu} \cup A_2^{\mu}$$

where

$$A_1^\mu = \{(x_i^\mu, x_j^\mu)/f_{ij} < q_{ij}\}$$

and the capacity of an arc $(x_i^\mu, x_j^\mu) \varepsilon A_1^\mu$ being

$$q_{ij}^\mu = q_{ij} - f_{ij}$$

and

$$A_2^\mu = \{(x_j^\mu, x_i^\mu)/f_{ij} > 0\}$$

with the capacity of an arc $(x_j^\mu, x_i^\mu) \varepsilon\ A_2^\mu$ being

$$q_{ij}^\mu = f_{ij} .$$

The arc costs are specified as

$$c_{ij}^\mu = c_{ij} \quad \text{for all arcs} \quad (x_i^\mu, x_j^\mu) \varepsilon A_1^\mu$$

$$c_{ji}^\mu = -c_{ij} \quad \text{for all arcs} \quad (x_j^\mu, x_i^\mu) \varepsilon A_2^\mu .$$

The graph $G^\mu(f)$ now represents incremental capacities and costs (relative to the flow pattern f) of any extra flow pattern to be introduced into G. The algorithm is then based on the following theorem:

Theorem 1:
f is a minimum cost flow value v if-and only if-there is no circuit Ø in $G^\mu(f)$,such that the sum of the costs of the arcs in Ø is negative.

We shall not be presenting the proof -the interested reader is referred to Christofides (1975). As a result of Theorem 1, the algorithm for the minimum cost flow problem reduces to building $G^\mu(f)$ and then finding out if there exists a circuit Ø in $G^\mu(f)$ with negative costs. This can be done with Floyd's algorithm to find the shortest paths between all pairs of vertices in a given graph.

Although we introduced this algorithm by assuming that all arc lengths (costs) $c_{ij} \geq 0$, this algorithm also works for c_{ij} unrestricted in sign. Going back to Step 3 of Floyd's algorithm, one has only to check if there exists an $c_{ii} < 0$ (where c_{ij} represents the minimum costs to reach vertex x_j from x_i using exactly k arcs), and if there is, then a negative cost circuit has been detected and its arcs can be found by using a recursive relation similar to (3.2.).

Minimum cost flow algorithm:

Step 1: Use the (s to t) maximum flow algorithm to find a feasible flow f of value v in the network G.

Step 2: Relative to flow f from the incremental network $G^{\mu}(f)$.

Step 3: Find a negative cost circuit $\emptyset$ in $G^{\mu}(f)$ with Floyd's algorithm.
If such a circuit exists, identify its arcs and go to Step 4.
If no such circuit can be found, Stop.

Step 4: Calculate d according to

$$d = \min_{(x_i^{\mu}, x_j^{\mu}) \in \emptyset} (q_{ij}^{\mu})$$

Send the maximum possible flow around the circuit such that the new flow pattern is still feasible in G (this is exactly d). The overall flow from s to t then remains unchanged at the value v, although its cost is reduced by $d.c(\emptyset)$, where $c(\emptyset)$ is the cost of the circuit $\emptyset$.

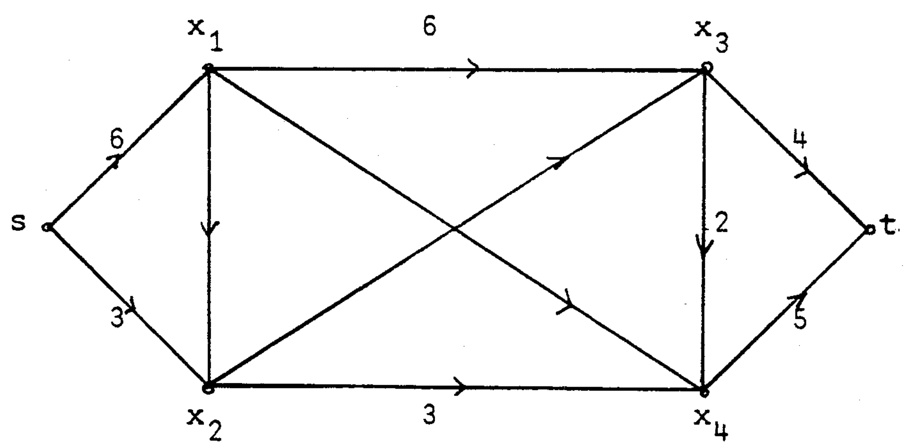
(i) For all (x_i^{μ}, x_j^{μ}) in $\emptyset$ with $c_{ij}^{\mu} < 0$ change the flow f_{ji} in the corresponding arc (x_j, x_i) of G from f_{ji} to $f_{ji} - d$.

(ii) For all (x_i^μ, x_j^μ) in Ø with $c_{ij}^\mu > 0$ change the flow f_{ij} in the corresponding arc (x_i, x_j) of G from f_{ij} to $f_{ij} + d$.

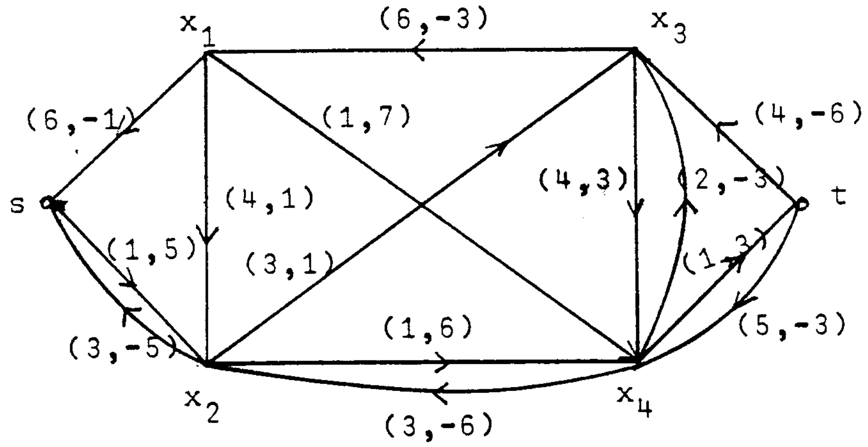
With this new flow pattern return to Step 2.

Example 2:
Using the data of Example 1 of this chapter, we shall now verify the optimal solution given in Fig.3.6 for the arc flows. For simplicity we shall perform Step 1 and 3 of the algorithm rather intuitively than with an algorithm.

Step 1:
Let us start with the following feasible flow pattern with flow value v=9 (the numbers of the arcs give the flow) and total flow costs C=1o2.



Step 2:
Compute the network $G^\mu(f)$ for the given flow pattern of Step 1. This results in the following network (first label is arc capacity, second label is arc cost):

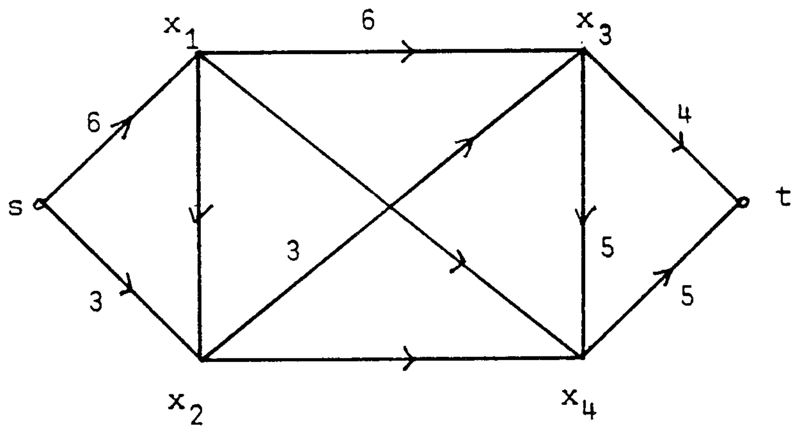


One possible circuit is then defined by the arcs (x_2,x_3), (x_3,x_4), (x_4,x_2) with total costs of $c(\emptyset)=1+3-6=-2$.

Step 4: We calculate

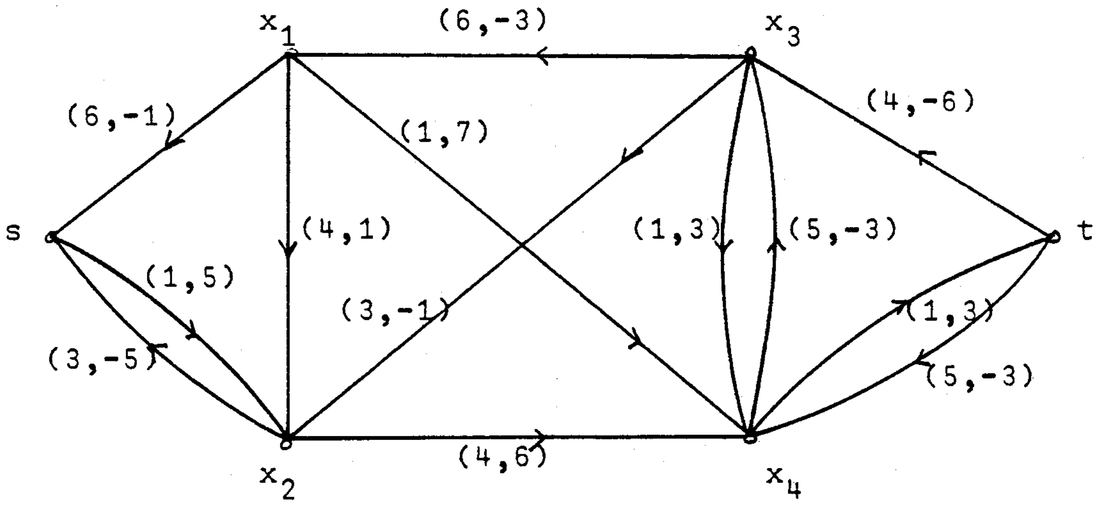
$$d = \min(3,4,3)=3$$

The new flow pattern is then



with flow costs C=96. Actually,from the optimal solution of Example 1 we know that C=96 already is the minimal total cost. Yet we have found another solution. but by detecting a circuit with zero costs, we can construct the optimal solution of Fig.3.6. Therefore we go back to

Step 2:

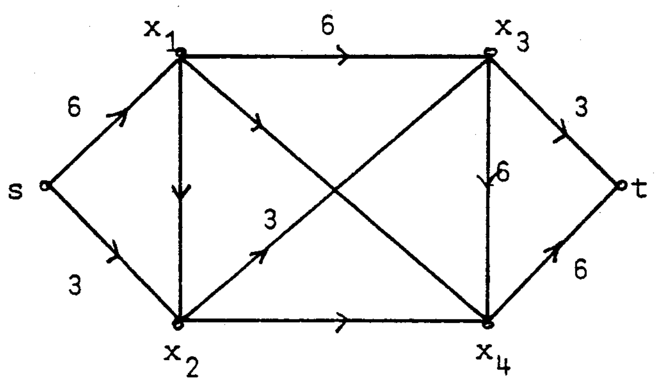


Now there exists no negative cost circle. Thus, an optimal solution has been found. But we can detect another optimal solution with the zero cost circle defined by the arcs (x_3,x_4), (x_4,t), (t,x_3).

Step 4:

$$d = \min (1,1,4) = 1$$

The new flow pattern is then



which is exactly the solution given in Fig.3.6.

```
C ... ***  PROGRAM FOR FINDING MINIMUM COST FLOW
C ... ***
C
C ... INPUT
C
C ... N      NUMBER OF VERTICES
C ... C      FLOW COST (ARC LENGTH)
C ... Q      CAPACITY ON ARCS
C ... V      WANTED FLOW
C ... S      ORIGIN VERTEX OF FLOW V
C ... T      DESTINATION VERTEX OF FLOW V
C
C ... OUTPUT
C
C ... F      MINIMUM COST FLOW
C ... C1     VALUE OF MINIMUM COST FLOW
C
       SUBROUTINE MINCOS(N,C,Q,V,S,T,F,C1)
       INTEGER N ,C(1),Q(1),V,S,T,F(1),C1,CU(2704),QU(2704)
       INTEGER D(2704),IG(2704)
       LOGICAL LOG
C
C ... STEP 1
C
       CALL MAXFLO(N,Q,F,S,T,V)
C
C ... STEP 2
C
5      CALL INCR(N,F,Q,C,QU,CU)
C
C ... STEP 3
C
       M=N*N
       DO 10 I=1,M
10     IG(I)=CU(I)
       CALL SPII(N,IG,D,LOG)
       IF(.NOT. LOG) GO TO 15
C
C ... STEP 4
C
       CALL NEWFLO(N,IG,D,QU,CU,F)
       GO TO 5
C
C ... PREPARATION OF OUTPUT
C
15     C1=0
       DO 20 I=1,M
20     C1=C1+F(I)*C(I)
       RETURN
       END
```

```
C ... *** INCREMENTAL NETWORK CONSTRUCTION RELATIVE TO FLOW F
C ... ***
C
C ... INPUT
C
C ... N     NUMBER OF VERTICES
C ... F(L)  FLOW ON ARC(I,J), WHERE L=(I-1)*N+J
C ... Q(L)  CAPACITY ON ARC(I,J)
C ... C(L)  FLOW COST (ARC LENGTH) ON ARC(I,J)
C
C ... OUTPUT
C
C ... QU(L) INCREMENTAL CAPACITY ON ARC(I,J)
C ... CU(L) INCREMENTAL FLOW COST ON ARC(I,J)
C
C
      SUBROUTINE INCR(N,F,Q,C,QU,CU)
      INTEGER N,F(1),Q(1),C(1),QU(1),CU(1)
C
      M=N*N
      DO 5 L=1,M
      QU(L)=0
5     CU(L)=0
      DO 15 L=1,M
      IF(F(L) .EQ. Q(L)) GO TO 10
      I=MIN0(0,C(L))
      IF(CU(L) .LT. I) GO TO 10
      QU(L)=Q(L)-F(L)
      CU(L)=C(L)
10    IF(F(L) .EQ. 0) GO TO 15
      I=(L-1)/N+1
      J=L-((I-1)*N)
      K=IND(J,I,N)
      I=MIN0(0,-C(L))
      IF(CU(K) .LT. I) GO TO 15
      QU(K)=F(L)
      CU(K)=-C(L)
15     CONTINUE
      RETURN
      END
```

```
C ... *** FINDING NEW FLOW WITH LESS COSTS
C ... ***
C
C ... INPUT
C
C ... N       NUMBER OF VERTICES
C ... IG      LENGTH OF SHORTEST PATHS IN INCREMENTAL NETWORK
C ... D       VECTOR FOR FINDING THE VERTICES BELONGING
C             TO THE SHORTEST PATHS
C ... QU      CAPACITIES IN INCREMENTAL NETWORK
C ... CU      COSTS IN INCREMENTAL NETWORK
C ... F       ACTUAL FLOW IN ORIGINAL NETWORK
C
C ... OUTPUT
C
C ... F       NEW IMPROVED FLOW IN ORIGINAL NETWORK
C
      SUBROUTINE NEWFLO(N,IG,D,QU,CU,F)
      INTEGER N,IG(1),D(1),QU(1),CU(1),F(1)
C
      M=2**18
      DO 5 I=1,N
      L=IND(I,I,N)
      IF(IG(L) .GE. 0) GO TO 5
      J=I
      GO TO 10
5     CONTINUE
10    I=J
15    K=J
      J=IND(I,K,N)
      J=D(J)
      L=IND(J,K,N)
      M=MIN0(M,QU(L))
      IF(J .NE. I) GO TO 15
20    K=J
      J=IND(I,K,N)
      J=D(J)
      L=IND(J,K,N)
      IF(CU(L) .LT. 0) GO TO 25
      F(L)=F(L)+M
      GO TO 30
25    L1=IND(K,J,N)
      F(L1)=F(L1)-M
30    IF(J .NE. I) GO TO 20
      RETURN
      END
```

3.3.2. Network with variable arc costs:

As this model represents so far a realistic and computable approach to descriptive and normative assignment, a lot of research work is going on in this field. A rather complete overview of the state of the art in the year 1974 is given in Florian (1976). A variety of algorithms already exists and one of the latest published is by Nguyen (1974). Some of them have been compared - see the paper by S.Nguyen in Florian (1976).

We shall generalize the model of chapter 3.3.1. in the sense that flows of a given quantitiy between all pairs of vertices are possible, the so called multicommodity flows, which is realistic, as there will be traffic flow not only between two cities but between all cities that are represented by vertices in the network. The trip matrix (g_{ij}) shall denote the flow density between vertex i (the source or origin) and vertex j (the sink or destination). Let us again number the vertices from 1 to n and the arcs from 1 to m; the first $1,2,\dots,n_o$ vertices are the origins. Let h_l denote the flow on the elementary path l (connecting an origin-destination pair i,j), and f_a^s the flow on arc a coming from origin s.

The total flow on arc a is therefore defined as

$$f_a = \sum_{s=1} f_a^s \qquad (3.23)$$

or as

$$f_a = \sum_{i,j} \sum_{l \in Q_{ij}} \delta_{al} h_l \qquad (3.24)$$

where Q_{ij} denotes the set of paths connecting the origin-destination (OD) pair ij, and δ_{al} equals 1 if arc a belongs to path l, and 0 otherwise. On the network, the relationships between the different flow variables are expressed by the flow conservation equations

$$\sum_{l \in Q_{ij}} h_l = g_{ij} \quad \text{for all OD-pairs ij} \qquad (3.25)$$

$$h_l \geq 0 , \quad \text{for all } l \in \bigcup_{ij} Q_{ij}$$

Note that (3.24) and (3.25) are very similar to the equations (3.9.) and (3.7.), respectively in case of only one OD-pair. (3.25) represents the vertex-path formulation of the problem. Similar to (3.4.) it can also be formulated as a vertex-arc problem

$$\sum_{a \in W_i} f_a^s - \sum_{a \in V_i} f_a^s = \begin{cases} -g_{si} & \text{if i is a destination vertex} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{for } i \neq s;, \ i=1,2,\ldots,n; \ s=1,2,\ldots,n_o \qquad (3.26)$$

$$f_a^s \geq 0 \quad \text{for } a=1,2,\ldots,m; \ s=1,2,\ldots,n_o$$

where W_i is the set of arcs beginning at i, and V_i the set of arcs ending at i. Naturally, in both formulations it is assumed that

$$g_{ij} \geq 0 \quad \text{for all pairs ij.}$$

As mentioned earlier, we are not considering arc capacity constraints. But it is worth mentioning that the model of chapter 3.3.1. can be extended in the above described way to a minimum cost-multicommodity flow problem. Unfortunately, such a model cannot be solved with any special network algorithm, rather the normal simplex-algorithm with some special features, as described in Hu (197o), has to be used.

Let now denote c_a the travel cost on an arc a. Then we are assuming that the travel costs depend on the flow in the sense that travel costs increase when the flow increases, i.e. we are considering traffic congestion with this model. That means

$$c_a = C_a(f_a)$$

where C_a is an increasing function of f_a. Then the objective function of the normative assignment problem can be stated as

$$\min: C = \sum f_a C_a(f_a) \qquad (3.27)$$

for all a=1,2,...,m

The problem is therefore to minimize (3.27) under the constraints (3.26). Because C_a is increasing, it can be shown that the objective function is convex, thus a unique global optimal solution exists.

Let now u_{ij} be the minimum travel cost to the user for the trip from vertex i to j. These costs are formed by the summation of the costs on the arcs of the path from i to j with minimum cost. Then Wardrop's first principle can be formulated as follows:

$$\text{If } h_l > 0 \quad \text{then} \quad \sum_a \delta_{al} C_a(f_a) = u_{ij} \tag{3.28}$$

$$\text{If } \sum_a \delta_{al} C_a(f_a) \geq u_{ij} \quad \text{then} \quad h_l = 0,$$

for all OD-pairs ij,

where summation is over all arcs a=1,2,...,m.

The meaning of (3.28) is,that if there is a positive flow h_l on a path $l \epsilon Q_{ij}$, then this must be a shortest path between OD-pair ij. On the other hand, if a path $l \epsilon Q_{ij}$ exists, such that travel cost on it is higher than on a shortest path, this path will not be used. Thus (3.28) gives Wardop's first principle.

Let us now define the minimization problem

$$\min \quad F = \sum_a \int_0^{f_a} C_a(x)dx \tag{3.29}$$

under the constraints (3.26). Again, (3.29) is a convex objective function as C_a is increasing, resulting in a unique global optimum. Then the Kuhn-Tucker conditions lead to a system of equations that are both necessary and sufficient for the optimum of the original problem (3.29) under the constraints (3.26). Without giving the proof here, it can be shown that the Kuhn-Tucker conditions of (3.29) and (3.26) lead to Wardrop's first principle, formulated in (3.28), as a necessary and sufficient condition of the optimum. This important result was first stated by M.Beckmann, C.McGuire and C.Winsten in 1956. A proof of this theorem is given in Steenbrink (1974). This result enables us to give the relations between descriptive and normative assignment. If we define travel costs on arcs as

$$\bar{c}_a = \bar{C}_a(f_a) = \frac{1}{f_a} \int_0^{f_a} C_a(x)dx \tag{3.3o}$$

then problem (3.29) can formally be written as

$$\min \quad F = \sum_a f_a \; \bar{C}_a(f_a) \tag{3.31}$$

which is exactly the objective of a normative assignment like the one stated in (3.27). Thus both assignments have the same solution, if

$$C_a(f_a) = \bar{C}_a(f_a) = \frac{1}{f_a} \int_0^{f_a} C_a(x)dx \tag{3.32}$$

which is true iff $C_a(x) \equiv C_a \equiv$ const. Therefore, in the case of constant travel costs, descriptive and normative assignments are equivalent.

Because of (3.31), the same solution methods apply to the descriptive and the normative assignment, both of them being complex nonlinear optimization problems. Besides heuristic methods (a reference to them can be found in Florian (1976)), only very recently effcient exact algorithms were developed, which can be found in Florian (1976) and Nguyen (1974).

For the simpler case of only one origin-destination pair of vertices, the algorithm, presented in chapter 3.3.1. for the minimum cost flow problem, can be used in a slightly adapted form. The solution then results in a normative assignment of the flow.

Step 1 of the algorithm remains unchanged. In Step 2 the incremental graph is built without computing the capacity constraints q^{μ}_{ij} of an arc (x^{μ}_i, x^{μ}_j), and the arc costs are now for an arc $(x^{\mu}_i, x^{\mu}_j) \varepsilon A^{\mu}_1$

$$c^{\mu}_{ij} = (f_{ij}+1).c_{ij}(f_{ij}+1)-c_{ij}(f_{ij}).f_{ij}$$

and

for an arc $(x^{\mu}_{j},x^{\mu}_{i})\varepsilon A^{\mu}_{2}$

$$c^{\mu}_{ji} = -(f_{ij}.c_{ij}(f_{ij}) - (f_{ij}-1).c_{ij}(f_{ij}-1)),$$

where $c_{ij}(f_{ij})$ represents the cost on arc (x_i,x_j), when the flow is f_{ij}.
Step 3 of the algorithm of chapter 3.3.1. remains unchanged and in Step 4 the additional flow d, to be sent around the circle is always set to

$$d=1.$$

This algorithm works in both cases, namely arcs with capacity constraints and arcs without capacity constraints. In the latter case, q_{ij} is set to the total flow v from the origin to the destination vertex. In the following program it is assumed that capacity-constraints do not exist, but it is easy to include them.

```
C ... ***  PROGRAM FOR FINDING MINIMUM COST FLOW
C ... *** WITH COSTS DEPENDING ON THE FLOW
C ... ***
C
C ... INPUT
C
C ... N      NUMBER OF VERTICES
C ... C      FLOW COST (ARC LENGTH)
C ... A    NUMBER OF ARCS
C ... MA   MAXIMUM NUMBER OF COEFFICIENTS PER COST FLOW
C ... V      WANTED FLOW
C ... S      ORIGIN VERTEX OF FLOW V
C ... T      DESTINATION VERTEX OF FLOW V
C
C ... OUTPUT
C
C ... F      MINIMUM COST FLOW
C ... C1     VALUE OF MINIMUM COST FLOW
C
       SUBROUTINE VARCOS(N,C,A,MA,V,S,T,F,C1)
       INTEGER N ,A,MA,V,S,T,F(1),C1,CU(400)
       INTEGER D(400),IG(400),Q(400)
       REAL C(1)
       LOGICAL LOG
C
C ... STEP 1
C
       M=N*N
       DO 25 I=1,M
25     Q(I)=0
       DO 30 I=1,A
       J=(I-1)*(MA+1)+1
       J=C(J)
30     Q(J)=V
       CALL MAXFLO(N,Q,F,S,T,V)
C
C ... STEP 2
C
5      CALL INCRVC(N,F,A,MA,C,CU,V)
C
C ... STEP 3
C
       M=N*N
       DO 10 I=1,M
10     IG(I)=CU(I)
       CALL SPII(N,IG,D,LOG)
       IF(.NOT. LOG) GO TO 15
C
C ... STEP 4
C
       CALL NEFLVC(N,IG,D,CU,F)
       GO TO 5
C
C ... PREPARATION OF OUTPUT
C
15     C1=0
```

```
      DO 20 I=1,M
      CL=CCC(N,A,MA,C,F,I)
20    C1=C1+F(I)*CL
      RETURN
      END



C ... *** INCREMENTAL NETWORK CONSTRUCTION RELATIVE TO FLOW F
C ... *** FOR MINIMUM COST FLOW ALGORITHM WITH VARIABLE
C ... *** ARC COSTS
C ... ***
C
C ... INPUT
C
C ... N     NUMBER OF VERTICES
C ... F(L)  FLOW ON ARC(I,J), WHERE L=(I-1)*N+J
C ... A     NUMBER OF ARCS
C ... MA    MAXIMUM NUMBER OF COEFFICIENTS PER ARC COST
C ... C     FLOW COST COEFFICIENTS
C ... V     TOTAL FLOW VALUE
C
C ... OUTPUT
C
C ... CU(L) INCREMENTAL FLOW COST ON ARC(I,J)
C
C
      SUBROUTINE INCRVC(N,F,A,MA,C,CU,V)
      INTEGER N,F(1),A,MA,CU(1),V
      REAL C(1)
C
      M=N*N
      DO 5 L=1,M
5     CU(L)=0
      DO 15 L=1,M
      IF(F(L) .EQ. V) GO TO 10
      IF(CU(L) .LT. 0) GO TO 10
      CL=CCC(N,A,MA,C,F,L)
      CU(L)=-CL*F(L)
      F(L)=F(L)+1
      CL=CCC(N,A,MA,C,F,L)
      CU(L)=CU(L)+CL*F(L)
      F(L)=F(L)-1
10    IF(F(L) .EQ. 0) GO TO 15
      I=(L-1)/N+1
      J=L-((I-1)*N)
      K=IND(J,I,N)
      CL=CCC(N,A,MA,C,F,L)
      CU(K)=-CL*F(L)
      F(L)=F(L)-1
      CL=CCC(N,A,MA,C,F,L)
      CU(K)=CU(K)+CL*F(L)
      F(L)=F(L)+1
15     CONTINUE
      RETURN
      END
```

```
C ... *** FINDING NEW FLOW WITH LESS COSTS
C ... *** WHEN ARC COSTS DEPEND ON FLOW
C ... ***
C
C ... INPUT
C
C ... N      NUMBER OF VERTICES
C ... IG     LENGTH OF SHORTEST PATHS IN INCREMENTAL NETWORK
C ... D      VECTOR FOR FINDING THE VERTICES BELONGING
C            TO THE SHORTEST PATHS
C ... CU     COSTS IN INCREMENTAL NETWORK
C ... F      ACTUAL FLOW IN ORIGINAL NETWORK
C
C ... OUTPUT
C
C ... F      NEW IMPROVED FLOW IN ORIGINAL NETWORK
C
      SUBROUTINE NEFLVC(N,IG,D,CU,F)
      INTEGER N,IG(1),D(1),CU(1),F(1)
C
      DO 5 I=1,N
      L=IND(I,I,N)
      IF(IG(L) .GE. 0) GO TO 5
      J=I
      GO TO 10
5     CONTINUE
10    I=J
20    K=J
      J=IND(I,K,N)
      J=D(J)
      L=IND(J,K,N)
      IF(CU(L) .LT. 0) GO TO 25
      F(L)=F(L)+1
      GO TO 30
25    L1=IND(K,J,N)
      F(L1)=F(L1)-1
30    IF(J .NE. I) GO TO 20
      RETURN
      END
```

```
C ... *** COMPUTATION OF VARIABLE COST
C ... ***
C
C ...
C ... INPUT
C
C ... N      NUMBER OF VERTICES
C ... A      NUMBER OF ARCS
C ... M      NUMBER OF COEFFICIENTS PER ARC COST
C ... C      COEFFICIENTS OF ARC COSTS
C ... F      FLOWS ON ARCS
C ... L      INDEX OF ARC FOR WHICH THE FLOW COSTS ARE COMPUTED
C
C ... OUTPUT
C
C ... CCC    FLOW COST ON ARC L WITH FLOW F(L)
C
      FUNCTION CCC(N,A,M,C,F,L)
      INTEGER N,A,M,F(1),L
      REAL C(1)
      CCC=0
      DO 5 I=1,A
      J=(I-1)*(M+1)+1
      J1=C(J)
      IF(L .NE. J1) GO TO 5
      GO TO 10
5     CONTINUE
      RETURN
10    DO 15 I=1,M
      J=J+1
      IF(F(L).EQ.0 .AND. I.EQ.1) GO TO 25
      CCC=CCC+C(J)*F(L)**(I-1.)
      GO TO 20
25    CCC=CCC+C(J)
20    CONTINUE
15    CONTINUE
      RETURN
      END
```

For the general case of multiple origin-destination pairs of vertices, analytical methods are mostly based on non-linear optimization methods (i.e. feasible direction methods). As we do not want to go into this theory,we present a different approach for solving the normative assignment problem which is based on an extension of the above presented algorithm for convex costs and only one origin-destination pair of vertices. Of course, also the descriptive assignment problem can be solved by this algorithm if the costs are transformed according to (3.3o).

The idea of the following algorithm is to improve the flow between one origin-destination pair,while the other flows remain unchanged. This is performed for all origin-destination pairs until no improvement can be found for any pair. In order to reduce computation-time,a different approach than before is used to find a feasible initial flow. This is done by assigning a suitable fraction of the total flow to the shortest paths, then recomputing the arc costs and,again,assigning flow to the now shortest paths. This procedure is repeated until all flow is assigned.

Algorithm for the normative traffic assignment problem:

Step 1:

Find the cheapest routes between all O-D pairs with costs $c_{ij}(0)$ with Floyd's algorithm.

Step 2:

Take some suitable fraction of the total flow and assign it all to these shortest routes. The suitable fraction (perhaps 1o%) should be chosen,so that this assignment will not already create congestion on certain links and, hopefully, will not cause very large changes in the costs $c_{ij}(h_{ij})$.

Step 3:

Recalculate the $c_{ij}(h_{ij})$ using the flows assigned in the last step. Recalculate the shortest routes. One may find now that the optimal routes have changed because congestion on the old routes has made new routes cheaper. Now take a new fraction of the total flow and assign it to these optimal routes.

Step 4:

Repeat Step 3 until all the flow has been assigned to some routes.

Step 5:

For each O-D pair improve the flow while leaving the other flows unchanged, until no more improvement can be found. Use the above mentioned algorithm for convex arc costs but with the following incremental graph:

Let f_{ij}^{st} be the flow on arc (i,j) going from origin vertex s to destination vertex t. Let h_{ij} be the total flow on arc (i,j) thus

$$h_{ij} = \sum_{\substack{s \varepsilon X \\ t \varepsilon X}} f_{ij}^{st} \qquad ,$$

where G = (X,A) is the given network. Let (g_{st}) be the trip matrix. The incremental graph $G^{\mu} = (X^{\mu}, A^{\mu})$ relative to flow f_{ij}^{st} from s to t and to the total flow h_{ij}, is given as

$$X^{\mu} = X$$

$$A^{\mu} = A_1^{\mu} \cup A_2^{\mu}$$

where

$$A_1^{\mu} = \{(x_i^{\mu}, x_j^{\mu}) | \; f_{ij}^{st} < g_{st}\}$$

and

$$A_2^\mu = \{(x_j^\mu , x_i^\mu) | f_{ij}^{st} > 0\} .$$

The arc costs are specified as

$$c_{ij}^\mu = (h_{ij}+1)c_{ij}(h_{ij}+1)-h_{ij} c_{ij}(h_{ij})$$

$$\text{for all arcs } (x_i^\mu, x_j^\mu) \epsilon A_1^\mu$$

Use G^μ for finding a flow f_{ij}^{st} from s to t with less cost.

```
C ... *** TRAFFIC ASSIGNMENT PROGRAM
C ... ***
C
C ... INPUT
C
C ... N      NUMBER OF VERTICES
C ... C      COEFFICIENTS OF ARC COST POLYNOMIAL
C ... A      NUMBER OF ARCS
C ... OD     ORIGIN-DESTINATION MATRIX
C ... MA     MAXIMUM NUMBER OF COEFFICIENTS PER COST FLOW
C
C ... OUTPUT
C
C ... F(I,J,K,L)   AMOUNT OF FLOW FROM VERTEX K TO L
C                  ON ARC(I,J)
C ... C1           TOTAL FLOW COST (TRANSPORTATION TIME)
C ... FL           TOTAL FLOW ON AN ARC
C
      SUBROUTINE TRAFAS(N,C,A,OD,MA,C1,F,FL)
      INTEGER OD(1),N,A,MA,C1,F(14,14,14,14)
      REAL C(1)
      INTEGER E(400),D(400),S,T,FL(1)
      LOGICAL LOG,ICA
      DO 5 I=1,N
      DO 10 J=1,N
      DO 15 K=1,N
      DO 20 L=1,N
20    F(I,J,K,L)=0
15    CONTINUE
10    CONTINUE
5     CONTINUE
C
C ... STEP 3
C
      DO 25 I=1,10
C
C ... STEP 1
C
      CALL COST(N,A,MA,C,F,E,FL)
      CALL SPII(N,E,D,LOG)
C
C ... STEP 2
C
25    CALL ASS(N,D,F,OD,I)
      KK=0
      NN=0
      DO 30 I=1,N
      DO 35 J=1,N
      DO 40 K=1,N
      DO 45 L=1,N
      IF(F(I,J,K,L) .EQ. 0) GO TO 45
      KK=KK+F(I,J,K,L)
      NN=NN+1
45    CONTINUE
40    CONTINUE
35    CONTINUE
```

```
30      CONTINUE
        KK=KK/NN
        KK=MAX0(2,KK)
50      KK=KK/2.
        ICA=.TRUE.
        KK=MAX0(KK,1)
        MM=2**17
        M=N*N
55      MN=0
        DO 60 L=1,M
        IF(OD(L) .GE. MM .OR. MN .GE. OD(L)) GO TO 60
        MN=OD(L)
        ML=L
60      CONTINUE
        IF(MN .EQ. 0 .AND. KK .GT. 1) GO TO 50
        IF(MN.EQ.0 .AND.(KK.EQ.1 .AND. ICA)) GO TO 70
        IF(MN .EQ. 0) GO TO 50
        MM=MN
        S=(ML-1)/N+1
        T=ML-((S-1)*N)
        CALL NETRAF(N,C,A,MA,MM,S,T,F,KK,FL,ICA)
        GO TO 55
70      CALL COST(N,A,MA,C,F,E,FL)
        C1=0
        DO 75 I=1,M
75      C1=C1+E(I)*FL(I)
        RETURN
        END
```

```
C ... *** COMPUTING ARC COSTS FOR GIVEN FLOW
C ... ***
C
C ... INPUT
C
C ... N         NUMBER OF VERTICES
C ... A         NUMBER OF ARCS
C ... MA        NUMBER OF COEFFICIENTS PER ARC COST
C ... C         COEFFICIENTS OF ARC COST POLYNOMIAL
C ... F         FLOW
C
C ... OUTPUT
C
C ... E         ARC COSTS
C ... FL        TOTAL FLOW ON AN ARC
C
      SUBROUTINE COST(N,A,MA,C,F,E,FL)
      INTEGER N,A,MA,F(14,14,14,14),E(1),FL(1)
      REAL C(1)
      M=N*N
      DO 5 I=1,M
5     FL(I)=0
      DO 10 I=1,N
      DO 15 J=1,N
      L=IND(I,J,N)
      DO 20 K=1,N
      DO 25 K1=1,N
25    FL(L)=FL(L)+F(I,J,K,K1)
20    CONTINUE
15    CONTINUE
10    CONTINUE
      DO 30 I=1,N
      DO 35 J=1,N
      L=IND(I,J,N)
35    E(L)=CCC(N,A,MA,C,FL,L)
30    CONTINUE
      RETURN
      END
```

```
C ... *** ASSIGNING 10% OF THE O-D FLOWS TO THE SHORTEST PATHS
C ... ***
C
C ... INPUT
C
C ... N        NUMBER OF VERTICES
C ... D        VECTOR DENOTING THE SHORTEST PATHS
C ... F        OLD FLOW
C ... OD       ORIGIN-DESTINATION MATRIX
C ... IA       NUMBER OF CALL
C
C ... OUTPUT
C
C ... F        NEW FLOW
C
       SUBROUTINE ASS(N,D,F,OD,IA)
       INTEGER N,D(1),F(14,14,14,14),OD(1),IA
       DO 5 I=1,N
       DO 10 J=1,N
       K=J
       K1=IND(I,J,N)
       IF(IA .LT.10) GO TO 30
       K2=0
       DO 25 L=1,N
       K2=K2+F(I,L,I,J)
25     CONTINUE
       K2=OD(K1)-K2
       GOTO 15
30     K2=OD(K1)/10.
15     L=K
       K=IND(I,L,N)
       K=D(K)
       F(K,L,I,J)=F(K,L,I,J)+K2
       IF(K .NE. I) GO TO 15
10     CONTINUE
5      CONTINUE
       RETURN
       END
```

```
C ... *** FINDING NORMATIVE IMPROVED FLOW
C ... ***
C
C ... INPUT
C
C ... N        NUMBER OF VERTICES
C ... C        ARC COST COEFFICIENTS
C ... A        NUMBER OF ARCS
C ... MA       NUMBER OF COEFFICIENTS PER ARC
C ... MM       AMOUNT OF FLOW FROM S TO T
C ... S        FLOW ORIGIN VERTEX
C ... T        FLOW DESTINATION VERTEX
C ... F        OLD FLOW
C ... KK       ALLOWED CHANGE OF FLOW
C ... FL       TOTAL FLOW ON AN ARC
C ... F        OLD FLOW BETWEEN S AND T ON AN ARC
C
C ... OUTPUT
C
C ... F        NEW NORMATIVE IMPROVED FLOW BETWEEN S AND T
C              OF VALUE MM
C ... ICA      IF ICA=.FALSE. THEN NEW AND OLD FLOW DIFFER
C
      SUBROUTINE NETRAF(N,C,A,MA,MM,S,T,F,KK,FL,ICA)
      INTEGER N,A,MA,MM,S,T,F(14,14,14,14)
      INTEGER FL(1),CU(400),D(400),IG(400),KK
      LOGICAL ICA
      REAL C(1)
      LOGICAL LOG
5     CALL INCRTA(N,F,A,MA,C,CU,MM,FL,S,T,KK)
      M=N*N
      DO 10 I=1,M
10    IG(I)=CU(I)
      CALL SPII(N,IG,D,LOG)
      IF( .NOT. LOG) RETURN
      CALL NEFLTA(N,IG,D,CU,F,S,T,KK)
      ICA=.FALSE.
      GO TO 5
      END
```

```
C ... *** INCREMENTAL NETWORK CONSTRUCTION RELATIVE TO FLOW F
C ... *** AND FL FOR TRAFFIC ASSIGNMENT ALGORITHM
C ... ***
C
C ... INPUT
C
C ... N     NUMBER OF VERTICES
C ... F(L)  FLOW ON ARC(I,J) OF FLOW FROM K TO L
C ... A     NUMBER OF ARCS
C ... MA    MAXIMUM NUMBER OF COEFFICIENTS PER ARC COST
C ... C     FLOW COST COEFFICIENTS
C ... V     TOTAL FLOW VALUE FROM S TO T
C ... FL    TOTAL FLOW ON ARC(I,J)
C ... S     FLOW ORIGIN VERTEX
C ... T     FLOW DESTINATION VERTEX
C ... KK    ALLOWED FLOW CHANGE ON ONE ARC
C
C ... OUTPUT
C
C ... CU(L) INCREMENTAL FLOW COST ON ARC(I,J)
C
C
      SUBROUTINE INCRTA(N,F,A,MA,C,CU,V,FL,S,T,KK)
      INTEGER N,FL(1),A,MA,CU(1),V,S,T,KK,F(14,14,14,14)
      INTEGER VV
      REAL C(1)
      CALL COST(N,A,MA,C,F,CU,FL)
      VV=V-KK
      M=N*N
      DO 5 L=1,M
5     CU(L)=0
      DO 15 L=1,M
      I=(L-1)/N+1
      J=L-((I-1)*N)
      IF(F(I,J,S,T) .GT.VV) GO TO 10
      IF(CU(L) .LT. 0) GO TO 10
      CL=CCC(N,A,MA,C,FL,L)
      CU(L)=-CL*FL(L)
      FL(L)=FL(L)+KK
      CL=CCC(N,A,MA,C,FL,L)
      CU(L)=CU(L)+CL*FL(L)
      FL(L)=FL(L)-KK
10    IF(F(I,J,S,T) .LT. KK) GO TO 15
      K=IND(J,I,N)
      CL=CCC(N,A,MA,C,FL,L)
      CU(K)=-CL*FL(L)
      FL(L)=FL(L)-KK
      CL=CCC(N,A,MA,C,FL,L)
      CU(K)=CU(K)+CL*FL(L)
      FL(L)=FL(L)+KK
15     CONTINUE
      RETURN
      END
```

```
C ... *** FINDING NEW FLOW WITH LESS COSTS
C ... *** FOR TRAFFIC ASSIGNMENT
C ... ***
C
C ... INPUT
C
C ... N      NUMBER OF VERTICES
C ... IG     LENGTH OF SHORTEST PATHS IN INCREMENTAL NETWORK
C ... D      VECTOR FOR FINDING THE VERTICES BELONGING
C            TO THE SHORTEST PATHS
C ... CU     COSTS IN INCREMENTAL NETWORK
C ... F      ACTUAL FLOW IN ORIGINAL NETWORK
C ... S      FLOW ORIGIN VERTEX
C ... T      FLOW DESTINATION VERTEX
C ... KK     FLOW CHANGE PER ARC
C
C ... OUTPUT
C
C ... F      NEW IMPROVED FLOW IN ORIGINAL NETWORK
C
      SUBROUTINE NEFLTA(N,IG,D,CU,F,S,T,KK)
      INTEGER N,IG(1),D(1),CU(1),F(14,14,14,14)
      INTEGER S,T,KK
C
      DO 5 I=1,N
      L=IND(I,I,N)
      IF(IG(L) .GE. 0) GO TO 5
      J=I
      GO TO 10
5     CONTINUE
10    I=J
20    K=J
      J=IND(I,K,N)
      J=D(J)
      L=IND(J,K,N)
      IF(CU(L) .LT. 0) GO TO 25
      F(J,K,S,T)=F(J,K,S,T)+KK
      GO TO 30
25    F(K,J,S,T)=F(K,J,S,T)-KK
30    IF(J .NE. I) GO TO 20
      RETURN
      END
```

If $G=(X,A)$ consists of n vertices, the minimum cost flow algorithm must be applied at least n^2 times (if there is a nonzero flow between all pairs of vertices), but usually will take kn^2, where k is some integer number. But as in each n^2 applications of the minimum cost flow algorithm, the total costs either decrease or remain the same (in this case the algorithm stops), the optimum will be found after a finite number k of iterations.

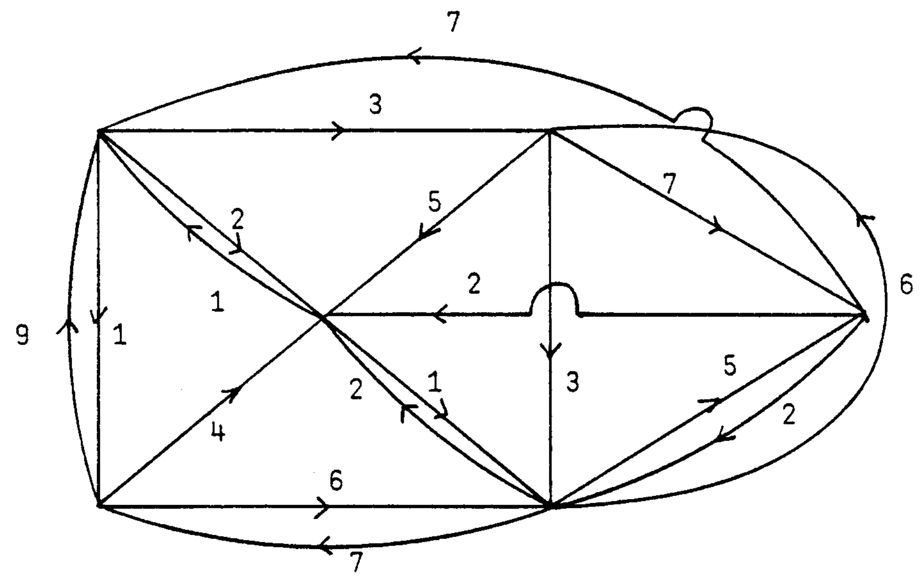
3.4. Exercises:

1) Find the shortest path from vertex A to vertex B with Dijkstra's algorithm for the following directed network given in matrix formulation:

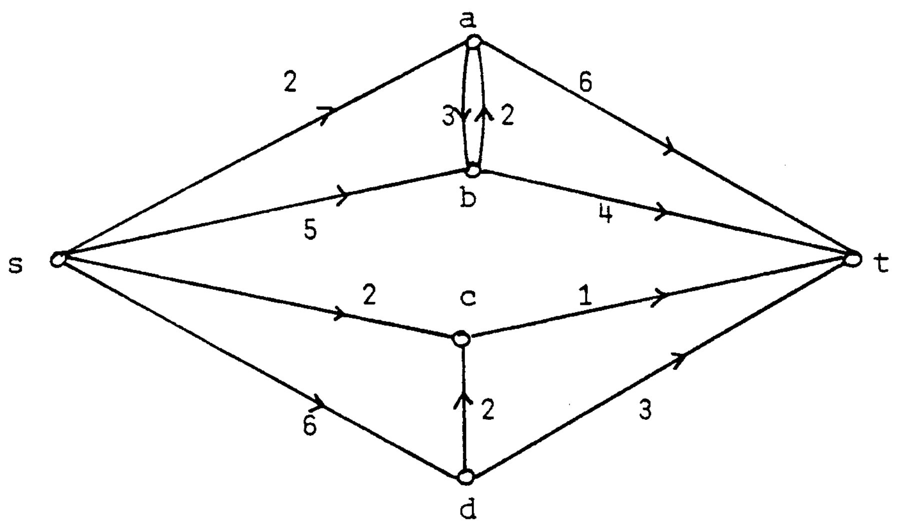
	A	x_1	x_2	x_3	x_4	x_5	B
A	0	3	0	2	5	0	0
x_1	0	0	3	1	3	4	0
x_2	0	0	0	0	0	0	3
x_3	0	0	4	0	2	5	0
x_4	0	0	1	0	0	0	5
x_5	0	0	0	0	0	0	2
B	0	0	0	0	0	0	0

If the number in the matrix is zero, then no arc exists between x_j and x_i. If the number is positive, then an arc exists and the number denotes the length (cost, travel time) of the arc.

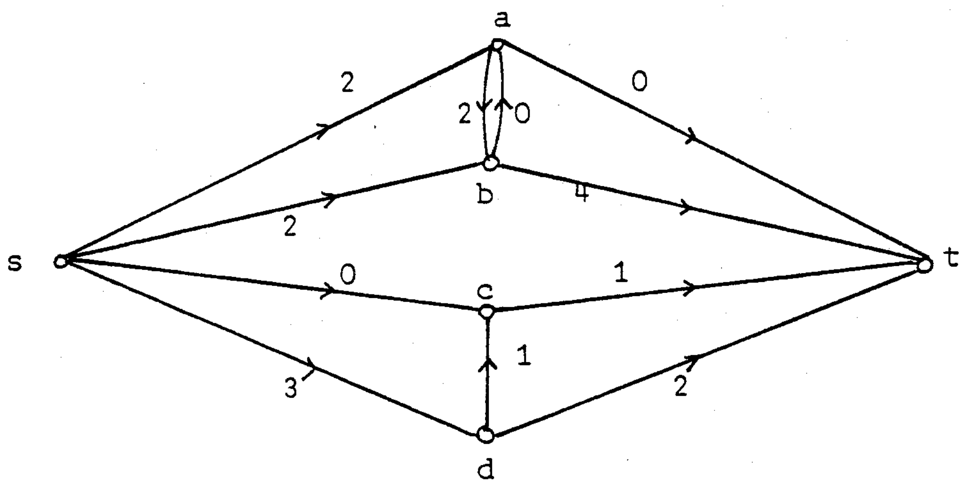
2) Find the shortest path between all pairs of vertices with Floyd's algorithm for the following network (the numbers denote the length of the arcs).



3) Given the following network (the numbers on the arcs denote the capacity):

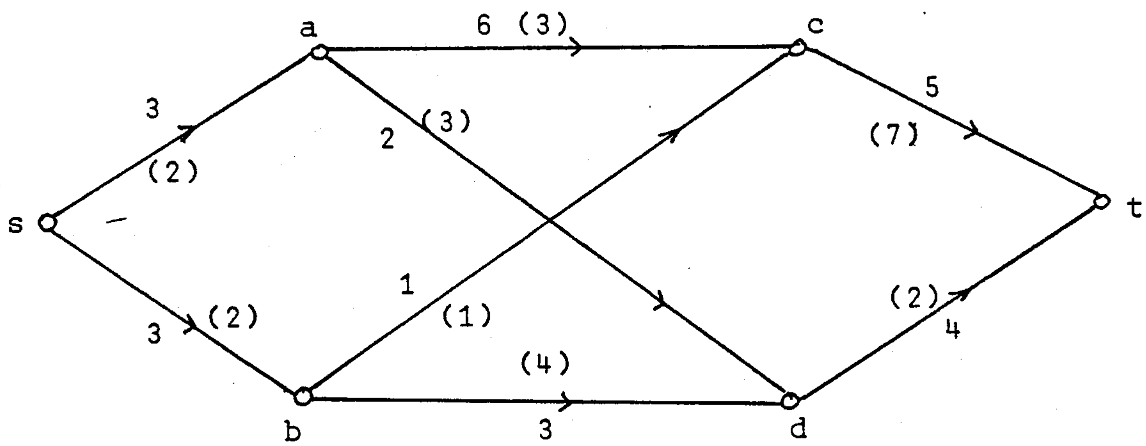


Let the existing flow be (the numbers on the arcs denote the flow, note that the sum of all flows going into vertices a,b,c, and d equals the sum of all flows going out of these vertices):



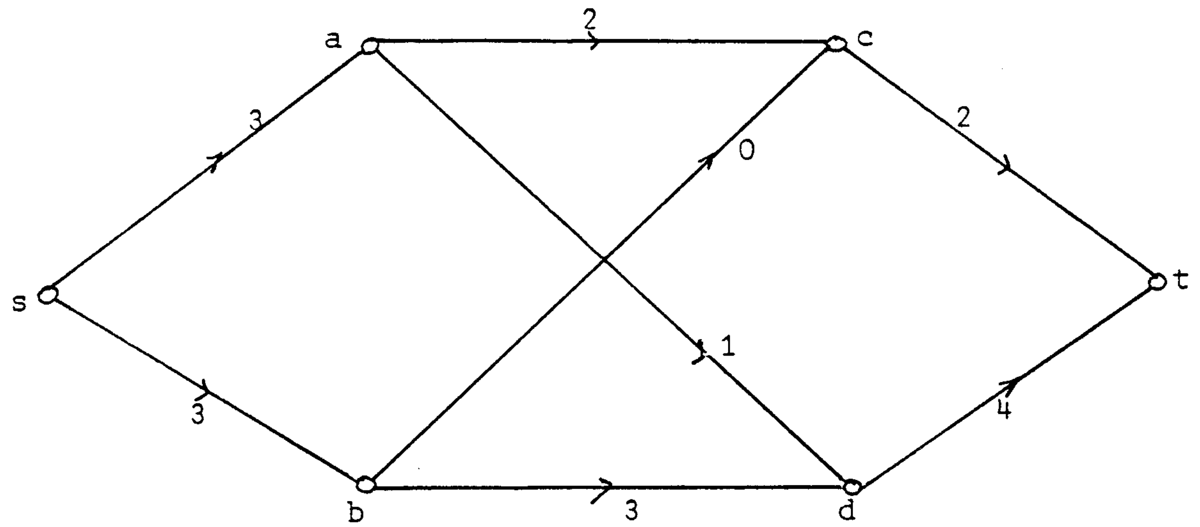
Find the maximum flow between s and t with the maximum flow algorithm.

4) Given the following road network (the numbers on the arcs denote the capacity "thousand cars per hour" and, the numbers in brackets on the arcs denote the travel time on this arc):

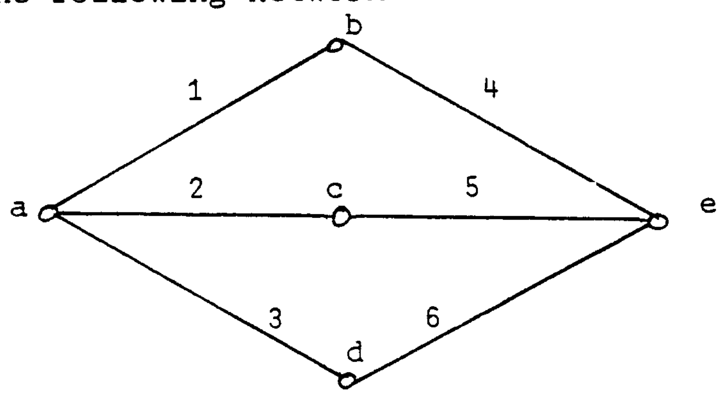


Let the flow from s to t be v=6 (thousand cars per hour). Formulate the normative traffic assignment problem as a linear programming problem and solve it.

Is the following traffic flow descriptive, normative or neither of both (the numbers on arcs denotes the flow)?:



5) Given the following network



with arc costs C_i depending on the arc flow f_i such that

$C_1 = 15 + 2f_1$ $\qquad$ $C_4 = 5 + 3f_4$

$C_2 = 5 + 9f_2$ $\qquad$ $C_5 = 10 + f_5$

$C_3 = 1 + 10f_3$ $\qquad$ $C_6 = 4 + 20f_6$

Let the flow value from a to e be 6 units. Find the descriptive and the normative assignment.

4. Choosing an optimal subnetwork

In the preceding chapter 3 we have been discussing real world planning problems that can arise on given networks, assuming special "behaviour" of the flow on such a network. Although the special problems we discussed were devoted to traffic theory, there are great similarities between, say, traffic flow in a road network and waste water flow in a canal system or information flow in a telephone network. Chapter 3 was preliminary in the sense that we are less interested in optimization on networks, but in optimization of networks. But, as we shall see later, in order to optimize a network, the way such a network is used (by flows) is part of the problem. In this chapter we will therefore be dealing with finding optimal subnetworks in various fields of public planning, i.e. waste water canal system, emergency service facilities, airline network planning as well as rail - and road - network planning.

4.1. Regional waste water management system

In the last decade the development of waste water management systems has become an important part of the efforts undertaken in industrialized countries, to keep the ecological damages under control. There seems to be a great tendency to build such waste water management systems rather on a regional level than by individual villages. The reasons for this are on the one side that the topographical situation and rivers can be better included and used within the system, resulting in reduced costs, on the other side, the marginal costs for building and running a waste water filter plant are decreasing for larger plants. For example, in the paper of Ahrens (1973) these costs c, depending on the amount of water x that could be purified, were measured by

$$c = 4\,500 \cdot x^{0.46}$$

units of money. In a paper by Polymeris (1977) it is mentioned that the cost of a filter-plant for 100.000 people is only six times the one for 10.000 people.

A regional waste water management system can be characterized by its villages, each of them producing an amount of $h_i \geq 0$ waste water, say per year, where i stands for some village represented by vertex i of a network. The number of villages plus the number of additional possible locations for filter plants plus the number of intersections of waste water canals give the total number of vertices in the network.

To each filter plant there is assigned a number x_i, representing the amount of water purified by the filter plant. Of course, x_i is not known but has to be computed for an optimal solution. Filter plants and villages are now connected by topographical possible canals, which are represented by arcs and to each arc (i,j), connecting vertex i and j, there is assigned a number f_{ij} indicating the amount of waste water that flows through this canal. To each f_{ij} and x_i costs $b_{ij}(f_{ij})$ and $a_i(x_i)$ are assigned that represent the costs of running a canal or a filter plant over a year, where the building costs are included (this can only be done if a certain planning horizon is defined). Then the optimization problem on a network G=(X,A) is given as

$$\sum_{\substack{k \\ (k,i)\in A}} f_{ki} + h_i = \sum_{\substack{j \\ (i,j)\in A}} f_{ij} + x_i \qquad \text{for all } i \in X \qquad (4.1)$$

(conservation equation)

$$f_{ij} \geq 0 \qquad \text{for all } (i,j)\varepsilon\, A$$

$$x_i \geq 0 \qquad \text{for all } i\varepsilon X$$

$$\min C = \sum_{i\varepsilon X} a_i(x_i) + \sum_{(i,j)\varepsilon A} b_{ij}(f_{ij}) \qquad (4.2)$$

(4.1) simply is Kirchhoff's law, as given in (3.4), but where each vertex can be a source or a sink vertex or both. Fortunately it is not a multicommodity flow problem as it does not matter to which filter plant the waste water flows. In (4.1) f_{ij} gives the capacity with which canal (ij) must be built (f_{ij}=0 means that the canal is not built at all) and x_i gives the size of the filter plant located at vertex i (x_i=0 indicates that this filter plant need not be built). As the marginal costs of canals as well as of filter plants are decreasing for growing size, this means the objective function (4.2) is concave, like the function in Fig.4.1.

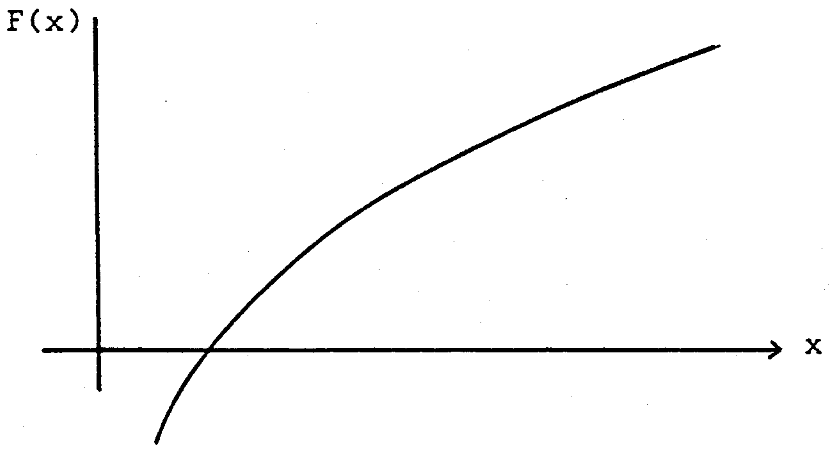
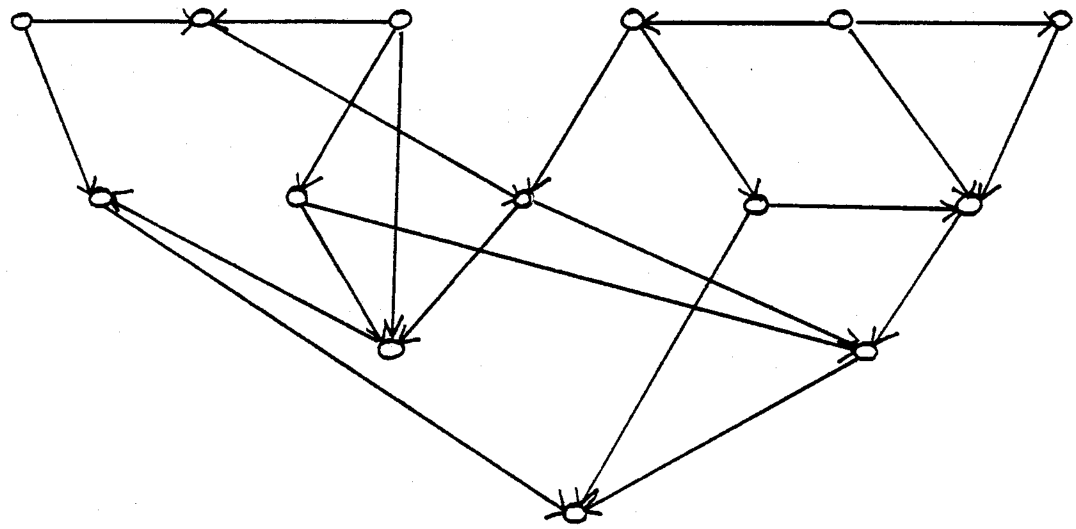


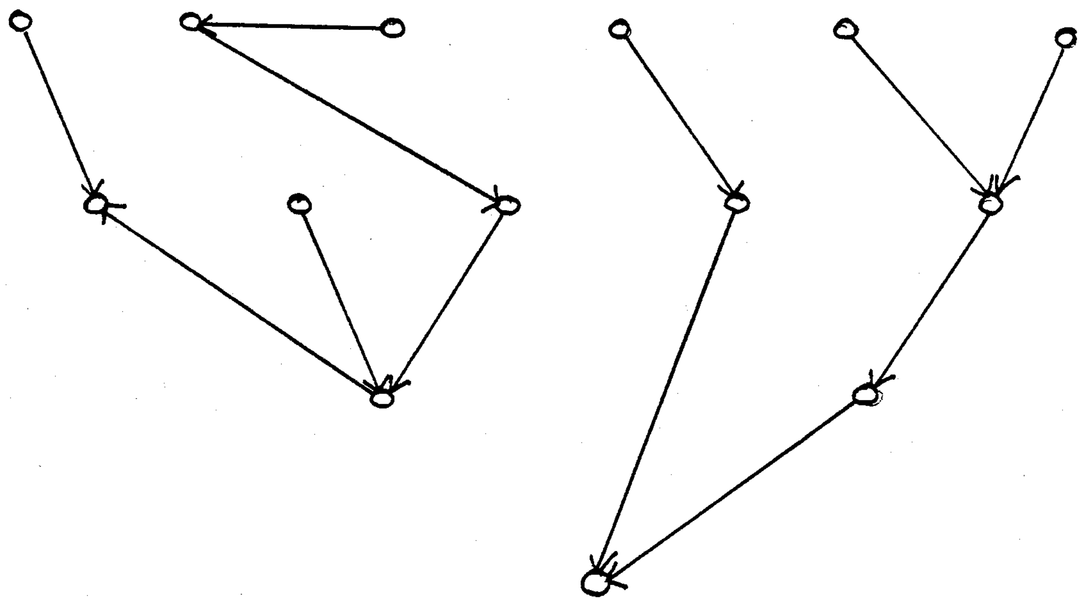
Fig.4,1, Concave function

Note the difference to the traffic assignment problem which either led to a linear or a convex objective function.

From the concavety of (4.2) follows that it is always better to assign all the flow going out from one vertex to one single arc (or filter-plant) than to split it up into different smaller flows. This intuitively obvious result can be formally proven by taking into account that the optimum of a concave programming problem, like (4.1) and (4.2), always must be in one of the extreme points of the given convex polyeder of (4.1) . But such a basic solution , as extreme points are called too, is characterized by having in the maximum as many non zero variables x_i and f_{ij} as there are conservation equations, i.e. as there are vertices in the network, thus to each vertex i only x_i or one f_{ij} may be positive. Going back to the definitions of chapter 2 this means that the optimal network only has vertices with outdgree 0 or 1 (outdegree of a vertex i is the number of arcs which have vertex i as their initial vertex), if only those arcs $(i,j)\varepsilon A$ are to be considered, for which $f_{ij} > 0$. A graph is called a tree if it has no circuit and if the outdegree of every vertex, except one (say vertex 1), is unity: the outdegree of vertex 1 (called the basis of the tree) being zero. In other words: the optimal network of our problem consists of a set of trees that are not connected with each other. Of course an optimal solution with only one tree is possible too. In each basis of a tree a filter plant is located, therefore the number of trees is equal to the number of filter plants. In Fig.4.2. a waste water network and a possible optimal solution is given as an example.



(a) waste water network



(b) Optimal solution

Fig.4.2.

As already stated in (4.1) we assume that each vertex can be the location of a filter plant, but some locations can be forbidden by assigning very large costs $a_i(x_i)$ to this location. The main question is then in which vertices to place filter plants. Of course, this question must be simultaneously solved with finding out which canals(arcs) to build. So far, only two algorithms seem to exist which solve problem (4.1)-(4.2). One is given by Ahrens (1974), who uses the fact that if all waste water sources g_i are integer, then the solution will also only have integer values for f_{ij} and x_i because the flow will not split up into smaller flows. Therefore he transforms the variables f_{ij} and x_i into weighted sums of boolean variables and then solves the problem with the additive algorithm of Balas, a well known enumerative algorithm. Yet, this approach does not seem very promising for larger networks. We rather follow the way suggested by Polyméris (1977). To use his algorithm we have to assume that the network originally given already is a tree, i.e. has no circuits and outdegree of all but one vertex is 1. This assumption seems to be rather restrictive. But in reality,most of the original networks seem to be trees or nearly trees (with very few circuits), because of the topographical situations (remember that rivers nearly always have tree-structure). If now the original network,in fact,has few circuits,one can solve the problem on all spanning trees (i.e. tree on a given graph that includes all vertices of the given graph) of the given network and then choose the best (cheapest) solution. If the number of spanning trees on a given graph G=(X,A) is low, then the following algorithm can be used to find all spanning trees. It should be noted that not every directed graph has a spanning tree. In this case the original problem is divided into smaller ones finding optimal solutions on a set of nonconnected trees. A detailed discussion of algorithms for the spanning tree problem is given in Christofides (1975).

Algorithm to find all spanning trees

Step 1: Start with an arbitrary spanning tree $T_o=(X,S_o)$ on $G=(X,A)$. Set $k=0$.

Step 2: Set $k=k+1$
Find another spanning tree T_k by removing such an arc $(x_i,x_j)\varepsilon S_{k-1}$ for which an arc $(x_i,x_l)\varepsilon A$ exists and for which no path from x_l to x_i exists and set $S_k=\{S_{k-1}-(x_i,x_j)\}\cup(x_i,x_l)$.
If no spanning tree T_k can be found, then Stop.

Step 3: Check, if this tree T_k has already been created. If so, delete this tree, mark the exchange of (x_i,x_j) to (x_i,x_l) as not being valid and go to Step 2. If tree T_k is new then store T_k and go to Step 2.

The proof that this algorithm works lies simply in the fact that Step 2 never creates a graph T_k with a circuit because a path x_l to x_i is not allowed and the outdegree of x_i remains 1, while all other outdegrees are unchanged.

We can now discuss the algorithm for finding the solution to (4.1) - (4.2), assuming that the given network already is a tree. The method used will be dynamic programming. We shall not give the theorems and proofs on which this algorithm is based and explained in detail by Polyméris (1977).

Let the given tree be $T = (X,S)$. Then we call Ω the class of all nonempty subsets of X for which a subset of S can be found, such that these vertices and arcs together form a partial subtree of T.

Let r: $\Omega \rightarrow X$ be a function, which states for each partial subtree $A \varepsilon \Omega$. its basis, which is the vertex with outdegree zero.

Let H: $\Omega \rightarrow$ set of subsets of X be a function, which gives for a set $A \varepsilon \Omega$ all vertices H(A) which do not belong to A, but for which an arc of S exists that connects each vertex from H(A) with a vertex of A.

Let K: $X \rightarrow \Omega$ denote a function, where K(i) for all $i \varepsilon X$ is the subset of all vertices of X for which a path to i exists (also $i \varepsilon K(i)$).

The meaning of the above definitions is illustrated in Fig.4.3.

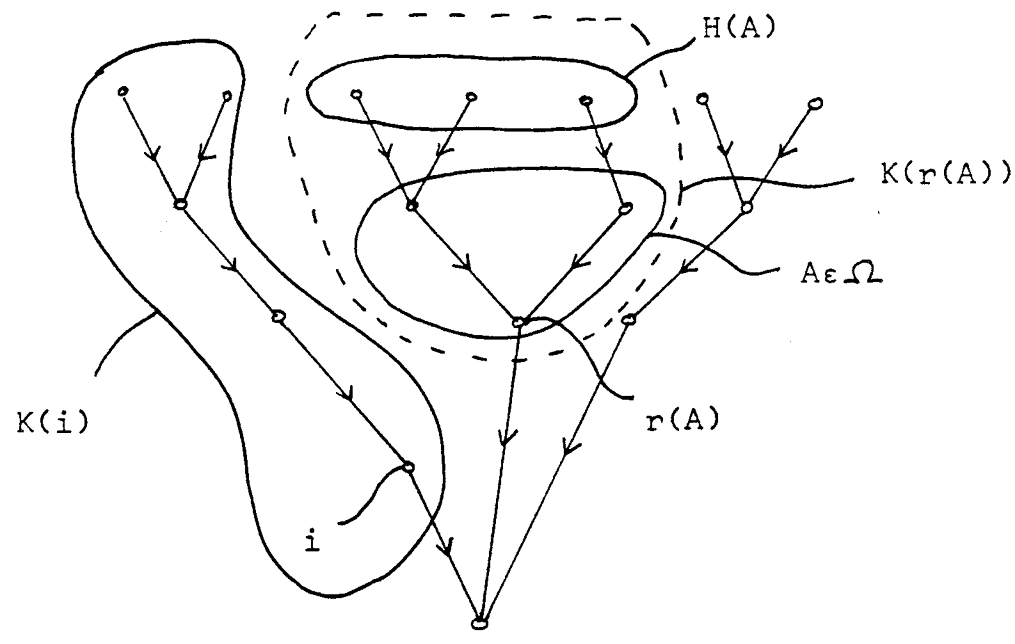


Fig. 4.3

Let G: $\Omega \rightarrow \emptyset$ (the class of all nonempty subsets of arcs S which together with a subset $A \varepsilon \Omega$ define a partial subtree of T), be a function that denotes all arcs which, together with a set of vertices of Ω, form a partial subtree of T which is then defined by (A,G(A)).

After these definitions, which are necessary to simplify the following notations, we look closer to the objective function (4.2).

Let us define q: $\Omega \rightarrow R_+$ as a function

$$q(A) = \sum_{i \varepsilon A} h_i \text{ for all } A \varepsilon \Omega \tag{4.3}$$

Now we want to find the total costs which arise if all waste water, created within a set of vertices $A \varepsilon \Omega$, is purified in a filter plant located at the basis of A, namely r(A). On this purpose we define a function f: $\Omega \rightarrow R$, which is given by

$$f(A) = a_{r(A)}(q(A)) + \sum_{\substack{(i,j)\varepsilon G(A) \\ j,i \varepsilon A}} b_{ij}(q(A \cap K(i))) \quad \text{for all } A \varepsilon \Omega \tag{4.4}$$

where a_i and b_{ij} are the costs defined in (4.2). f(A) as defined in (4.4) gives the costs for cleaning all waste water of A in r(A).

As we already discussed earlier, an optimal solution will be one where in some vertices all the water flowing to these vertices will be purified. Thus an optimal solution can be characterized by a set of partial subtrees of T, which are not connected and where the set of all vertices of these partial subtrees is the set of vertices of T, namely X itself.

Let now $\Lambda \subset \Omega$ denote such a class of sets of vertices of partial subtrees. If Λ describes the set of partial subtrees which minimize (4.2) then it must also be true that

$$\sum_{A \in \Lambda} f(A) \quad \text{is minimum} ,$$

compared with all other possible Λ'.

Let now $g: X \to R$ be a function defined by

$$g(i) = \min_{\substack{A \in \Omega \\ r(A)=i}} \{f(A) + \sum_{k \in H(A)} g(k)\} \qquad \text{for all } i \in X \qquad (4.5)$$

This function can now be computed recursively, starting at the "top" of the tree, which contains the vertices with indegree zero (that means $H(.)=0$ for these vertices) and then continuing with the vertices that are connected by an arc and so on until the basis $r(X)$ is reached.

Let $L: X \to \Omega$ denote the largest set $A \in \Omega$ for which $g(i)$ is minimum, $L(i)$ thus denoting the largest optimal set of vertices which send all their waste water to the filter-plant located in vertex $i \in X$.

Polyméris (1977) now proves that $g(r(X))$ gives the minimum costs of (4.2).

The optimal sets of vertices $L(i)$ can then be found as follows.

Start with $L(r(X))$. Find all vertices $k \in H(L(r(X)))$. For all these vertices k, $L(k)$ gives the next optimal sets.

Then find vertices $j \epsilon H(L(r(X)) \cup L(k))$; $k \epsilon H(L(r(x)))$ leading to L(j). This recursive search process must be performed until $H(L(r(X)) \ \ldots\ ) = 0$. Then the optimal partial subtrees of T have been found.

```
C ... *** PROGRAM FOR COMPUTING AN OPTIMAL WASTE
C ... *** WATER MANAGMENT SYSTEM
C
C ... INPUT
C
C ... N       NUMBER OF VERTICES
C ... SU      SUCCESSOR FUNCTION, I.E. SU(J) DENOTES THE
C             SUCCESSOR OF VERTEX J IN THE GIVEN TREE
C             IT HOLDS THAT I<SU(I)
C ... H(J)    AMOUNT OF WASTE WATER PRODUCED IN VERTEX J
C ... A       COEFFICIENTS FOR COST POLYNOMIAL GIVING THE
C             COSTS FOR BUILDING A FILTER PLANT AT
C             VERTEX I WITH SIZE X. THE POLYNOM IS THEN
C             GIVEN AS
C             A((I-1)*NA+1)*X**0+ ... +A((I-1)*NA+NA)*X**(1/(NA-1))
C ... NA      NUMBER OF COEFFICIENTS A FOR EACH VERTEX
C ... B       COEFFICIENTS FOR COST POLYNOMIAL GIVING THE
C             COSTS FOR BUILDING A CANAL FROM VERTEX I TO
C             VERTEX SU(I) WITH SIZE X. THE POLYNOM IS THEN
C             GIVEN AS
C             B((I-1)*NB+1)*X**0+ ... +B((I-1)*NB+NB)*X**(1/(NB-1))
C ... NB      NUMBER OF COEFFICIENTS B FOR EACH CANAL COST
C
C ... OUTPUT
C
C ... C1      MINIMUM COSTS FOR WASTE WATER MANAGMENT SYSTEM
C ... SFP(I)  DENOTES THE SIZE OF FILTER PLANT AT VERTEX I
C             (SFP(I)=0 MEANS THAT NO FILTER PLANT SHOULD
C             BE BUILT AT I)
C ... SCA(I)  DENOTES THE SIZE OF THE CANAL FROM I TO SU(I)
C
      SUBROUTINE WAWA(N,SU,H,A,NA,B,NB,C1,SFP,SCA)
      INTEGER N,SU(1),NA,NB,MM(30)
      REAL H(1),A(1),B(1),C1,SFP(1),SCA(1)
      INTEGER SET(930),LMM,LOP,OP(30)
      DO 5 I=1,N
5     SFP(I)=2.**18.
      DO 10 JJ=1,N
      LMM=1
      MM(1)=JJ
15    CONTINUE
      DO 20 I=1,N
      DO 30 J=1,LMM
      IF(I .EQ. MM(J)) GO TO 20
30    CONTINUE
      DO 25 J=1,LMM
      IF(SU(I) .NE. MM(J)) GO TO 25
      LMM=LMM+1
      MM(LMM)=I
      GO TO 15
25    CONTINUE
20    CONTINUE
      DO 35 I=1,LMM
      K1=2**17
      DO 40 J=I,LMM
      IF(K1 .LE. MM(J)) GO TO 40
```

```
      K1=MM(J)
      K2=J
40    CONTINUE
      MM(K2)=MM(I)
      MM(I)=K1
35    CONTINUE
      CALL MINI(LMM,MM,SU,H,A,NA,B,NB,SFP,LOP,OP,JJ,N)
      I=(JJ-1)*(N+1)+1
      SET(I)=LOP
      I=I+1
      I1=I+LOP-1
      DO 45 J=I,I1
      J1=J-I+1
45    SET(J)=OP(J1)
10    CONTINUE
      C1=SFP(N)
      DO 50 I=1,N
      J=N+1-I
      IF(SFP(J) .LE. 0.) GO TO 50
      J1=(J-1)*(N+1)+1
      J2=SET(J1)+J1
      J1=J1+1
      DO 55 I1=J1.J2
      I2=SET(I1)
      IF(I2 .EQ. J) GO TO 55
      SFP(I2)=0
55    CONTINUE
50    CONTINUE
      DO 60 I=1,N
60    SCA(I)=0
      DO 65 I=1,N
      IF(SFP(I) .GT. 0) GO TO 65
      SCA(I)=H(I)
65    CONTINUE
      N1=N-1
      DO 70 I=1,N1
      J=SU(I)
      IF(SFP(J) .GT. 0) GO TO 70
      SCA(J)=SCA(J)+SCA(I)
70    CONTINUE
      DO 75 I=1,N
      IF(SFP(I) .GT. 0) SFP(I)=H(I)
75    CONTINUE
      DO 80 I=1,N1
      J=SU(I)
      IF(SFP(J) .LE.0) GO TO 80
      SFP(J)=SFP(J)+SCA(I)
80    CONTINUE
      RETURN
      END
```

```
C ... *** PROGRAM FOR COMPUTING AN OPTIMAL WASTE
C ... *** WATER PARTIAL SUBTREE ON A GIVEN SUBTREE
C
C ... INPUT
C
C ... L      NUMBER OF VERTICES
C ... SU     SUCCESSOR FUNCTION, I.E. SU(J) DENOTES THE
C            SUCCESSOR OF VERTEX J IN THE GIVEN TREE
C            IT HOLDS THAT I<SU(I)
C ... H(J)   AMOUNT OF WASTE WATER PRODUCED IN VERTEX J
C ... A      COEFFICIENTS FOR COST POLYNOMIAL GIVING THE
C            COSTS FOR BUILDING A FILTER PLANT AT
C            VERTEX I WITH SIZE X. THE POLYNOM IS THEN
C            GIVEN AS
C            A((I-1)*NA+1)*X**0+ ... +A((I-1)*NA+NA)*X**(1/(NA-1))
C ... NA     NUMBER OF COEFFICIENTS A FOR EACH VERTEX
C ... B      COEFFICIENTS FOR COST POLYNOMIAL GIVING THE
C            COSTS FOR BUILDING A CANAL FROM VERTEX I TO
C            VERTEX SU(I) WITH SIZE X. THE POLYNOM IS THEN
C            GIVEN AS
C            B((I-1)*NB+1)*X**0+ ... +B((I-1)*NB+NB)*X**(1/(NB-1))
C ... NB     NUMBER OF COEFFICIENTS B FOR EACH CANAL COST
C ... N(I)   VERTICES IN THE GIVEN SUBTREE. IT HOLDS
C            THAT N(I)<N(I+1)
C ... JJ     ROOT OF THE VERTICES IN N
C ... G(I)   MINIMUM COSTS FOR CONSTRUCTING A WASTE
C            WATER NETWORK WITH ROOT I. IF G(I)=2**18
C            THESE COSTS ARE NOT YET COMPUTED
C ... LEN    TOTAL LENGTH OF THE ORIGINAL TREE
C
C ... OUTPUT
C
C ... G(JJ)        SEE ABOVE
C ... LOP          NUMBER OF VERTICES IN OP
C ... OP(I)        VERTICES BELONGING TO THE OPTIMAL TREE
C                  WITH ROOT JJ. IT HOLDS THAT OP(I)<OP(I+1)
C
      SUBROUTINE MINI(L,N,SU,H,A,NA,B,NB,G,LOP,OP,JJ,LEN)
      INTEGER N(1),SU(1),OP(1),L,NA,NB,LOP,JJ,M(30),II,KK,LL
      REAL H(1),A(1),B(1),G(1)
      LOGICAL LOG
      KK=L-1
      LL=1
5     CALL KOMB(L,N,LL,M,KK,II,SU,LOG,JJ)
      IF(LOG) RETURN
      QQ=0.
      DO 10 I=1,LL
      J=M(I)
10    QQ=QQ+H(J)
      FF=COMP(NA,A,QQ,JJ)
      DO 15 I=1,LL
      I1=M(I)
      IF(I1 .EQ. JJ) GO TO 15
      QQ=H(I1)
      I3=I-1
      DO 20 J=1,I3
```

```
      J1=M(J)
      IF(J1 .EQ. JJ) GO TO 20
      NN=J1
25    NN=SU(NN)
      IF(NN .EQ. JJ) GO TO 20
      IF(NN .NE.I1) GO TO 25
      QQ=QQ+H(J1)
20    CONTINUE
      FF=FF+COMP(NB,B,QQ,I1)
15    CONTINUE
      DO 30 I=1,LEN
      DO 45 J=1,LL
      IF(I .LT. M(J)) GO TO 50
      IF(I .EQ. M(J)) GO TO 30
45    CONTINUE
50    DO 35 J=1,LL
      IF(SU(I) .GT. M(J)) GO TO 35
      IF(SU(I) .LT. M(J)) GO TO 30
      FF=FF+G(I)
      GO TO 30
35    CONTINUE
30    CONTINUE
      IF(FF .GT. G(JJ)) GO TO 5
      G(JJ)=FF
      LOP=LL
      DO 40 I=1,LL
40    OP(I)=M(I)
      GO TO 5
      END
```

```
C .. *** FINDING ANOTHER PARTIAL SUBTREE FOR A GIVEN
C ... *** ROOT
C
C ... INPUT
C
C ... L       TOTAL NUMBER OF VERTICES IN GIVEN SUBTREE
C ... N(I)    I=1,2,...,L. DENOTES THE VERTICES IN THE
C             SUBTREE. IT HOLDS THAT N(I)<N(I+1).
C ... LL      NUMBER OF ACTUAL CHOSEN VERTICES OUT OF
C             VERTICES N(1),...,N(L)
C ... M(J)    J=1,...,LL. VERTICES IN THE PARTIAL SUBTREE
C             IT HOLDS THAT M(J)<M(J+1)
C ... KK      POINTER ON VECTOR N
C ... II      POINTER ON VECTOR M
C ... SU(I)   DENOTES THE SUCCESSOR VERTEX IN THE GIVEN
C             TREE. IT HOLDS THAT I<SU(I).
C ... JJ        ROOT VERTEX
C
C ... OUTPUT
C
C ... LL      SEE ABOVE
C ... KK      SEE ABOVE
C ... II      SEE ABOVE
C ... M(J)    SEE ABOVE
C ... LOG     IF LOG=.TRUE., THEN NO MORE PARTIAL SUB-
C             TREES EXIST
C
      SUBROUTINE KOMB(L,N,LL,M,KK,II,SU,LOG,JJ)
      INTEGER L,SU(1),N(1),M(1),LL,KK,II
      LOGICAL LOG
      LOG=.FALSE.
20    II=LL
25    IF(KK .EQ. L) GO TO 30
      KK=KK+1
      M(LL)=N(KK)
      GO TO 5
30    IF(II .EQ. 1) GO TO 35
      II=II-1
      DO 40 J=1,L
      IF(M(II) .GT. N(J)) GO TO 40
      KK=J
      GO TO 45
40    CONTINUE
45    L1=KK+LL-II+1
      IF(L1 .GT. L) GO TO 30
      DO 50 J=II,LL
      KK=KK+1
50    M(J)=N(KK)
      KK=L
      M(LL)=N(L)
      GO TO 5
35    IF(LL .EQ. L) GO TO 55
      LL=LL+1
      DO 60 J=1,LL
60    M(J)=N(J)
      KK=L
```

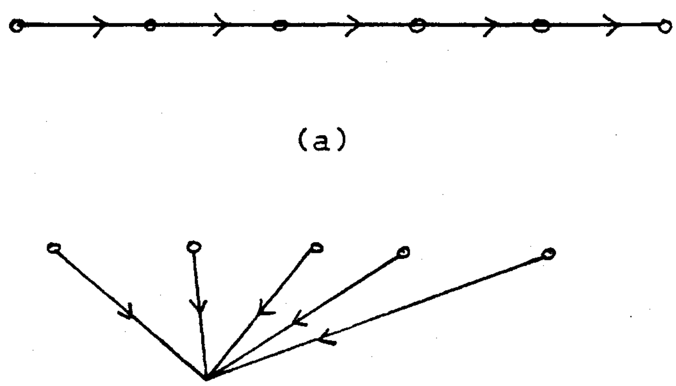
```
      M(LL)=N(L)
5     LL1=LL-1
      IF(LL .EQ. 1) RETURN
       DO 10 I=1,LL1
      L1=M(I)
      DO 15 J=I,LL
      IF(SU(L1) .GT. M(J)) GO TO 15
      IF(SU(L1) .EQ. M(J)) GO TO 10
      GO TO 20
15    CONTINUE
      GO TO 20
10    CONTINUE
      RETURN
55    LOG=.TRUE.
      RETURN
      END
```

```
C ... *** COMPUTATION OF VARIABLE COST
C ... ***
C
C ...
C ... INPUT
C
C ... M      NUMBER OF COEFFICIENTS PER VERTEX COST
C ... C      COEFFICIENTS OF VERTEX COSTS
C ... F      QUANTITY
C ... L      INDEX OF VERTEX FOR WHICH THE COSTS ARE COMPUTED
C
C ... OUTPUT
C
C ... COMP   COST ON VERTEX L WITH QUANTITY F
C
       FUNCTION COMP(M,C,F,L)
       INTEGER M,L
       REAL C(1)
       COMP=0
       IF(F .LE. 0) RETURN
       J=(L-1)*M
       DO 15 I=1,M
       J=J+1
       IF(I.EQ.1) GO TO 25
       COMP=COMP+C(J)*F**(1./(I-1.))
       GO TO 15
25     COMP=COMP+C(J)
15     CONTINUE
       RETURN
       END
```

Let us finally do some computational considerations. As each $A \varepsilon \Omega$ defines one vertex r(A) as its basis, the value of $f(A) + \sum_{k \varepsilon H(A)} g(k)$ has to be computed for all $A \varepsilon \Omega$ to finally find g(r(X)). Therefore the computing time will be proportional to the number of sets in Ω. For a general tree T, this number can hardly be forecasted. However, for simple tree structures we can derive this number.

In the case where the tree has only one top vertex (with indegree zero) as given in Fig. 4.4 (a), the number of sets in Ω , if the number of vertices in tree T is n, is given by

$$\frac{n(n+1)}{2} \tag{4.6}$$



(a)

(b)

Fig. 4.4

In the case where the tree with n vertices has n-1 top vertices as shown in Fig. 4.4 (b), the number of sets in Ω is

$$n + \sum_{i=1}^{n-1} \binom{n-1}{i} = 2^{n-1} - 1+n \qquad (4.7)$$

In this case Polyméris (1977) suggests a better usage of the concavity of the objective (4.2), which results in a much lower computation time than the one given in (4.7) but we will not go into this.

4.2. Location of emergency service facilities

Organisation of emergency services has received considerable interest in the last years. Like ambulance systems and fire prevention systems,all such emergency services have in common that they have to reach as quickly as possible the place where an emergency situation occurs. Therefore,the time between a telephone call announcing such an emergency and the arrival at the emergency place has to be minimized - the so called response time. Considering an area in which such an emergency system is to be built ,the problem of where to locate the emergency service facilities is a crucial one for determining the response time. As such emergency service facilities are usually cars using the road network of the area and as the vertices of such a road network can be road intersections as well as subareas of the given area, the location problem can be viewed as finding optimal locations at vertices in a given network. Now two ways of stating the problem are possible. We can either fix the costs for such an emergency service and optimize the service level or we can fix the service level and minimize its costs. Here we shall use the latter approach. The service level as defined by the response time can be measured, for example, by the average response time,as the response time depends on the location of the emergency,which is, of course, stochastic in nature, thus leading to a stochastic response time. But in many emergency cases,like accidents or a fire , not the average response time is the crucial parameter, but the maximum response time that can occur . We shall therefore fix the maximum response time, i.e. the maximum distance from a vertex where a service facility is located to any vertex which has to be served by this facility. As we assume that enough facilities are used, such that any emergency call can be answered immediatly and no delay can occur because all facilities are occupied, the number of locations of such emergency service facilities mainly determines the costs, if one thinks about the costs to build up a house or garage serving as a location and also to keep this house in good working conditions. Thus we see that costs are minimized if the number of such locations is minimized.

We can now formulate the problem completely: Given a directed or nondirected road network with travel costs on each arc. Then for a given maximum allowed travel cost (time) T, find the minimum number of vertices such that all other vertices can be reached from any one of those vertices in less than the maximum travel time T.

This problem can now be stated mathematically. If the maximum response time T has been decided upon, then, for any vertex i, only the set of vertices within T of i can provide acceptable emergency service to i; this set will be denoted as N_i . If d_{ji} is the minimum travel time (along the shortest path) from any vertex j to vertex i, the set N_i can be defined as

$$N_i = \{j \mid d_{ji} \leq T_{ij} \text{ a vertex possible for facility location}\} \tag{4.8}$$

If there are n vertices which have to be served by emergency facilities, there will be n sets N_i, and each set will have at least one number, if one takes $d_{ii}=0$. Note that these n vertices need not be all the vertices of the given network.

It can well be that some vertices only serve for facility location and also that some vertices may not serve for facility location. All these cases can be included into the definition of N_i.

To structure the mathematical formulation, the following decision variables are now defined:

$$x_j = \begin{cases} 0, & \text{if no facility is established at vertex } j \\ 1, & \text{if a facility is established at vertex } j \\ & \text{for all possible facility location vertices } j. \end{cases} \qquad (4.9)$$

As already discussed, any vertex i that has to be served by an emergency facility must have at least one facility location within T. Recalling that the set of potential facility locations within T of i is N_i and using (4.9), we can write this requirement as

$$\sum_{j \in N_i} x_j \geq 1 \qquad \text{for } i = 1,2,\ldots,n \qquad (4.1o)$$

and the objective z that is to be minimized is the total number of facility locations used

$$\text{min: } z = \sum_{j=1}^{m} x_j \qquad (4.11)$$

where m is the total number of possible facility locations. Note that $n \leq H$, $m \leq H$, where H is the number of vertices of the given network. (4.9),(4.1o) and (4.11) together give the complete description of the emergency service facilities - location problem. If the costs for locating the facilities are not the same at different vertices, we can also use the objective

$$\text{min: } \bar{z} = \sum_{j=1}^{m} c_j x_j \qquad (4.12)$$

where c_j denotes the cost for location at vertex j.

Hakimi (1964 and 1965) was the first to consider such problems. Also in Christofides (1975) some algorithms for solving such location problems are discussed. We shall follow here the approach given by Toregas et.al. (1971), who suggested an heuristic algorithm based on the simplex-algorithm that seems to give good results and has the advantage of solving large problems, which is not the case for the exact algorithm developed by Hakimi (1964 and 1965).

Algorithm to find the optimal location vertices

Step 1:

For a given maximum response time T compute the member vertices of each set N_i for all vertices i to be served.

Step 2:

Solve (4.1o) and (4.11) or (4.12) as a linear programming problem with

$$x_j \geq 0 \qquad \text{for } j = 1,2,\ldots,m \qquad (4.13)$$

If the solution is integer (i.e. all x_j = 0 or 1), then the optimal solution has been found. Stop.
If a fractional solution has been found with z^o being the value of the objective for this solution, then solve (4.11) or (4.12) under the restriction (4.1o) and (4.13) with the additional constraint

$$\sum_{j=1}^{m} x_j \geq [z^o] + 1 \qquad (4.14)$$

where z^o is the integer part of z^o. Stop.

Toregas et al. (1971) report that, although fractional solutions could occur if (4.14) holds, the problems they solved always turned out to be integer, either immediatly or with the help of (4.14). Thus, all that is needed is a standard linear programming code and some subroutine that produces for a given network and a given T the constraints (4.1o). Of course, the given model is only meaningful if the travel costs are deterministic rather than stochastic by nature, which is not the case if travel costs depend on travel flow and this flow varies largely. Therefore, this model seems to be less suitable to urban areas with heavy traffic congestion, in which case a stochastic model is appropriate.

4.3. Optimal network for an airline

We shall discuss the following problem: Given a set of airports (the vertices) which should be connected somehow by airplanes. Then two problems arise: Either the transportation demand (trip matrix) between all pairs of vertices is given and this demand has to be satisfied with a minimum of necessary flight hours, say per week. As the necessary flight hours determine the number of aircrafts (for example, 26 flight hours necessary within 24 hours can only be produced by at least two airplanes), this model can be viewed as one to determine the minimal number of airplanes for a given demand. Such a model is discussed in Miller (1967).

A somehow complementary problem to the above one is the optimal fulfilling of a given transportation demand where the number of airplanes and,therefore flight hours per week is fixed. This model can be viewed as one to determine the optimal airplane - supply for a given transportation demand and a fixed number of airplanes.

Such a model is mainly of interest to a domestic airline because between nearly all vertices (airports) an arc (flight connection) exists, thus leaving enough possibilities for optimizing, while in international airlines usually only flight connections between a foreign airport and a domestic one exist and none between two foreign airports, thus reducing the amount of possible networks drastically. For simplicity we shall be considering only one type of network flow (i.e. one type of airplane), although this model can be generalized.

Let us call the airports (vertices) $i \epsilon X$ and the possible flight connections (arcs) $(i,j) \epsilon A$, where the given network is $G=(X,A)$. Let f_{ij} denote the flow on arc (i,j), meaning the number of airplanes flying on this route, say per week. To each arc assigned are the flight costs (time) c_{ij} including also the necessary time for preparing the airplane on ground (loading, unloading, filling up etc.).

We now want an optimal airplane supply for a given trip matrix. As well known, a flight, which does not go non-stop from the origin to the destination airport, takes much more time than a non-stop flight. Thus, the demand will best be fulfilled if as many people as possible can go with a non-stop flight connection. But this objective cannot be optimized directly because this would mean not only an optimal assignment of flights to arcs, but also of passengers to flights which would complicate the model a lot. We therefore try to give an objective which is closely related to the original one but has the advantage of being easy to handle.

If the trip matrix is given by (g_{ij}), the objective is

$$\max: \sum_{(i,j) \epsilon A} f_{ij} \cdot g_{ij} = z \qquad (4.15)$$

which results in giving proportional weight to flights between two vertices according to the demand. This objective has, of course, to be optimized under certain constraints, the first of which are the conservation equations, which have the simple form

$$\sum_{\substack{j \\ (i,j)\varepsilon A}} f_{ij} - \sum_{\substack{j \\ (j,i)\varepsilon A}} f_{ji} = 0 \qquad , \quad i\varepsilon X \qquad (4.16)$$

as no airplanes are destroyed or created in any vertex. If we denote p the number of passengers that can be carried by a single airplane (this number is equal for all airplanes as we consider only one type), then $p.f_{ij}$ gives the number of passengers that can be carried from vertex i to j in a week. Although people may have to use such a flight from i to j even when i is not their origin or j not their destination, if a non-stop flight is not available to them, the supply on this route should not substantially exceed the demand, therefore leading to

$$0 \leq f_{ij} \leq \left[\frac{g_{ij} \, . \, (1+\beta)}{p}\right] \quad , \quad (i,j)\varepsilon A \qquad (4.17)$$

where ß is the allowed oversupply and [.] denotes the integer part of the number. Also, we want each vertex to be served by at least one airplane a week and a connection possibility between all pairs of vertices (A is, of course, connected), thus

$$1 \leq \sum_{\substack{j \\ (i,j)\varepsilon A}} f_{ij} \qquad , \qquad i\varepsilon X \; , \qquad (4.18)$$

and between all pairs of vertices i and j there exists a path. Naturally we assume that all demands g_{ij} are at least as large as to justify one flight per week on all arcs in the minimum (otherwise this arc will not be considered at all). Finally, we have to restrict the total number of flight hours by

$$\sum_{\substack{i,j \\ (i,j)\varepsilon A}} f_{ij} \cdot c_{ij} \leq B \qquad (4.19)$$

If, for example, the air company runs three airplanes for 1o hours a day,each during 7 days, then B can be computed as 21o hours/week.

Now, the objective function (4.15) can be changed to a minimization problem by multiplying with (-1). Then, this transformed objective together with the constraints (4.16) and (4.17) is a minimum cost flow problem for which we already developed an algorithm in chapter 3.3.1. But if we further consider the constraints (4.18) and (4.19), no special algorithm is known for this problem and therefore only an algorithm for the general linear, integer programming problem seems to be appropriate. Unfortunately, such an algorithm will only apply for small networks as the number of integer variables is approximately growing with n^2, where n is the number of vertices of G=(X,A). Thus, a heuristic algorithm is meaningful in this case. For this algorithm we shall assume that for each arc (i,j)εA,there also exists an arc (j,i)εA. This assumption in practice is always satisfied. The idea of the algorithm is to find quickly a feasible solution that satisfies all the constraints, continuing then by sequentially assigning flights to arcs, which are still feasible and maximize the objective. The problem of finding a feasible solution is very similar to the problem of computing a Hamiltonian circuit. (A Hamiltonian circuit is a circuit passing once,and only once,through each vertex of the graph). If we want to find the least cost Hamiltonian circuit this would be the well known travelling salesman problem, which we shall discuss in a later chapter. Here now we are looking for a circuit that passes at least once through all vertices and has low cost. Such a circuit is then a good feasible solution for our original problem. Considering the weights of the objective g_{ij} and the travelling costs c_{ij}, it is obvious that if two equal demands $g_{ij} = g_{lk}$ exist and $c_{ij} < c_{lk}$, then a flight should be scheduled to arc (i,j). Bearing this in mind, we consider new arc costs as

$$d_{ij} = \frac{g_{ij}}{c_{ij}} \quad , \quad (i,j)\varepsilon A \qquad (4.2o)$$

for the given network G = (X,A).

Algorithm for solving the airline problem

Step 1:

Start with an arbitrary vertex $i\varepsilon X$ and mark this vertex as the starting point. Mark all other vertices as being unserved and set $f_{ij}=0$, $(i,j)\varepsilon A$.

Step 2:

For vertex $i\varepsilon X$ find all unserved vertices j, for which

$f_{ij}+1 \leq \left[\frac{g_{ij}(1+\beta)}{p}\right]$, that means find

$$j\varepsilon H_i = \{j \mid (i,j)\varepsilon A \text{ , } j \text{ unserved and } f_{ij} \leq \left[\frac{g_{ij}(1+\beta)}{p}\right] - 1\}$$

If $H_i \neq \emptyset$ then choose this vertex k, for which

$$d_{ik} = \max_{j\varepsilon H_i} (d_{ij}).$$

Mark vertex k as being served and set

$$f_{ik} = f_{ik} + 1 \text{ .}$$

Set i = k and perform again Step 2.

If $H_i = \emptyset$, go to Step 3.

Step_3:

Let $U \subset X$ be the set of unserved vertices.
If $U \neq \emptyset$, then find the shortest paths p_{ij} from i to all $j \varepsilon U$. Check, if for all arcs of the shortest paths it holds that

$$f_{ij} \leq \left[\frac{g_{ij}(1+\beta)}{p}\right] - 1 \qquad (4.21)$$

and eliminate those paths, for which (4.21) is not satisfied. If no paths exist for which (4.21) holds, then no feasible solution can be found. Stop. If such paths exist, choose the vertex k with the minimum shortest path. Mark k as being served and set for all arcs along this shortest path

$$f_{ij} = f_{ij} + 1 \; .$$

Set i=k and got to Step 2.
If $U = \emptyset$ go to Step 4.

Step_4:

Find shortest path from vertex k to the starting point (vertex) of Step 1 that satisfies (4.21). If no such path exists, then no feasible solution can be found, Stop.

If such a path exists, then set for all arcs along this path

$$f_{ij} = f_{ij} + 1.$$

Proof, if

$$\sum_{(i,j)\varepsilon A} f_{ij} \cdot c_{ij} \leq B \qquad (4.22)$$

If (4.22) is not satisfied, then no feasible solution could be found. Stop.
Otherwise, go to Step 5.

Step 5:

Among all arcs $(k,l)\varepsilon A$ choose the one, for which

$$f_{kl} \leq \left[\frac{g_{kl}(1+\beta)}{p}\right] - 1$$

$$f_{lk} \leq \left[\frac{g_{lk}(1+\beta)}{p}\right] - 1$$

$$\sum_{(i,j)\varepsilon A} f_{ij}\, c_{ij} + c_{kl} + c_{lk} \leq B$$

and for which

$$\frac{g_{kl}}{c_{kl}} + \frac{g_{lk}}{c_{lk}} = \max_{(i,j)\varepsilon A}\left(\frac{g_{ij}}{c_{ij}} + \frac{g_{ji}}{c_{ji}}\right).$$

among all possible arcs.

Is such an arc exists, set

$$f_{kl} = f_{kl} + 1$$

$$f_{lk} = f_{lk} + 1$$

and perform again Step 5.

If no such arc exists, then a solution has been found. Stop.

The heuristic algorithm will produce a fairly good result except in cases where a feasible solution cannot be found. This can occur , when for most of the arcs

$$\left[\frac{g_{ij}(1+\beta)}{p}\right] \approx 1$$

or, when B in (4.19) gives a very tight bound. Both cases are rather unlikely in practical situations.

```
C ... *** ALGORITHM FOR SOLVING THE AIRLINE PROBLEM
C ... ***
C
C ... INPUT
C
C ... N       NUMBER OF VERTICES
C ... G(L)    TRIP MATRIX THAT IS NUMBER OF PEOPLE WHO
C             WANT TO TRAVEL FROM VERTEX I=(L-1)/N+1
C             TO VERTEX J=L-((I-1)*N)
C ... C(L)    FLIGHT COSTS (TIME) FROM VERTEX I TO J
C             (AS DEFINED ABOVE). IF C(L)=0 THEN NO
C             DIRECT FLIGHT FROM I TO J IS ALLOWED.
C ... BETA    ALLOWED OVERSUPPLY OF SEATS ON EACH ROUTE
C ... P       NUMBER OF SEATS PER AIRPLANE
C ... B       MAXIMUM AVAILABLE FLIGHT HOURS PER TIME UNIT
C
C ... OUTPUT
C
C ... F(L)    NUMBER OF DIRECT FLIGHTS FROM I TO J
C ... LOG     IF LOG=FALSE THEN NO FEASIBLE SOLUTION
C             HAS BEEN FOUND
C
      SUBROUTINE AIRL(N,G,C,BETA,P,F,LOG,B)
      INTEGER N,G(1),C(1),P,F(1),H(90),HH,IG(900),ID(900),B
      LOGICAL LOG,ICA
C
C ... STEP 1
C
      LOG=.TRUE.
      M=N*N
      DO 5 I=1,M
5     F(I)=0
      HH=1
      H(HH)=1
      DO 35 I=1,M
35    IG(I)=C(I)
      CALL SPII(N,IG,ID,ICA)
C
C ... STEP 2
C
10    D=0
      KD=0
      DO 15 I=1,N
      J=H(HH)
      L=IND(J,I,N)
      IF(C(L) .LE. 0) GO TO 15
      DO 20 J=1,HH
      IF(H(J) .EQ. I) GO TO 15
20    CONTINUE
      E=FLOAT(G(L))/FLOAT(C(L))
      IF(D .GE. E) GO TO 15
      D=E
      KD=I
15    CONTINUE
      IF(KD .EQ. 0) GO TO 25
      J=H(HH)
```

```
      L=IND(J,KD,N)
      F(L)=F(L)+1
      HH=HH+1
      H(HH)=KD
      DO 76 I=1,N
      DO 77 J=1,HH
      IF(H(J) .EQ. I) GO TO 76
77    CONTINUE
      GO TO 10
76    CONTINUE
      GO TO 30
C
C ... STEP 3
C
25    I=H(HH)
      JJ=0
      MI=2**17
      DO 40 J=1,N
      DO 55 K=1,HH
      IF(H(K) .EQ. J) GO TO 40
55    CONTINUE
      L1=IND(I,J,N)
      IF(MI .LE. IG(L1)) GO TO 40
      J1=J
45    K1=J1
      J1=IND(I,K1,N)
      J1=ID(J1)
      L=IND(J1,K1,N)
      AA=G(L)*(1.+BETA)/P-1.
      LL=INT(AA)
      IF(F(L) .GT. LL) GO TO 40
      IF(J1 .NE. I) GO TO 45
      JJ=J
      MI=IG(L1)
40    CONTINUE
      IF(JJ .NE. 0) GO TO 50
      LOG=.FALSE.
      RETURN
50    J1=JJ
      HH1=HH+1
60    K1=J1
      J1=IND(I,K1,N)
      J1=ID(J1)
      L=IND(J1,K1,N)
      F(L)=F(L)+1
      HH=HH+1
      H(HH)=K1
      IF(J1 .NE. I) GO TO 60
      HH2=HH
65    MM=H(HH1)
      H(HH1)=H(HH2)
      H(HH2)=MM
      HH1=HH1+1
      HH2=HH2-1
      IF(HH1 .LT.HH2) GO TO 65
      DO 66 I=1,N
      DO 67 J=1,HH
```

```
      IF(H(J) .EQ. I) GO TO 66
67    CONTINUE
      GO TO 10
66    CONTINUE
C
C ... STEP 4
C
30    I=H(HH)
      J=1
70    K1=J
      J=IND(I,K1,N)
      J=ID(J)
      L=IND(J,K1,N)
      AA=G(L)*(1.+BETA)/P-1.
      LL=INT(AA)
      IF(F(L) .GT. LL) GO TO 75
      IF(J .NE. I) GO TO 70
      J=1
80    K1=J
      J=IND(I,K1,N)
      J=ID(J)
      L=IND(J,K1,N)
      F(L)=F(L)+1
      IF(J .NE. I) GO TO 80
      B1=0
      DO 85 I=1,M
85    B1=B1+F(I)*C(I)
      IF(B1 .LE. B) GO TO 90
      LOG=.FALSE.
      RETURN
75    DO 95 II=1,M
      IG(II)=C(II)
      LL=G(II)*(1.+BETA)/P-1.
      IF(F(II) .GT. LL) IG(II)=0
95    CONTINUE
      CALL SPII(N,IG,ID,ICA)
      J=1
      L=IND(I,J,N)
      IF(IG(L) .LT. 2**17) GO TO 30
      LOG=.FALSE.
      RETURN
C
C ... STEP 5
C
90    LL1=0
      LL2=0
      AA=0
      DO 100 I=1,N
      DO 105 J=1,N
      IF(I .EQ. J) GO TO 105
      L1=IND(I,J,N)
      L2=IND(J,I,N)
      IF(C(L1).LE.0 .OR. C(L2).LE.0) GO TO 105
      LL=G(L1)*(1.+BETA)/P-1.
      IF(F(L1) .GT. LL) GO TO 105
      LL=G(L2)*(1.+BETA)/P-1.
      IF(F(L2) .GT. LL) GO TO 105
```

```
      BB1=B1+C(L1)+C(L2)
      IF(BB1 .GT. B) GO TO 105
      AAA=FLOAT(G(L1))/FLOAT(C(L1))+FLOAT(G(L2))/FLOAT(C(L2))
      IF(AA .GE. AAA) GO TO 105
      AA=AAA
      LL1=L1
      LL2=L2
105   CONTINUE
100   CONTINUE
      IF(AA .LE. 0) RETURN
      B1=B1+C(LL1)+C(LL2)
      F(LL1)=F(LL1)+1
      F(LL2)=F(LL2)+1
      GO TO 90
      END
```

4.4. Optimal network of a pipeline system

In this chapter we shall be dealing with the problem of constructing a pipeline system that can transport natural-gas from the gas fields to a separation plant, where the gas is separated from its valuable by-products and impurities. Because usually for gas produced from onshore fields where the separation is performed directly at the well, the following model is mainly devoted to offshore wells where the gas is transported through pipelines to some separation plant on land. The methods for analysing such a problem were first presented by Rothfarb et al. (197o) and some faster but approximative methods were given by Zadeh (1973). We shall state a different approach that is like the one by Rothfarb heuristic by nature and tries to reduce the time-consuming exact computation of the optimal pipe diameters. - The design of an offshore natural-gas system has two aspects. First, to reduce investment costs, the total length of all pipelines (arcs) should be as short as possible, but connect all gas fields (vertices) with the separation plant. It is quite obvious that the resulting network therefore will be a spanning tree of the original network G(X,A), which is the network of all gasfields (vertices) and all possible pipelines.

Second, for minimizing the operating costs, the loss of gas pressure on its way from a gas field to the separation plant should be as low as possible. The maximum allowable pressure is some constant P_{max}, which is the same for all types of pipelines and the pressure available at each well is at least P_{max} .Because the gas has to be recompressed at the separation plant, the cost for this recompression is determined by the lowest pressure of gas arriving from any well. As the pressure is the same at all wells, the lowest gas pressure can be found in the pipeline with the greatest pressure lost (the so-called critical pipeline path). The loss of gas pressure is a function of the pipeline length and the pipeline parameter

$$\Delta P = P_2 - P_1 = F(L,D), \qquad (4.23)$$

where P_1 is the output pressure, P_2 is the input pressure, D is the pipeline diameter and L is the pipeline length. Of course, the longer the pipeline will be, the larger the difference P_2-P_1 will be, while in contrast, P_2-P_1 will be decreasing if D increases.

As it seems impossible to solve the optimization problem of finding the optimal spanning tree and pipeline diameters in one step, the problem is divided into two subproblems: first, to find an optimal spanning tree: then an optimal diameter for each arc of the tree such that the sum of the investment and operating costs over a given planning horizon is minimized.

Because the compression cost depends on the highest loss pressure in any pipeline path and because the loss of pressure for a given diameter depends on the length, it is obvious that we should find a tree such that the longest pipeline path connecting a gas field (vertex) with the separation plant is minimized. Having found this path, the other arcs should be included in a way that no path is longer than the critical path and that the sum over the length of all arcs is minimized in order to minimize the investment costs. For the so determined spanning tree, an assignment of a pipeline diameter to each arc of the tree must be performed to minimize the total costs. Because the largest diameter will minimize loss of pressure but has the highest investment costs, the optimal diameter can only be found if the operating cost for a given loss of pressure and the investment cost for a given diameter are known and the planning horizon is given.

<u>Algorithm for finding the optimal spanning tree</u>

<u>Step 1</u>:

For the given network G=(X,A), where $x_o \varepsilon X$ is the separation plant and the arcs εA give all possible pipeline connections between vertices and to each arc (i,j) a length c_{ij} is assigned, find the shortest paths between x_o and all other vertices $j \varepsilon X$. Denote the length of the shortest path from j to x_o with p_j. Order the vertices such that $p_1 \geqslant p_2 \geqslant \dots \geqslant p_n$, where n is the number of vertices in X (besides x_o).

<u>Step 2</u>:

Put x_o in $Y \subset X$, the set of all already with x_o connected vertices. Set k=1.

<u>Step 3</u>:

Find all shortest paths q_{kj} from vertex k to all vertices $j \varepsilon Y$ in the given network G=(X,A). (Note that for k=1 this path already has been found to be p_1).

Find

$$q_{kr} = \min_{j \varepsilon Y} q_{kj} \tag{4.24}$$

such that

$$q_{kr} + q_{ro} \leqslant p_1$$

Include vertex k and all vertices that lie on the path q_{kr} into Y. Include all arcs that lie on the path q_{kr} into $S \subset A$ for the spanning tree T = (X,S)

<u>Step 4</u>:

Set k = k+1 . If $k > n$ then Stop.
If $k \leqslant n$ go to Step 5.

Step 5:

If $k \in Y$, then go to Step 4.
If $k \notin Y$, then go to Step 3.

The algorithm finds a spanning tree of G=(X,A) for which the longest path in T=(X,S) between any vertex and x_o is as short as possible.

The other arcs are chosen in a way to minimize the sum of the length of all arcs in S. This is performed by (4.24).

Having found the spanning tree T=(X,S),the assignment of the pipeline diameters still remains. We shall not go into this problem which has been discussed for a discrete number of possible diameters by Rothfarb et al. (197o) and for a continuous number of pipeline diameters (restricted to a maximum and a minimum one) by Zadeh (1973).

```
C ... *** PROGRAM FOR FINDING A SPANNING TREE ON A
C ... *** GIVEN GRAPH, WHERE THE LONGEST SHORTEST PATH
C ... *** IS MINIMUM AND THE TOTAL LENGTH OF THE
C ... *** SPANNING TREE IS MINIMAL (PIPELINE PROBLEM)
C ... ***
C
C ... INPUT
C
C ... N       NUMBER OF VERTICES
C ... C(L)    LENGTH OF THE ARC FROM I=(L-1)/N+1 TO
C             VERTEX J=L-((L-1)*N), L=1,...,N*N. IF
C             C(L)=0, THEN NO ARC EXISTS.
C
C ... OUTPUT
C
C ... F(L)    SUCCESSOR FUNCTION, I.E. F(L) DENOTES THE
C             SUCCESSOR OF VERTEX L IN THE FOUND SPANNING
C             TREE
C
       SUBROUTINE OPTREE(N,C,F)
       INTEGER N,C(1),F(1),IG(900),ID(900),A(30),B(30)
       LOGICAL LOG
C
C ... STEP 1
C
       M=N*N
       DO 5 I=1,M
5      IG(I)=C(I)
       CALL SPII(N,IG,ID,LOG)
       MM=2**17
       LL=1
       A(LL)=1
10     MN=0
       LL=LL+1
       K=0
       DO 15 I=2,N
       J=IND(I,1,N)
       IF(IG(J).GE.MM .OR. MN.GT.IG(J)) GO TO 15
       IF(MN .EQ. IG(J)) GO TO 20
       MN=IG(J)
       LL=LL-K
       K=0
       A(LL)=I
       GO TO 15
20     K=K+1
       LL=LL+1
       A(LL)=I
15     CONTINUE
       IF(LL .EQ. N) GO TO 25
       MM=MN
       GO TO 10
C
C ... STEP 2
C
25     LL=1
       B(LL)=1
```

```
      K=2
C
C ... STEP 3
C
30    MM=2**17
      KK=A(K)
      L=IND(A(2),1,N)
      IP1=IG(L)
      IG(1)=0
      DO 35 I=1,LL
      J=B(I)
      L=IND(KK,J,N)
      L1=IND(J,1,N)
      IPP=IG(L)+IG(L1)
      IF(IPP .GT. IP1) GO TO 35
      IF(IG(L) .GE. MM) GO TO 35
      MM=IG(L)
      II=J
35    CONTINUE
      J=II
40    K1=J
      J=IND(KK,K1,N)
      J=ID(J)
      LL=LL+1
      B(LL)=J
      F(J)=K1
      IF(J .NE. KK) GO TO 40
C
C ... STEP 4
C
50    K=K+1
      IF(K .GT. N) RETURN
C
C ... STEP 5
C
      DO 45 I=1,LL
      IF(A(K) .EQ. B(I)) GO TO 50
45    CONTINUE
      GO TO 30
      END
```

4.5. Optimal expansion of a railway system

With the growing interest in mass transportation systems, the improvement of railway systems has become an important question in many countries. Because most of the railway networks, at least in Europe, were built at the beginning of this century, these networks do not fit in many cases to present transportation demands. Some lines that used to be essential are not so any more, while others have been of growing importance. Of course, investments for improving the situation are restricted and therefore the question remains, which possible improvements should be realized under the given budget constraint and which should be left for consideration at some time in the future. Improvement can mean two things: improving existing lines (arcs) for higher speed or capacity and building completely new arcs connecting towns (vertices) that have not been directly connected yet. Both cases can easily be combined, if the improvement of an existing arc is considered as building a new arc with shorter transportation time than the old one. To each new arc (i,j) (combining vertices i and j) assigned are the construction costs q_{ij} and to all (new and old) arcs (i,j) assigned are the transportation costs (time) c_{ij}. As we are considering a railway system, we do not introduce arc capacities as a constraint, because in practice so far, the capacity of an arc has rarely been restrictive. It is more the number of available railway-coaches that seemsto restrict the transportation capacity of a railway-system.

Yet, we have only discussed the constraints for the network improvement and not its objectives. But quite naturally we can adapt the objective of chapter 3.3. and try to minimize total travel time for a given trip-matrix. To do so, we must first know how trains, and therefore people, will travel along a given network. Because no capacity constraints for arcs are given, and because the travel time does not depend on the travel flow, it is obvious that people will travel along shortest paths between their origin and their destination and usually routes for trains are chosen to lie on shortest paths as well. At least, if there are two trains, one along the shortest path between two vertices and one not, people will, according to Wardrop's principles, choose the train along the shortest path.

We have now completely defined the problem and can state it more formally in the following way:
Let P be the set of all subsets of I (the set of all possible arcs to be constructed). Then for a given network $G=(X,A)$, a given tripmatrix $[t_{ij}]$, where $i,j \in X$, and a given budget B, find sets $O \in P$, for which the investment costs q_{ij} do not exceed the budget

$$\sum_{(i,j) \in O \subset P} q_{ij} \leq B \tag{4.25}$$

where $i,j \in X$ (we do not consider to connect new vertices that are not already a member of X). For all such feasible sets $O \in P$, find the one for which

$$\min: \quad F = \sum_{\substack{i,j \in X \\ O \in P}} t_{ij} \, p^{O}_{ij} \tag{4.26}$$

where p^{O}_{ij} denotes the shortest path from vertex i to j in the network $G^{O} = (X, A \cup O)$ and F is the sum of the travel time of each passenger over all possible origin-destination pairs.

Because the construction of an arc can never result in increasing shortest paths, it must hold that for any set $R \subset O \in P$

$$\sum_{i,j \in X} t_{ij} \, p^{R}_{ij} \geq \sum_{i,j \in X} t_{ij} \, p^{O}_{ij} \tag{4.27}$$

For the optimal solution it is therefore sufficient to take only those sets $O \varepsilon P$ into consideration, for which

$$\sum_{(i,j)\varepsilon O} q_{ij} \leq B \qquad (4.28)$$

and

$$\sum_{(i,j)\varepsilon Q} q_{ij} > B \quad \text{for all } Q \varepsilon P \text{ with } Q \supset O.$$

Such sets O we shall call maximal sets and the set of all maximal sets we denote by $M \subset P$. Although the number of elements in M cannot be given generally in a formula, it is usually much smaller than the number of elements in P, which is, if there are n arcs in I, 2^n-1.

Let us consider building costs for arcs in I, which are all equal and let the budget B be given such that exactly n-m arcs can be built. Then M, the set of all maximal sets, contains

$$\binom{n-m}{m}$$

elements. If we remember formula (4.7), it holds that

$$\binom{n-m}{m} = 2^n-1 - \sum_{\substack{i=1 \\ i \neq m}}^{n} \binom{n}{i} \quad . \qquad (4.29)$$

Because $\binom{n}{i} = \binom{n}{n-i}$, the worst case happens to be $m = [n/2]$, the integer part of n/2. In this case, (4.29) can be written as

$$\binom{n-\ n/2}{n/2} = 2^n-1-(2^{[n/2]-1}-1+2^{[(n+1)/2]-1}-1) \ .$$

Although this upper bound in a special case is not very promising, in practical cases the number of arcs that can be built by constraint (4.25) will be rather low and further reduced by (4.28). It is therefore meaningful to find all maximal sets (in M) and compute the objective (4.26) for all such sets, thus finding the minimal one. If a maximal set has been found, the problem of computing (4.26) reduces to finding the shortest paths p^o_{ij} between all pairs of vertices for which Floyd's algorithm can be used. Yet, we have only to state an algorithm to find all maximal sets efficiently.

Let the arcs in I be called j=1,2,...,n. Let q_j be the construction cost for arc j.

Algorithm for finding all maximal sets:

Step 1:

Set x(2) = x(3) = ... = x(n) = 0.
Set x(1) = 1 and k = 1. Go to Step 2.

Step 2:

Compute

$$E = \sum_{i=1}^{k} q_{x(i)} .$$

If $E \leq B$, go to Step 3
else go to Step 6.

Step 3:

If $x(k) < n$, set k=k+1, set x(k) = x(k-1) + 1 and go to Step 2

otherwise the set of arcs x(1), x(2),..., x(k) is a maximal set.
Set x(k) = x(k)+1 and go to Step 4.

Step 4:

If $k > 1$, set k = k-1, set x(k) = x(k) + 1 and go to Step 5,
If $k \leq 1$, stop - all maximal sets have been found.

Step 5:

If $x(k+1) - x(k) \leq 1$, go to Step 4
otherwise go to Step 2.

Step 6:

If $x(k) < n$, set $x(k) = x(k) + 1$ and go to Step 2.
Otherwise, if $k > 1$, the set of arcs $x(1), x(2), \ldots, x(k-1)$ is a maximal set. Go to Step 7,
Otherwise, if $k=1$, Stop - all maximal sets have been found.

Step 7:

Set $k=k-1$, set $x(k) = x(k) + 1$ and go to Step 2.

The algorithm, although looking rather complicated, is very easy to understand if we look at an example with 5 arcs. We can interpret the algorithm as a type of branching technique with (4.25), as its bounds for deciding to backtrack. The branching tree for 5 arcs is given in Fig.4.5.

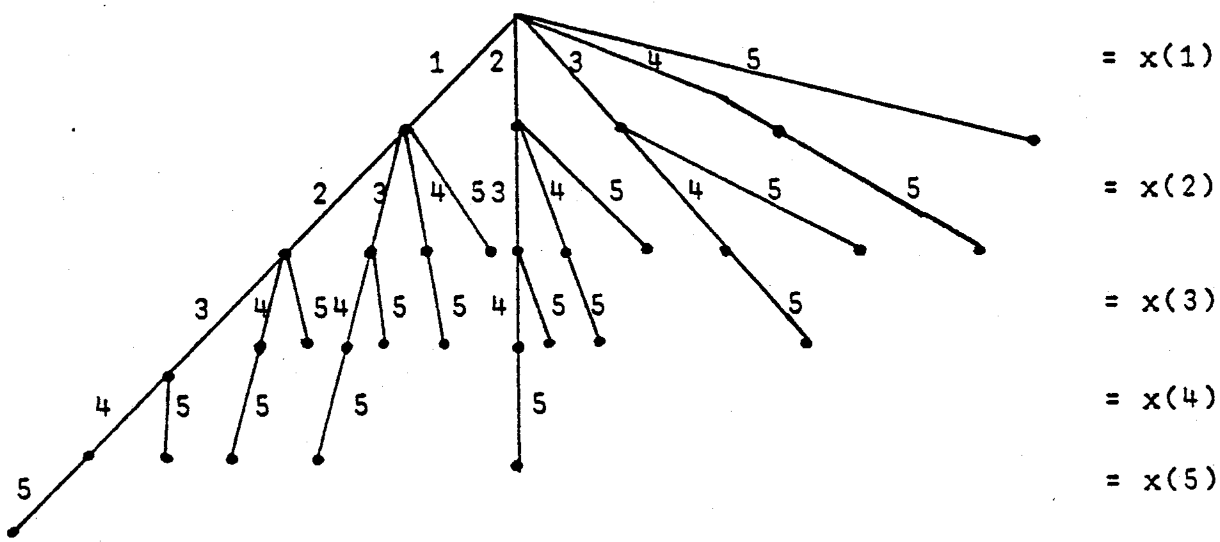


Fig. 4.5

The numbers along the arcs of the branching denote the value that is given to x(1), x(2),...,x(5) respectively. As can be easily seen out of Fig.4.5., each vertex in Fig.4.5. represents one element of the set of all subset of I, namely P. The algorithm now always branches along the most left arc in the tree of Fig.4.5. (and also in general). In any vertex two cases are possible. Either an arc can be included into the set $O \varepsilon P$, such that (4.25) still holds, in which case a maximal set has not been found and branching is done along the number of the arc that is included in O (see Step 3). Or no such arc could be found, in which case a maximal set already has been found (see Step 6). If we reached a final vertex in Fig.4.5. for which (4.25) still holds, then a maximal set has also been found (see Step 3). If a maximal set has been found, then backtracking is performed to the predecessor vertex of the actual one (see Step 4 and 7), thus cutting off all succeeding vertices. If the arc furthest to the right has been reached (which is indicated by x(1)=n), all branches have been examined and the algorithm stops.

Of course the presented method not only works for railway systems, but also for other transportation systems where congestion cannot occur because a time scheduling is made for the travelling vehicles. Thus it applies also to urban underground railway systems and to tram networks, but it does not apply to car traffic on roads. The algorithm can also be used to find a completely new network, which is nowadays especially important for urban underground railway systems. The algorithm remains unchanged, only in Step 2 it has to be proven, if the actual set of arcs O together with the set of vertices X define a strongly connected graph $G^O=(X,O)$. If not, then Step 3 has to be performed and if x(k)=n, then no feasible set of arcs has been found in this case. To check if the graph is strongly connected, a necessary condition is that all vertices have a degree greater than zero. If this is true, then a simple algorithm finds out if the network is connected.

Algorithm to determine if a network is strongly connected:

Step_1:

For a given network $G=(X,A)$ define a matrix $R=(r_{ij})$ as follows

$$r_{ij} = \begin{cases} 1 & \text{if } \operatorname{arc}(i,j) \in A \\ 0 & \text{if } \operatorname{arc}(i,j) \notin A \end{cases}$$

Step_2:

Compute for $p \leq n$, where n is the number of vertices in X

$$B = R + R^2 + R^3 + \ldots + R^p \quad ,$$

where "+" is the addition in the Boolean sense (i.e. 1+1=1).

If for some $p \leq n$ the elements of the matrix $B=(b_{ij})$ are all equal to 1 ($b_{ij}=1$, $i,j=1,\ldots,n$) then the given network $G=(X,A)$ is connected, otherwise not.

```
C ... *** PROGRAM FOR FINDING THE OPTIMAL
C ... *** EXPANSION OF A RAILWAY SYSTEM
C ... ***
C
C ... INPUT
C
C ... N       NUMBER OF VERTICES
C ... C(L)    TRANSPORTATION TIME IN THE ORIGINAL NETWORK
C             FROM VERTEX I=(L-1)/N+1 TO VERTEX
C             J=L-((I-1)*N), L=1,..,N*N. IF C(L)=0, THEN
C             NO ARC EXISTS.
C ... NN      MAXIMUM NUMBER OF ARCS TO BE CONSTRUCTED
C ... Y(M)    DENOTES THAT A NEW ARC CAN BE CONSTRUCTED
C             FROM VERTEX I=(Y(M)-1)/N+1 TO J=Y(M)-((I-1)*N),
C             FOR M=1,..,NN
C ... Q(M)    CONSTRUCTION COST FOR ARC Y(M)
C ... CC(M)   TRANSPORTATION TIME ON ARC Y(M)
C ... B       TOTAL AVAILABLE INVESTMENT BUDGET
C ... T(L)    NUMBER OF PEOPLE WHO WANT TO TRAVEL FROM VERTEX I
C             (SEE ABOVE) TO VERTEX J (SEE ABOVE) - TRIP MATRIX
C
C ... OUTPUT
C
C ... KK      OPTIMAL NUMBER OF ARCS TO BE CONSTRUCTED
C ... YY(M)   ARC TO BE CONSTRUCTED - DEFINED LIKE Y(M) -
C             FOR M=1,..,KK
C ... IMPR    OPTIMAL TRANSP. TIME/ORIGINAL TRANSP. TIME
C
      SUBROUTINE OPRAIL(N,C,NN,Y,Q,CC,B,T,KK,YY,IMPR)
      INTEGER N,NN,Y(1),T(1),KK,YY(1),X(30),ID(900),IG(900),STEP
      REAL C(1),Q(1),CC(1),B,IMPR
      LOGICAL LOG
C
C ... INITIALIZATION
C
      M1=N*N
      DO 5 I=1,M1
5     IG(I)=C(I)
      CALL SPII(N,IG,ID,LOG)
      ORG=0
      DO 10 I=1,M1
10    ORG=ORG+IG(I)*T(I)
      OPT=ORG
      KK=0
      IMPR=OPT/ORG
      K=1
      DO 15 I=1,NN
15    X(I)=0
      X(K)=1
      STEP=1
C
C ... FINDING NEXT MAXIMAL SET
C
20    LOG=.FALSE.
      CALL MAXSET(K,X,B,NN,Q,LOG,STEP)
      IF(LOG) RETURN
```

```
      DO 25 I=1,M1
25    IG(I)=C(I)
      DO 30 I=1,K
      J=X(I)
      J1=Y(J)
30    IG(J1)=CC(J)
      CALL SPII(N,IG,ID,LOG)
      AA=0
      DO 35 I=1,M1
35    AA=AA+IG(I)*T(I)
      IF(AA .GE. OPT) GO TO 20
      OPT=AA
      KK=K
      IMPR=OPT/ORG
      DO 40 I=1,KK
      J=X(I)
40    YY(I)=Y(J)
      GO TO 20
      END
```

```
C ... *** PROGRAM FOR FINDING THE NEXT MAXIMAL SET
C ... ***
C
C ... INPUT
C
C ... K       NUMBER OF ARCS IN THE MAXIMAL SET
C ... X(I)    ARC NUMBER IN THE MAXIMAL SET, I=1,..,K
C             IT HOLDS THAT X(I)<X(I+1) FOR ALL I.
C ... B       TOTAL AVAILABLE INVESTMENT BUDGET
C ... N       MAXIMUM NUMBER OF ARCS TO BE CONSTRUCTED
C ... Q(I)    CONSTRUCTION COST FOR ARC I, I=1,..,N
C ... STEP    DENOTES THE LABEL WHERE PROGRAM SHALL START
C
C ... OUTPUT
C
C ... K       NEW NUMBER OF ARCS IN THE MAXIMAL SET
C ... X(I)    NEW ARC NUMBERS IN THE MAXIMAL SET, I=1,..,K
C ... LOG     IF LOG=TRUE, ALL MAXIMAL SETS HAVE BEEN FOUND
C
       SUBROUTINE MAXSET(K,X,B,N,Q,LOG,STEP)
       INTEGER K,X(1),N,STEP
       REAL B,Q(1)
       LOGICAL LOG
       GO TO (2,35,7),STEP
35     X(K)=X(K)+1
       GO TO 4
C
C ... STEP 2
C
2      E=0
       DO 10 I=1,K
       J=X(I)
10     E=E+Q(J)
       IF(E .LE. B) GO TO 3
       GO TO 6
C
C ... STEP 3
C
3      IF(X(K) .LT. N) GO TO 15
       STEP=2
       RETURN
15     K=K+1
       X(K)=X(K-1)+1
       GO TO 2
C
C ... STEP 4
C
4      IF(K .GT. 1) GO TO 20
       LOG=.TRUE.
       RETURN
20     K=K-1
       X(K)=X(K)+1
C
C ... STEP 5
C
       JJ=X(K+1)-X(K)
```

```
      IF(JJ .LE. 1) GO TO 4
      GO TO 2
C
C ... STEP 6
C
6     IF(X(K) .EQ. N) GO TO 25
      X(K)=X(K)+1
      GO TO 2
25    IF(K .GT. 1) GO TO 30
      LOG=.TRUE.
      RETURN
30    STEP=3
      K=K-1
      RETURN
C
C ... STEP 7
C
7     X(K)=X(K)+1
      GO TO 2
      END
```

4.6. Optimal expansion of a road network

Although we are considering the same problem for a road network than we were discussing for a railway system in the last chapter ,the method for solving this problem will have to be quite a different one. The reason for needing another solution method is the way traffic is assigned to a network. Although we assume in the last chapter that arcs have unlimited capacity and travel costs are constant leading to route choices along the shortest paths in the network, it is no longer true for car traffic , as we already dicussed in chapter 3.3. In fact, it is much more realistic to assume that travel cost along an arc is an increasing function of the travel flow and that car drivers behave according to Wardrop's first principle (descriptive assignment). Because of these assumptions for modelling road traffic, the useful result of chapter 4.5 given in formula (4.27), stating that the construction of an additional arc in the network does not increase the total travel time (costs) over all travelling persons, does not remain true any more. In fact, we are confronted with the rather paradoxical situation that a new road can even increase total travel time. A rather famous example for such a situation is the so-called Paradox of Braess, which we are presenting now to illustrate what we just stated.

Let us consider a simple network as given in Fig.4.6.

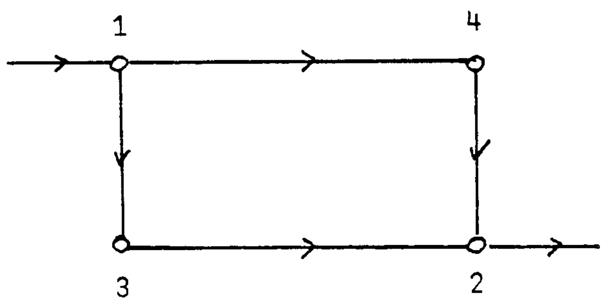


Fig.4.6.

If the flows along the arcs are called x_{13}, x_{14}, x_{32} and x_{42} respectively, we assume the travel costs (time) which are increasing functions of the flow, to be

$$
\begin{aligned}
c_{13} &= 10x_{13} \\
c_{14} &= 50 + x_{14} \\
c_{32} &= 50 + x_{32} \\
c_{42} &= 10x_{42} \quad .
\end{aligned}
\qquad (4.30)
$$

We further assume that there exists a flow of 6 units from vertex 1 to 2. The descriptive assignment of the units to the network will therefore result in 3 units using the path 1-3-2 and 3 units using the path 1-4-2, thus leading to total travel costs of

$$3.(10.3+50+3) + 3.(10.3+50+3) = 498 \qquad (4.31)$$

The network of Fig. 4.6. is now expanded by another arc as given in Fig.4.7. The traveling cost

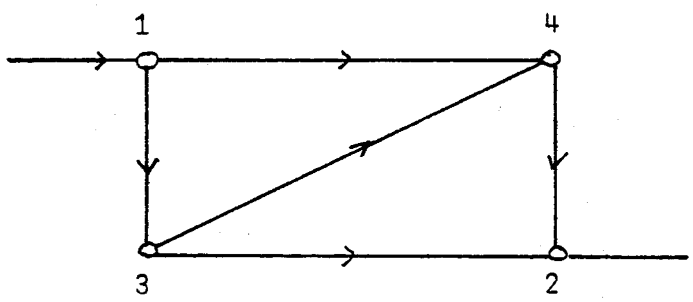


Fig. 4.7.

along the new arc (3,4) is assumed to be

$$c_{34} = 10 + x_{34} \ . \qquad (4.32)$$

When solving the descriptive assignment problem, we get 2 units using path 1-4-2, 2 units using path 1-3-2 and two units using path 1-3-4-2. The user's costs for each unit however are now 92 (instead of 83 for the original network); the total travel costs therefore 552. So the addition of an arc means an increase of user's costs of about 11 per cent.

In the last situation the assignment with 3 units on path 1-3-2 and 3 units on path 1-4-2 is not a solution of the assignment problem according to Wardrop's first principle. For in that case, it would be better for a unit on 1-3-2 with user's costs 83, to use path 1-3-4-2 with user's costs 81.

A similar example to the one given, which can be found in Steenbrink (1974), is stated in Leblanc (1975). Quite a lot of algorithms have been proposed to solve the problem of road network investment and a very good review of those algorithms is presented in Steenbrink (1974). But all of them do not consider a situation like the Paradox of Brass that could occur . Therefore these algorithms do not seem to be of great validity for real situations, although Steenbrink reports on an application of his algorithm to the Dutch road network. The only algorithm so far presented to consider Braess' Paradoxon was published by Leblanc (1975), which is a branch-and-bound algorithm. We shall state his algorithm in the following.

For a fixed set of vertices X and a fixed trip matrix between all these vertices, let us denote by N(A) the optimal value of the objective of the normative assignment as given in (3.27) for a set of arcs A with associated flow costs. For the same set of vertices X and trip matrix, let us denote by D(A) the optimal value of the objective of the descriptive assignment as given in (3.29) for a set of arcs A. Both N(A) and D(A) can be computed with a traffic assignment algorithm, which we discussed in detail in chapter 3.3.

Like in the last chapter, let I denote the set of all arcs j (with associated construction costs q_i and flow costs) that are considered to be included into the already existing network G=(X,A).

For the purpose of the following algorithm, we divide I into 3 subsets O,P,Q, where

$$\begin{aligned} I = {} & O \cup P \cup Q \\ & O \cap P = \emptyset \\ & P \cap Q = \emptyset \\ & O \cap Q = \emptyset \end{aligned}$$

O denotes the set of all arcs that are constructed. P denotes the set of arcs that are not constructed. Q denotes the set of arcs for which a decision has not yet been made. Let B denote the total budget available for road investment.

Because normative assignment always leads to better results (in terms of transportation costs) than descriptive assignment, and because for the normative assignment the objective will not increase if a new arc is built (this is equivalent to (4.27)), the following holds

$$D(A) \geq N(A) \geq N(A \cup O) \qquad (4.33)$$

for any set of arcs A and additional set of arcs O. We shall need (4.33) for computing the lower bound of the objective in each node of the branching tree of the following algorithm.

<u>Algorithm for finding an optimal road network</u>

<u>Step 1</u> (Initialization):

Set $O = P = \emptyset$ and $Q = I$.
Set $F = N(A \cup Q)$ and $M = \infty$.
Set value of the origin node $V(O,P,Q) = N(A \cup Q)$

Step_2:

Choose an arc $j \varepsilon Q$.

a) Set $Q_1 = Q - j$

$O_1 = O \cup j$.

Compute, if

$$\sum_{j \varepsilon O_1} q_j \leqslant B . \qquad (4.34)$$

If (4.34) is not true, set the value associated to the node $V(O_1, P, Q_1) = \infty$ and go to Step 2b.

If Q_1 is empty, compute $D(A \cup Q_1)$ and set

$$M = \min (M, D(A \cup O_1)). \qquad (4.35)$$

If Q_1 is not empty, set the value associated to (O_1, Q_1, P)

$$V(O_1, P, Q_1) = V(O, P, Q) . \qquad (4.36)$$

b) Set $P_1 = P \cup j$.

If Q_1 is empty, compute $D(A \cup O)$ and set

$$M = \min(M, D(A \cup O)) \qquad (4.37)$$

If Q_1 is not empty, set

$$V(O, P_1, Q_1) = N(A \cup O \cup Q_1) \qquad (4.38)$$

Set $V(O,P,Q) = \infty$.

Step_3:

Find a node (among those already analyzed), characterized by the set (O,P,Q), such that

$$Q \neq \emptyset$$
$$V(O,P,Q) < M . \qquad (4.39)$$

If no such node exists, the set of arcs O associated with the actual value of M is the optimal one to build. Stop.

If such nodes exist , find the one with the minimum value $V(.)$. Fix the associated set (O,P,Q) and go to Step 2.

```
C ... *** PROGRAM FOR FINDING AN OPTIMAL ROAD NETWORK
C ... ***
C
C ... INPUT
C
C ... N         NUMBER OF VERTICES (STREET INTERSECTIONS, TOWNS)
C ... NA        NUMBER OF ARCS IN THE ORIGINAL NETWORK
C ... MA        NUMBER OF COEFFICIENTS PER ARC FLOW COST (TRAVEL TIME)
C ... C(L)      IF L=(I-1)*(MA+1)+1, THEN J=(C(L)-1)/N+1 DENOTES THE
C               ORIGIN VERTEX AND K=C(L)-((J-1)*N) THE DESTINATION
C               VERTEX OF THE ARC, WHOSE TRAVEL COST IS GIVEN BY
C               C(L+1)+C(L+2)*F+C(L+3)*F**2+...+C(L+MA)*F**(MA-1),
C               WHERE F IS THE FLOW ON ARC(J,K) AND L=1,...,NA*(MA+1)
C ... T(L)      NUMBER OF PEOPLE WHO WANT TO TRAVEL FROM VERTEX
C               I=(L-1)/N+1 TO VERTEX J=L-(I-1)*N, L=1,..,N*N
C ... B         TOTAL AVAILABLE INVESTMENT BUDGET
C ... NN        NUMBER OF ROADS(ARCS) TO BE CONSIDERED FOR CONSTRUCTION
C ... CC(L)     DEFINED LIKE C(L) FOR THE NEW ROADS, L=1,...,NN*(MA+1)
C ... Q(I)      CONSTRUCTION COST FOR THE ROAD DEFINED BY
C               CC(L), L=(I-1)*(MA+1)+1 AND I=1,...,NN
C
C ... OUTPUT
C
C ... KK        OPTIMAL NUMBER OF ARCS (ROADS) TO BE CONSTRUCTED
C ... YY(M)     ARC(I,J) TO BE CONSTRUCTED, L=YY(M) AND I,J
C               DEFINED AT T(L)
C ... IMPR      OPTIMAL TRANSP. TIME/ORIGINAL TRANSP.TIME
C
      SUBROUTINE OPROAD(N,NA,MA,C,T,B,NN,CC,Q,KK,YY,IMPR)
      INTEGER N,NA,MA,T(1),NN,KK,YY(1)
      REAL B,Q(1),IMPR,C(1),CC(1)
      INTEGER F(14,14,14,14),FL(400),O(100,15),QQ(100,15),P(15)
      REAL V(100)
      LOGICAL LOG
C
C ... STEP 1 (INITIALIZATION)
C
      KNOT=1
      O(KNOT,1)=0
      QQ(KNOT,1)=NN
      P(1)=NN
      DO 5 I=1,NN
      P(I+1)=I
5     QQ(KNOT,I+1)=I
      VM=2.**30.
      LOG=.FALSE.
      CALL WERT(N,NA,MA,C,T,CC,P,F,FL,LOG,KC)
      V(KNOT)=KC
      LOG=.TRUE.
      P(1)=0
      CALL WERT(N,NA,MA,C,T,CC,P,F,FL,LOG,KC)
      ORG=KC
C
C ... STEP 2
C
C ...  A)
```

```
C
2       KNOT1=KNOT
        V(KNOT+1)=V(KNOT)
        V(KNOT)=2.**30.+1.
        QQ(KNOT+1,1)=QQ(KNOT,1)-1
        O(KNOT+1,1)=O(KNOT,1)+1
        I1=O(KNOT+1,1)+1
        I2=QQ(KNOT,1)+1
        DO 20 I=2,I2
20      QQ(KNOT+1,I)=QQ(KNOT,I)
        I3=O(KNOT,1)+1
        DO 25 I=2,I3
25      O(KNOT+1,I)=O(KNOT,I)
        O(KNOT+1,I1)=QQ(KNOT,I2)
        E=0
        DO 10 I=2,I1
        J=O(KNOT+1,I)
10      E=E+Q(J)
        IF(E .GT. B) GO TO 15
        IF(QQ(KNOT+1,1) .LE. 0) GO TO 30
        KNOT=KNOT+1
        GO TO 15
30      LOG=.TRUE.
        I1=O(KNOT+1,1)+1
        DO 35 I=1,I1
35      P(I)=O(KNOT+1,I)
        CALL WERT(N,NA,MA,C,T,CC,P,F,FL,LOG,KC)
        V(KNOT+1)=KC
        IF(VM .LE. V(KNOT+1)) GO TO 15
        KNOT=KNOT+1
        VM=V(KNOT)
C
C ... STEP 2 B)
C
15      I1=O(KNOT1,1)+1
        DO 40 I=1,I1
40      O(KNOT+1,I)=O(KNOT1,I)
        I1=QQ(KNOT1,1)
        DO 45 I=2,I1
45      QQ(KNOT+1,I)=QQ(KNOT1,I)
        QQ(KNOT+1,1)=I1-1
        IF(QQ(KNOT+1,1) .LE. 0) GO TO 50
        LOG=.FALSE.
        P(1)=QQ(KNOT+1,1)+O(KNOT+1,1)
        DO 55 I=2,I1
55      P(I)=QQ(KNOT+1,I)
        I2=O(KNOT+1,1)+1
        DO 60 I=2,I2
        J=I1+I-1
60      P(J)=O(KNOT+1,I)
        CALL WERT(N,NA,MA,C,T,CC,P,F,FL,LOG,KC)
        V(KNOT+1)=KC
        IF(VM .LE. V(KNOT+1)) GO TO 65
        KNOT=KNOT+1
        GO TO 65
50      LOG=.TRUE.
        I1=O(KNOT+1,1)+1
```

```
      DO 70 I=1,I1
70    P(I)=O(KNOT+1,I)
      CALL WERT(N,NA,MA,C,T,CC,P,F,FL,LOG,KC)
      V(KNOT+1)=KC
      IF(VM .LE. V(KNOT+1)) GO TO 65
      KNOT=KNOT+1
      VM=V(KNOT)
C
C ... STEP 3
C
65    VVM=VM
      DO 75 I=1,KNOT
      IF(V(I) .GT. VVM) GO TO 75
      IF(QQ(I,1).GT.0 .AND. V(I).EQ.VVM) GO TO 75
      VVM=V(I)
      IKNOT=I
75    CONTINUE
      IF(VVM .GE. VM) GO TO 80
      JKNOT=KNOT1
      L1=KNOT-1
      IF(IKNOT .NE. L1) KNOT=KNOT-1
      V(JKNOT)=V(KNOT)
      I1=O(KNOT,1)+1
      DO 95 I=1,I1
95    O(JKNOT,I)=O(KNOT,I)
      I1=QQ(KNOT,1)+1
      DO 100 I=1,I1
100   QQ(JKNOT,I)=QQ(KNOT,I)
      IF(IKNOT .NE. L1) GO TO 90
      KNOT=KNOT-1
      GO TO 2
90    V(KNOT)=V(IKNOT)
      I1=O(IKNOT,1)+1
      DO 105 I=1,I1
105   O(KNOT,I)=O(IKNOT,I)
      I1=QQ(IKNOT,1)+1
      DO 110 I=1,I1
110   QQ(KNOT,I)=QQ(IKNOT,I)
      V(IKNOT)=V(KNOT+1)
      I1=O(KNOT+1,1)+1
      DO 115 I=1,I1
115   O(IKNOT,I)=O(KNOT+1,I)
      I1=QQ(KNOT+1,1)+1
      DO 120 I=1,I1
120   QQ(IKNOT,I)=QQ(KNOT+1,I)
      GO TO 2
80    IMPR=VM/ORG
      KK=O(IKNOT,1)
      DO 125 I=1,KK
      J=O(IKNOT,I+1)
      L=(J-1)*(MA+1)+1
125   YY(I)=CC(L)
      RETURN
      END
```

```
C ... *** FUNCTION FOR COMPUTING THE TOTAL TRANSPORTATION TIME
C ... *** FOR DESCRIPTIVE OR NORMATIVE ASSIGNMENT
C
C
C ... INPUT
C
C ... THE PARAMETERS N,NA,MA,C,T,CC SEE UNDER SUBROUTINE OPROAD
C ... FL(L)        FLOW ON ARC WITH NUMBER L, L=1,N*N
C ... F(I,J,K,L) FLOW FROM VERTEX K TO L ON ARC(I,J)
C ... LOG          IF LOG=FALSE, A NORMATIVE ASSIGNMENT IS FOUND
C                  IF LOG=TRUE, A DESCRIPTIVE ASSIGNMENT IS FOUND
C ... P(L)         P(1) DENOTES THE NUMBER OF NEW ROADS TO BE
C                  CONSIDERED - P(I), I=1,...,P(1), DEFINE THE
C                  ARC NUMBERS VIA CC(L), L=(P(I)-1)*(MA+1)+1
C
C ... OUTPUT
C
C ... KC           TOTAL TRANSPORTATION TIME
C
      SUBROUTINE WERT(N,NA,MA,C,T,CC,P,F,FL,LOG,KC)
      INTEGER N,NA,MA,T(1),F(14,14,14,14),FL(1),P(1),KC,E(400)
      REAL C(1),CC(1),CN(1000),CX(1000)
      LOGICAL LOG
      NAN=NA+P(1)
      DO 5 I=1,NA
      J=(I-1)*(MA+1)+1
      CN(J)=C(J)
      CX(J)=C(J)
      DO 10 I1=1,MA
      J=J+1
      CN(J)=C(J)
      CX(J)=C(J)
      IF(.NOT.LOG) GO TO 10
      CN(J)=CN(J)/FLOAT(I1)
10    CONTINUE
5     CONTINUE
      J1=NA*(MA+1)
      DO 15 I=1,P(1)
      J=(P(I+1)-1)*(MA+1)+1
      J1=J1+1
      CN(J1)=CC(J)
      CX(J1)=CC(J)
      DO 20 I1=1,MA
      J=J+1
      J1=J1+1
      CN(J1)=CC(J)
      CX(J1)=CC(J)
      IF(.NOT.LOG) GO TO 20
      CN(J1)=CN(J1)/FLOAT(I1)
20    CONTINUE
15    CONTINUE
      CALL TRAFAS(N,CN,NAN,T,MA,KC,F,FL)
      IF(.NOT. LOG) RETURN
      CALL COST(N,NAN,MA,CX,F,E,FL)
      KC=0
      M=N*N
```

```
      DO 25 I=1,M
25    KC=KC+E(I)*FL(I)
      RETURN
      END
```

The idea of the algorithm is very straight forward. The normative assignment is used as a lower bound for the descriptive assignment because of (4.33). In Step 3 we always search for the node with the lowest bound. Branching from a node means that an arc in Q not yet decided upon is being built (Step 2a) or not built (Step 2b). If it is built, then the budget constraint must hold (4.34). If there are no arcs left in Q, then the descriptive assignment can be computed, thus giving an upper bound (4.39) for the allowed lower bounds. The new lower bound V(.) need not be computed because it is exactly the one of its predecessor (4.36), as the lower bound of a node is always found by assuming that all arcs not yet decided upon will be constructed, leading to the lowest bound possible of

$$N(A \cup O \cup Q) \ ,$$

and therefore in Step 2a

$$O \cup Q = O_1 \cup Q_1 \ .$$

If the arc is not built (Step 2b), then the new lower bound has to be computed (4.38).

It is obvious that the given algorithm is very time consuming because computing (4.35), (4.36), (4.37) and (4.38) is very expensive. Thus only for a small set I and/or a small network G=(X,A), this analysis will be possible. For large networks it will therefore be necessary to drop the idea of finding the optimal network according to a descriptive assignment and rather find the optimal network according to a normative assignment for which Steenbrink (1974) proposed a heuristic algorithm that does apply even for very large networks But one must be aware of the fact that this algorithm does not consider the possibility of Braess' Paradoxon and therefore quite invalid results might be possible. Note that in case of the normative assignment, the algorithm for rail network expansion can be applied, because the objective of the normative assignment will be decreasing if an arc is included into the network. Therefore only the maximal sets have to be considered, thus leading to the same algorithm as in the last chapter.

4.7. Exercise

Let G = (X,A) be some road network and T be the trip matrix. Let I = {a,b,c}be the set of arcs that are considered for construction. Let the construction cost q_i be

$$q_a = 1.000$$
$$q_b = 2.000$$
$$q_c = 900$$

and the available budget B be B = 195o

Find the optimal set of arcs to be constructed with the algorithm of Leblanc.

Use the following transportation time of the descriptive - D(O) - and the normative - N(O) - assignment:

$$D(A \cup \{a\}) = 4.000 \text{ hours}$$
$$D(A \cup \{a\} \cup \{c\}) = 4.500 \text{ hours}$$
$$N(A \cup \{a\} \cup \{b\} \cup \{c\}) = 3.200 \text{ hours}$$
$$N(A \cup \{a\} \cup \{c\}) = 3.300 \text{ hours}$$
$$N(A \cup \{b\} \cup \{c\}) = 3.500 \text{ hours}$$
$$N(A \cup \{c\}) = 4.200 \text{ hours}$$

5. Sequential construction of networks under investment constraints

Having now exploited the structure an optimal network should have, this network has to be constructed. Usually the amount of money to build such a network or parts of it - no matter if this network is a transportation network - a pipeline system or a waste water canal system is rather large and cannot be available all at one time. Rather the investment is being made over some years. Besides, the capacity of the construction firms is a limited one and therefore not all parts of the wanted network can be built at the same time. These are the obvious reasons that the time to finish the network construction usually is not less then,say,five years. Because of such a rather long period during which the investments would not show any positive result concerning the objectives we discussed, it is meaningful to ask if not some arcs that are constructed could be finished in less time and already be used before the total network comes in to operation. For example, if a waste water canal system is to be constructed, why cannot at least the water in the already built canals be cleared in the filter plant, or why cannot an arc connecting two railway stations be used before finishing the other arcs? But if the construction of the network is done sequentially (i.e. one arc after the other) according to the,say, annual amount of money available and to the limited capacity of the construction firms, we are confronted with the question which arc to construct first, which second and so on, thus leading to the problem of finding the optimal sequence in constructing arcs of a given network, with which we shall deal in this chapter.

5.1. Sequential construction of a waste water management system

In chapter 4.1. the problem of finding an optimal network for the waste water was discussed. There the total costs had to be minimized under the restriction that all waste water produced in each vertex of the network had to be purified . As we now assume that this optimal network has been found, we want to construct the arcs of the network and the filter plant itself (to each of which the construction costs are associated) in such a sequence that under an annual budget constraint the amount of water that can be already purified during the construction time of the whole network is maximized. For developing the optimization model two important assumptions have to be made, namely:

- The construction time for an arc or the filter-plant is totally defined by the construction cost and the available amount of money, the sum of which is assumed to be a piecewise, nondecreasing linear function of the time as given in Fig.5.1.

- All arcs and the filter plant can be constructed independently of each other.

Both assumptions, of course, are a simplification of reality but seem to be valid enough to produce practicable results, as was shown in a case study by Knecht (1975), who also stated the algorithm which we shall present here. An earlier work on this subject was presented by Cembrovicz (1972).

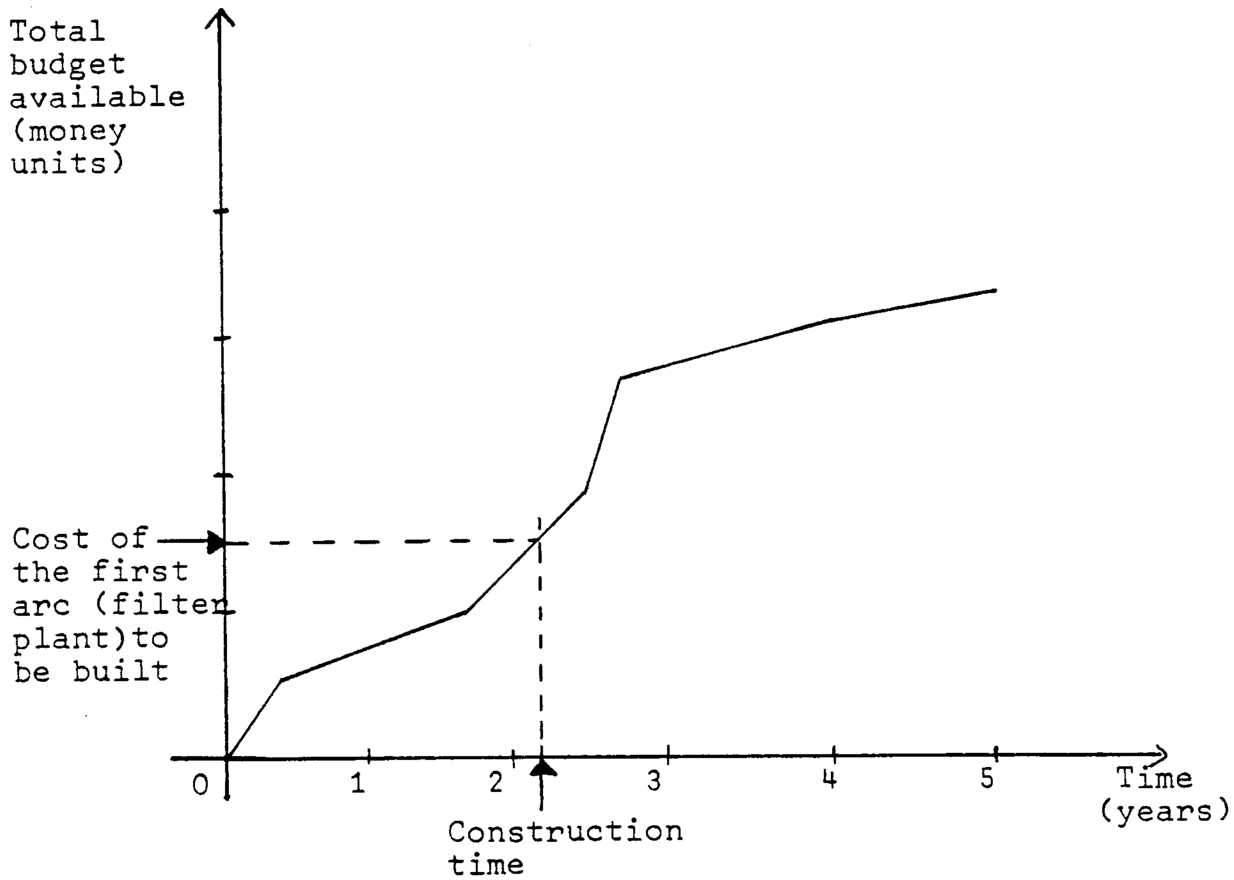


Fig. 5.1.

Because of these two assumptions and in view of the objective of maximizing the total amount of waste water flowing through the filter plant, three quite obvious theorems can be formulated that are necessary to develop the algorithm:

- It is always optimal to construct the filter plant first. This is clear -because no water can be purified before the filter plant is in operation.
- It is always better to construct one arc after the other instead of constructing two or more arcs at the same time. Because the construction time of two arcs is the sum of the construction time of each arc alone (according to the first assumption), building both arcs at the same time will result

in finishing them at the same time t_1, while if they are built one after the other, one of them will be ready at some time t_2, the other one at t_1, where $t_2 < t_1$. Thus water from the first arc can be transported to the filter plant during the time period $t_1 - t_2$, if this arc is constructed before the other while, in the other case, no water of both arcs can be cleaned before t_1.

- It is always better to construct an arc between two vertices x and y, such that there is a path using the already constructed arcs from x or y to the filter plant. If neither from x nor y exists a path to the filter plant, then, although the arc is built, no water can flow from x and y to the filter plant and therefore does not meet the objective. - Because we know from chapter 4.1. that the optimal network will always be a tree with a filter plant at its basis, each vertex (except the basis) has outdegree one (i.e. the number of arcs with this vertex as their origin is one). Therefore, from each vertex there is exactly one path to the filter plant in the optimal network.

A very obvious way of constructing the arcs would be to choose always the arc among the possible ones (such that there is always a path from the newly connected vertex to the filter plant) such that the total amount of water that can be purified in the short run is maximized. But this is not at all optimal as the following example will show. Assume a network as given in Fig.5.2. where the numbers at the vertices denote the amount of waste water produced there and where the construction costs of all arcs are equal and the construction time of an arc is one time unit (according to the budget). Let us further assume that all arcs within

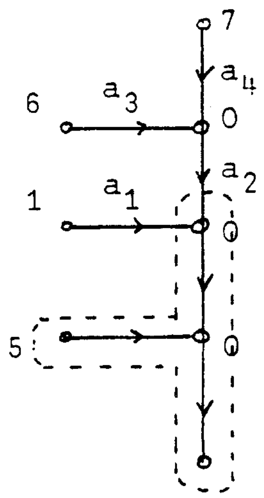


Fig. 5.2

the dotted line are already existing. Then the sequence of constructing the immediatly best arc would be $a_1 - a_2 - a_4 - a_3$, therefore the amount of water flowing to the filter plant wihtin 4 time units being.

$$(1) + (1+0) + (1+0+7) + (1+0+7+6) = 24 .$$

But if we choose the sequence a_2-a_4-a_3-a_1,the amount of water now purified would be

$$(0) + (0+7) + (0+7+6) + (0+7+6+1) = 34 .$$

Let us define the part of construction k(k=1,2,...,n, the total number of all vertices) as being all the constructions necessary to connect a vertex with the already existing part of the network. Besides building the arc that connects this vertex with the rest, this part of construction also includes all other works that are perhaps necessary on this purpose. The time for realizing k under the budget constraint is called t(k), where t(k) can be computed, if the cost of k as well as the starting time for the construction of k is known. The way t(k) is then computed can be seen in Fig.5.1.

Let now P be a feasible sequence $k_1, k_2, \ldots, k_n$ of parts of constructions $k = 1,2,\ldots,n$ and let $g_j^{(P)}$ be the amount of water flowing to the filter plant in the j-th construction period. If the construction sequence P is realized, then our objective can be stated as

$$\max: f = \sum_{j=1}^{n} g_j^{(P)} \cdot t(k_j) \quad . \qquad (5.1)$$

The method chosen for solving this problem is dynamic programming, which is possible, because the objective can be stated recursively and is monotonically non-decreasing. Besides, the problem can be fully described by defining the system variables X_j, $j=1,2,\ldots,n$, denoting the set of vertices that are connected with the filter plant after the j-th part of construction has been finished, and the decision variables y_j,$(j=1,2,\ldots,n)$, which denotes the vertex that is connected with the filter plant in the j-th part of construction. Clearly it holds that

$$X_j = X_{j-1} \cup \{y_j\}, \quad j = 1,2,\ldots,n$$

where $X_o = \emptyset$.

Because we know that the filter plant has to be built first, this means, that

$$y_1 = 1$$

$$X_1 = \{1\}$$

if we denote the basis of the tree by 1. The idea of the dynamic programming algorithm is to start by assuming that all vertices have been connected. Then going one step further, one assumes that two vertices are still left for connection.

Then for all possible realized tree configurations X_{n-2}, the optimal sequence of connecting the last two vertices, y_{n-1} and y_n, is computed. The next step is performed by assuming that there are still 3 vertices left and the last two vertices are connected according to the best solution found for each configuration in the step before. Continuing in this way, one finally reaches the state of the system, where only one vertex (the filter plant) is connected, thus the optimal sequence can be computed. This verbally stated algorithm can be stated as follows:

Let us define

$g(X)$ ... amount of water that can flow to the filter plant per time unit, given the set of connected vertices X.

X_{j-1} ... set of all connected vertices at the beginning of the j-th part of construction (systems variable).

y_j ... number of vertex to be connected in the j-th part of construction (decision variable) - therefore $X_j = X_{j-1} \cup \{y_j\}$.

$f(X_{j-1})$ ... maximum amount of water flowing to the filter plant from the beginning of the j-th part of construction until the end of the n-th part of construction (the whole network), given the systems variable X_{j-1} - of course $f(X_n) = 0$ and the objective as stated in (5.1) can be written as $f = f(X_o) = f(\emptyset)$.

$C(X_{j-1})$ ... set of all predecessor-vertices of the vertices in X_{j-1}.

n ... total number of vertices in the given tree.

$q(y_j)$ .. costs to construct vertex y_j.

$t(b)$ ... total time necessary to construct a subtree with costs b.

Algorithm for finding the optimal construction sequence

Step 1 (Initialization):

Set $j=n$, $X_n=\{y_1,\ldots,y_n\}$ and $f(X_n)=0$.

Step 2: Compute

$$f(X_{j-1}) = \max_{y_j \varepsilon C(X_{j-1})} \Big[f(X_{j-1} \cup \{y_j\}) + g(X_{j-1}).(t(\sum_{y_1 \varepsilon X_{j-1} \cup \{y_j\}} q(y_1)) - t(\sum_{y_1' \varepsilon X_{j-1}} q(y_1')))\Big] \tag{5.2}$$

for all possible sets of vertices X_{j-1} , such that X_{j-1} defines a connected subtree of the given tree X_n and the filter plant, at vertex $y_1=1$, belongs to X_{j-1}, $1 \varepsilon X_{j-1}$. Note that $f(X_{j-1} \cup \{y_j\})$ is already known because $X_j = X_{j-1} \cup \{y_j\}$. Set $j=j-1$ and go to Step 3.

Step 3 (Termination):

If $j \geqslant 2$ go to Step 2. Note that $X_o=\emptyset$ and $X_1=\{1\}$.

If $j = 1$ the optimal solution has been found and is given as $f = f(X_o) = f(X_1)$.

The optimal sequence $y_1(=1) - y_2 - \ldots - y_n$ can be found out of (5.2) recursively. The optimal vertex y_2 is the one for which (5.2) was maximum if $X_1 = \{y_1\}$, knowing now $X_2 = \{y_1,y_2\}$, the optimal vertex y_3 can be found and so on.

The main problem of the algorithm is the amount of X_j's to be analyzed and stored at each step, although this number is substantially reduced due to the fact that the network has to be connected all the time and that we know that the first vertex to be constructed is the filter plant itself. Thus, this algorithm will only apply to smaller networks. Knecht (1975) reports on an application to a network with 16 vertices, where, as he says, the computational limit nearly has been reached. But, of course, the computation time depends strongly on the type of tree, for which a solution has to be found. For example, if, in the extreme case, the set $C(X_{j-1})$ always contains only one element - a tree of this type is drawn in Fig.4.4.a - then the computation will be very easy. In contrary, if $C(X_{j-1})$ contains all vertices that are not member of X_{j-1} - an example is shown in Fig.4.4.b - then the computation will be difficult.

```
C ... *** PROGRAM FOR FINDING THE OPTIMAL CONSTRUCTION SEQUENCE
C ... *** OF A WASTE WATER CANAL-TREE-NETWORK
C ... ***
C
C ... INPUT
C
C ... N        NUMBER OF VERTICES IN THE TREE
C ... NSU(I)   SUCCESSOR VERTEX OF VERTEX I IN THE TREE (I<NSU(I))
C              THE FILTER PLANT IS LOCATED AT VERTEX N
C ... WA(J)    WASTE WATER PRODUCED AT VERTEX J PER YEAR
C ... IP       NUMBER OF PERIODS WITH DIFFERENT INVESTMENT BUDGETS
C ... TIP      LENGTH OF PERIODS IN YEARS (EACH PERIOD HAS SAME LENGTH)
C ... BB(L)    INVESTMENT BUDGET AVAILABLE UNTIL THE END OF PERIOD L
C ... Q(J)     COSTS TO CONSTRUCT CONNECTION OF VERTEX J
C
C ... OUTPUT
C
C ... OP       MAXIMUM AMOUNT OF WATER TO BE CLEANED DURING
C              CONSTRUCTION TIME
C ... YY(I)    OPTIMAL SEQUENCE OF CONNECTING THE VERTICES, WHERE
C              YY(1) DENOTES THE FIRST VERTEX TO BE CONNECTED AND
C              YY(N) DENOTES THE LAST ONE
      SUBROUTINE OPCOSE(N,NSU,WA,IP,TIP,BB,Q,OP,YY)
      INTEGER N,NSU(1),IP,YY(1)
      REAL WA(1),TIP,BB(1),Q(1),OP
      LOGICAL LOG
      INTEGER IX(1000),MX(35),MY(35),JF(1000,34)
      REAL F(1000)
C
C ... STEP 1 (INITIALIZATION)
C
      KK1=1
      KK3=KK1+1
      KK2=KK1
      K=N
      DO 5 I=1,N
5     MX(I)=I
      CALL CODE(K,MX,II)
      IX(KK1)=II
      F(KK1)=0.
C
C ... STEP 2
C
2     DO 10 I=1,KK1
      II=IX(I)
      CALL DECODE(II,K,N,MX)
      K1=K-1
      DO 15 J=1,K1
      L1=0
      DO 20 L=1,K1
      L1=L1+1
      IF(L .EQ. J) L1=L1+1
20    MY(L)=MX(L1)
      CALL CONN(N,NSU,K1,MY,LOG)
      IF(.NOT. LOG) GO TO 15
      CALL CODE(K1,MY,II)
```

```
      IF(KK3 .GT. KK2) GO TO 25
      DO 30 M=KK3,KK2
      IF(IX(M) .NE. II) GO TO 30
      V=F(I)+G(K1,MY,WA)*(T(K,MX,Q,IP,BB,TIP)-T(K1,MY,Q,IP,BB,TIP))
      IF(V .LE. F(M)) GO TO 15
      F(M)=V
      LX=N-K
      IF(K .EQ. N) GO TO 14
      DO 16 LY=1,LX
16    JF(M,LY)=JF(I,LY)
14    JF(M,LX+1)=MX(J)
      LX1=LX+1
      GO TO 15
30    CONTINUE
25    KK2=KK2+1
      IF(KK2 .LT. 1000) GO TO 29
      PRINT *,' STORAGE IS TOO SMALL'
      RETURN
29    IX(KK2)=II
     F(KK2)=F(I)+G(K1,MY,WA)*(T(K,MX,Q,IP,BB,TIP)-T(K1,MY,Q,IP,BB,TIP))
      LX=N-K
      IF(K .EQ. N) GO TO 24
      DO 26 LY=1,LX
26    JF(KK2,LY)=JF(I,LY)
24    JF(KK2,LX+1)=MX(J)
      LX1=LX+1
15    CONTINUE
10    CONTINUE
      DO 35 I=KK3,KK2
      F(I-KK1)=F(I)
      IX(I-KK1)=IX(I)
      LX=N-K+1
      DO 36 LY=1,LX
36    JF(I-KK1,LY)=JF(I,LY)
35    CONTINUE
      KK1=KK2-KK1
      KK3=KK1+1
      KK2=KK1
      K=K-1
C
C ... STEP 3 (TERMINATION)
C
      IF(K .GE. 2) GO TO 2
      OP=F(1)
      YY(1)=N
      LX=N-1
      J=1
      DO 40 I=1,LX
      J=J+1
40    YY(J)=JF(1,N-I)
      RETURN
      END
```

```
C ... *** PROGRAM FOR TRANSFORMING A SET OF NUMBERS INTO
C ... *** ONE NUMBER
C ... ***
C
C ... INPUT
C
C ... K      NUMBER OF ELEMENTS IN SET MX
C ... MX(I)  ELEMENT I, I=1,..,K, IN SET MX
C
C ... OUTPUT
C
C ... II     CODE NUMBER
C
      SUBROUTINE CODE(K,MX,II)
      INTEGER K,MX(1),II
      II=0
      DO 5 I=1,K
5     II=II+2**MX(I)
      RETURN
      END
```

```
C ... *** PROGRAM FOR DECODING AN INTEGER NUMBER INTO A
C ... *** SET OF NUMBERS
C ... ***
C
C ... INPUT
C
C ... II      INTEGER CODE NUMBER
C ... K       NUMBER OF ELEMENTS IN THE SET MX
C ... N       HIGEST VALUE OF AN ELEMENT IN THE SET MX
C
C ... OUTPUT
C
C ... MX(I)  ELEMENT I IN SET MX, I=1,..,K. MX(I)<MX(I+1) FOR ALL I
C
      SUBROUTINE DECODE(II,K,N,MX)
      INTEGER II,K,N,MX(1)
      J1=II
      K1=K
      DO 5 I=1,N
      I1=N-I+1
      J2=J1-2**I1
      IF(J2 .LT. 0) GO TO 5
      MX(K1)=I1
      IF(J2 .EQ. 0) GO TO 10
      K1=K1-1
      J1=J2
5     CONTINUE
10    CONTINUE
      RETURN
      END
```

```
C ... *** PROGRAM WHICH CHECKS IF A SET OF VERTICES MY DEFINES
C ... *** A CONNECTED SUBTREE WITH BASIS N ON THE TREE GIVEN BY NSU
C ... ***
C
C ... INPUT
C
C ... N         NUMBER OF VERTICES IN THE TREE
C ... NSU(I)    SUCCESSOR VERTEX OF VERTEX I, I<NSU(I)
C ... K1        NUMBER OF VERTICES IN SET MY
C ... MY(J)     ELEMENT J IN SET MY
C
C ... OUTPUT
C
C ... LOG       IF LOG=TRUE MY DEFINES A FEASIBLE SUBTREE
C               ELSE MY IS NOT A FEASIBLE SUBTREE
C
      SUBROUTINE CONN(N,NSU,K1,MY,LOG)
      INTEGER N,NSU(1),K1,MY(1)
      LOGICAL LOG
      IF(MY(K1) .EQ. N) GO TO 5
      LOG=.FALSE.
      RETURN
5     K2=K1-1
      IF(K1 .EQ. 1) GO TO 25
      DO 10 I=1,K2
      J=MY(I)
15    J=NSU(J)
      IF(J .EQ. N) GO TO 10
      DO 20 K=1,K1
      IF(J .GT. MY(K)) GO TO 20
      IF(J .EQ. MY(K)) GO TO 15
      LOG=.FALSE.
      RETURN
20    CONTINUE
10    CONTINUE
25    LOG=.TRUE.
      RETURN
      END
```

```
C ... *** FUNCTION FOR COMPUTING THE AMOUNT OF WATER FLOWING
C ... *** TO THE FILTER PLANT PER YEAR FROM THE VERTICES IN
C ... *** THE SET MX
C ... ***
C
C ... INPUT
C
C ... K        NUMBER OF ELEMENTS IN SET MX
C ... MX(I)    ELEMENT I OF SET MX
C ... WA(J)    AMOUNT OF WASTE WATER PRODUCED PER YEAR AT VERTEX J
C
C ... OUTPUT
C
C ... G        AMOUNT OF WASTE WATER PRODUCED BY VERTICES IN MX
C
      FUNCTION G(K,MX,WA)
      INTEGER K,MX(1)
      REAL WA(1),G
      G=0.
      DO 5 I=1,K
      J=MX(I)
5     G=G+WA(J)
      RETURN
      END
```

```
C ... *** FUNCTION FOR COMPUTING THE CONSTRUCTION TIME OF A
C ... *** SUBTREE OF VALUE B
C ... ***
C
C ... INPUT
C
C ... K         NUMBER OF ELEMENTS IN SET MX
C ... MX(I)     ELEMENT I OF SET MX
C ... Q(J)      COSTS FOR CONSTRUCTING CONNECTION OF VERTEX J
C ... IP        NUMBER OF PERIODS WITH DIFFERENT AVAILABLE INVESTMENT
C               BUDGETS - EACH PERIOD HAS THE SAME LENGTH
C ... TIP       LENGTH OF ONE PERIOD IN YEARS
C ... BB(L)     BUDGET AVAILABLE UNTIL THE END OF PERIOD L, L=1...,IP
C ... OUTPUT
C
C ... T         TOTAL CONSTRUCTION TIME FOR THE ELEMENTS IN MX
C
      FUNCTION T(K,MX,Q,IP,BB,TIP)
      INTEGER K,MX(1),IP
      REAL Q(1),BB(1),TIP
      QQ=0.
      DO 5 I=1,K
      J=MX(I)
5     QQ=QQ+Q(J)
      DO 10 I=1,IP
      IF(QQ .GT. BB(I)) GO TO 10
      IF(I .EQ. 1) GO TO 20
      J1=I-1
      GO TO 15
10    CONTINUE
20    T=TIP*QQ/BB(1)
      RETURN
15    T=J1*TIP+TIP*(QQ-BB(J1))/(BB(J1+1)-BB(J1))
      RETURN
      END
```

5.2. Sequential construction of a railway network

We are confronted with quite a similar problem to the one of the last chapter, when dealing with the expansion of a rail network. Assuming that we already know the optimal network by applying the algorithm of chapter 4.5, we now want to know the optimal sequence of construction, such that the total transportation time during the construction period is minimum. Of course, we assume that the trip matrix does not change during this period and that people travel along shortest paths. Although applying the dynamic programming algorithm of the last chapter is possible, this does not lead to an efficient algorithm because no reductions for the set X_j can be made according to the restriction that the network has to be connected and, besides, the objective function is much more difficult to compute than in chapter 5.2., because the shortest paths between all pairs of vertices have to be computed. If we denote by T(k) the time of construction for arc k, k=1,2,...,m, according to the budget constraint, if t_{ij} denotes the number of people travelling from vertex i to j and if p_{ij}^{O} denotes the length of the shortest path from vertex i to j with the set O of newly built arcs, the objective can be written as to find some permutation of the sequence of construction of the arcs $k_1, k_2, \ldots, k_m$ such that

$$\min: \quad F = \sum_{l=1}^{m} \sum_{i,j \in X} t_{ij} \, p_{ij}^{O_{l-1}} T(k_l) \qquad (5.3)$$

where X is the set of all vertices and $O_l = O_{l-1} \cup \{k_l\}$, $O_o = \emptyset$.

Obviously, the computation of $p_{ij}^{O_l}$ is the critical part of the objective. Therefore, instead of optimizing (5.3), we suggest an heuristic algorithm that solves the problem

$$\bar{F} = \sum_{l=1}^{m} \{ \min_{\substack{k_l \varepsilon P-O_{l-1} \\ k_{l+1} \varepsilon P-O_l}} \left[\sum_{i,j \varepsilon X} t_{ij} \, p_{ij}^{O_{l-1}} T(k_l) + t_{ij} \, p_{ij}^{O_l} T(k_{l+1}) \right] \} \tag{5.4}$$

where P is the set of all arcs to be constructed.
(5.4) only gives a suboptimal solution to (5.3).

6. Selection of routes within a given network

Although we were already dealing with route selection in connection with traffic assignment for roads and for trains, we shall devote now a chapter to this problem, discussing more intensively normative route selection problems with which public services are confronted. These problems, as we shall see, are rather different to those we already discussed.

6.1. Street cleaning routes

Given a network of roads, where the arcs respresent streets (possibly, if the network is directed, only one way of the street) and the vertices represent intersections of these streets. Such a network needs to be serviced regularly on many purposes. The most regular service is usually the cleaning of the streets (especially in urban areas), but in winter the snow removal can be even more important. Besides, there exist services that do not deal with the streets directly, but have to use all the streets, for example a postman delivering letters or a truck collecting garbage from the households. All these services have in common that a shortest route (or routes) has to be found such that all arcs of the network are used and that this route is a circuit, meaning that the initial and final vertex of the route is the same. Although the application of this problem is broad, we refer to it as the problem of street cleaning as it has been extensively studied - see Liebling (197o), applying his algorithm to the street cleaning of Zürich, and Beltrami & Bodin (1974) doing the same for New York City. Using the shortest circuit that passes through all arcs at least once, guarantees that the service (street cleaning, snow removal etc.) can be performed with the minimum number of service facilities (trucks) and manpower, thus resulting in cost-minimization. - We shall divide this chapter into two parts. First we will deal with the problem of finding the optimal route without restricting the route length, thus resulting in the well-known Chinese postman problem. Using the solution methods for this problem,we can then attack the more realistic problem where the route length is restricted. Therefore routes have to be found such that again,all arcs belong at least to one of these routes and that the total length of the routes is minimum.

6.1.1. Street cleaning without limited route length - the Chinese postman problem

Given a network G=(X,A) (directed or nondirected), a circuit (path) that passes through all arcs (in the right direction) exactly once is called an Eulerian circuit (path). Obviously not all graphs have Eulerian circuits (or paths), but if such a circuit exists,it means that the graph can be drawn on paper by following this circuit and without lifting the pencil from the paper. The basic theorem on the existence of an Eulerian circuit is as follows:

Theorem: A connected,nondirected graph G contains an Eulerian circuit (path) if,and only if,the number of vertices of odd degree is 0 (0 or 2 for a path).

This condition is of course necessary because any Eulerian circuit must use one link to arrive and a different to leave the vertex,since any link must be traversed exactly once. Hence, if G contains an Eulerian circuit,all vertex degrees must be even. We shall not show the sufficiency of the condition directly, rather we shall present an algorithm to find an Eulerian circuit in a graph with all vertex degrees to be even.

A very similar theorem holds in the case of a directed graph, namely:

Theorem: A connected,directed graph G contains an Eulerian circuit (path) if,and only if,the indegrees $d_t(x_i)$ and the outdegrees $d_o(x_i)$ of the vertices satisfy the condition:

for the case of a circuit: $d_t(x_i) = d_o(x_i)$ for all vertices
for the case of a path (where p is the initial and q the final vertex of the Eulerian path):

$$d_t(x_i) = d_o(x_i) \quad \text{for all } x_i \neq p \text{ or } q$$

$$d_t(q) = d_o(q) + 1$$

and $$d_t(p) = d_o(p) - 1$$

The algorithm for finding such an Eulerian circuit (or path) is based on a very simple idea. Start at any vertex and proceed going along arcs which have not been used yet until the initial vertex is reached again and no unused arc can be found to leave the initial vertex again. This is always possible because each vertex that has an unused arc to reach the vertex also must have an unused arc for leaving (because the degree is even), only the initial vertex has an already used arc for leaving. Having already used all arcs, an Eulerian circuit has been found. If not, one goes back along the just found circuit until a vertex is reached which has not yet used arcs incident to it. Then proceed along such an arc until the starting point of this new circuit is reached again and include this circuit to the other one. Proceed like this until all arcs have been passed exactly once, thus giving the Eulerian circuit. We shall state the algorithm also in a formal way, as we shall need it for the street cleaning problem.

Algorithm for finding an Eulerian circuit

Assumed is a graph $G=(X,A)$, for which the mentioned theorems guarantee that an Eulerian circuit exists.

Step 1:

Choose any vertex $z \in X$ as the initial vertex. Set $x=z$, $F=A$, $y=z$, $K=\emptyset$ and $H=\emptyset$.
Let $\Gamma(v)$ denote the set of all vertices to which there is an arc connection from v. Therefore $(v,\Gamma(v))$ denotes the set of arcs going out from v.

Step 2:

Find the set of arcs

$$(y,\Gamma(y)) \cap F = B \ .$$

If $B=\emptyset$ then go to Step 4.

In the other case choose w, such that $(y,w) \in B$ and, if possible, $w \neq x$

Set

$$F=F-(y,w)$$

$H=[H,(y,w)]$, denoting the sequence of arcs already passed along.

Step 3:

Set $y=w$ and go to Step 2.

Step 4:

Let the last arc in the sequence of H be (u,v), thus

$$H = [\ldots, (u,v)]$$

Delete (u,v) in H and put it in the sequence of arcs K, thus

$$K = [(u,v),K]$$

Step 5:

Check if H is empty. If so, the sequence of arcs in K is an Eulerian circuit. Stop.

If not, set $y=u$ and $x=u$ and go to Step 2.

```
C ... *** PROGRAM FOR FINDING AN EULERIAN CICUIT IN A NONDIRECTED
C ... *** GRAPH
C ... ***
C
C ... INPUT
C
C ... N       NUMBER OF VERTICES IN THE GRAPH
C ... F(L)    NUMBER OF TIMES THE ARC(I,J), L=IND(I,J,N), MAY BE
C             PASSED. F(L)=F(L1), L1=IND(J,I,N) IS ASSUMED
C
C ... OUTPUT
C
C ... NA      NUMBER OF ARCS IN THE GRAPH
C ... KK(I)   DENOTES THE I-TH VERTEX IN THE EULERIAN CIRCUIT
C             KK(1)=1 AND KK(NA+1)=1
C
      SUBROUTINE EULER(N,F,NA,KK)
      INTEGER N,F(1),NA,KK(1),H(900),K(900),U,W,X,Y,KX,HX
C
C ... STEP 1
C
      X=1
      Y=1
      KX=0
      HX=1
      H(1)=1
C
C ... STEP 2
C
2     DO 10 I=1,N
      IF(I .EQ. X) GO TO 10
      L=IND(Y,I,N)
      IF(F(L) .EQ. 0) GO TO 10
      W=I
      GO TO 15
10    CONTINUE
      L=IND(Y,X,N)
      IF(F(L) .EQ. 0) GO TO 4
      W=X
15    F(L)=F(L)-1
      L1=IND(W,Y,N)
      F(L1)=F(L1)-1
      HX=HX+1
      H(HX)=W
C
C ... STEP 3
C
      Y=W
      GO TO 2
C
C ... STEP 4
C
4     IF(KX .GT. 0) GO TO 20
      KX=KX+1
      K(KX)=H(HX)
20    HX=HX-1
```

```
      IF(HX .EQ. 0) GO TO 25
      KX=KX+1
      U=H(HX)
      K(KX)=H(HX)
C
C ... STEP 5
C
      Y=U
      X=U
      GO TO 2
C
C ... TERMINATION
C
25    NA=KX-1
      DO 30 I=1,KX
      J=KX-I+1
30    KK(J)=K(I)
      RETURN
      END
```

Coming back to our problem, we are not interested in finding an Eulerian circuit on a very special graph but on a shortest circuit, using each arc at least once for an arbitrary graph. But the latter problem can now be transformed into the problem of finding an Eulerian graph. Let us assume that the graph on which we want to solve the Chinese postman problem (i.e. find the shortest circuit using each arc at least once) is non-directed (for directed graphs the transformation is quite the same). Of the graph $G=(X,A)$ some vertices will then have even degrees (let this set be X^+) and the other vertices (in the set $X^-=X-X^+$) will have odd degrees. Now the sum of the degrees d_i of all vertices $x_i \varepsilon X$ is equal to twice the number of links in A (since each link adds unity to the degrees of its two end vertices) and is therefore an even number 2m. Hence

$$\sum_{x_i \varepsilon X} d_i = \sum_{x_i \varepsilon X^+} d_i + \sum_{x_i \varepsilon X^-} d_i = 2m$$

and since $\sum_{x_i \varepsilon X^+} d_i$ is even , $\sum_{x_i \varepsilon X^-} d_i$ is also even, which means that the number of vertices in the set X^- (with odd degree) is even. If we now connect arbitrary pairs of vertices with odd degree, doing this for all such vertices with artificial links in the set L, this resulting in the graph $\bar{G} = (X, A \cup L)$, then all vertices in $\bar{G}$ now have even degree and thus an Eulerian circuit on $\bar{G}$ can be found. Now the length of an Eulerian circuit is just the sum of the length of all arcs in $A \cup L$. As A is given originally, therefore L must be chosen in a way such that the sum of the length of arcs in L is minimum. As we do not build in new arcs into the original graph G, the set L consists of arcs or paths in A and therefore L denotes those arcs in A which have to be passed more than once in order to pass all arcs in A at least once. To find the optimal set L ,the shortest paths between all possible pairs of vertices in X^- is computed in the graph $G=(X,A)$. These paths are the shortest possible connections between the vertices with odd degree. Let us denote the length of these shortest paths by p_{ij}, for x_i, $x_j \varepsilon X^-$ and the set of arcs of which each shortest path consists by S_{ij}. Then the problem of finding those pairs of vertices x_i, $x_j \varepsilon X^-$, such that the sum of the p_{ij}'s associated to these pairs is minimum, is a so-called assignment problem which can be solved efficiently with the minimum cost flow algorithm of chapter 3.3.1. On this purpose we have to define the directed network $N=(Y,D)$. The set of vertices consists of the vertices s (where the flow starts) and t (where the flow ends) and twice the set of vertices X^-, (say X_1^- and X_2^-)

$$Y = \{s,t\} \cup X_1^- \cup X_2^- \quad . \tag{6.1}$$

The set of arcs D consists of the arcs

$$\begin{aligned} &(s,x_i) \text{ with capacity of one unit and zero costs,} \\ &\qquad x_i \varepsilon X_1^- \; . \\ &(x_i,x_j) \text{ with capacity of one unit and costs } p_{ij}, \\ &\qquad x_i \varepsilon X_1^-, x_j \varepsilon X_2^- \; , \quad x_i \neq x_j \; . \\ &(x_j,t) \text{ with capacity of one unit and zero costs,} \\ &\qquad x_j \varepsilon X_2^- \; . \end{aligned} \tag{6.2}$$

If we now send a flow of value v=n (the number of vertices in X^-) from s to t with minimum cost, the solution will give the set of pairs between which there is nonzero flow, such that the sum of the associated p_{ij}'s is minimum. If we now form the set of arcs L by the sets of arcs S_{ij} that are associated to the optimal solution of the minimum cost problem and find an Eulerian circuit on $\bar{G}=(X,A\cup L)$, we solved the Chinese postman problem.

Algorithm for solving the Chinese postman problem

Step 1:

Given the nondirected graph $G=(X,A)$. Find the set of vertices with odd degree X^- .

Step 2:

Compute the shortest paths in G between all pairs of vertices in X^-.

Step 3:

Solve the minimum cost flow assignment problem with flow v=n (the number of vertices in X^-) for the network defined in (6.1) and (6.2)

Step 4:

Put all arcs that belong to the shortest path of a pair of vertices (x_i, x_j), $x_i, x_j \varepsilon X^-$, $x_i \neq x_j$ for which a nonzero flow has been found in Step 3 into the set L. Do this for all pairs (x_i, x_j) with nonzero flow.

Step 5:

Find an Eulerian circuit on the graph $\bar{G} = (X, A \cup L)$, thus denoting an optimal street cleaning tour of one vehicle on the graph G=(X,A), where L denotes the set of arcs in A, which have to be passed more than once.

```
C ... *** PROGRAM FOR SOLVING THE CHINESE POSTMAN PROBLEM
C ... ***
C
C ... INPUT
C
C ... N       NUMBER OF VERTICES IN THE GIVEN GRAPH
C ... C(L)    ARC LENGTH FOR ARC(I,J) AND L=IND(I,J,N)
C
C ... OUTPUT
C
C ... LENGTH  TOTAL LENGTH OF THE TOUR
C ... NUMB    NUMBER OF ARCS INCLUDED IN THE TOUR
C ... KK(I)   DENOTES THE I-TH VERTEX IN THE EULERIAN CIRCUIT
C             I=1,...,NUMB+1, KK(1)=KK(NUMB+1)
C
      SUBROUTINE CHIPOS(N,C,LENGTH,NUMB,KK)
      INTEGER N,C(1),LENGTH,NUMB,KK(1)
      INTEGER P(625),F(1600),X(25),D(1600)
      LOGICAL LOG
C
C ... STEP 1
C
      M=N*N
      LENGTH=0
      DO 5 I=1,M
      LENGTH=LENGTH+C(I)
      F(I)=0
      IF(C(I) .NE. 0) F(I)=1
5     CONTINUE
      LENGTH=LENGTH/2
      I3=0
      DO 10 I=1,N
      I2=0
      DO 15 J=1,N
      L1=IND(I,J,N)
15    I2=I2+F(L1)
      I1=I2+1
      I1=I1/2
      I2=I2/2
      IF(I1 .EQ. I2) GO TO 10
      I3=I3+1
      X(I3)=I
10    CONTINUE
      IF(I3 .LE. 1) GO TO 1
C
C ... STEP 2
C
      CALL SPII(N,C,D,LOG)
C
C ... STEP 3
C
      DO 20 I=1,I3
      I1=X(I)
      DO 25 J=1,I3
      I2=X(J)
      L1=IND(I,J,I3)
```

```
      L2=IND(I1,I2,N)
25    P(L1)=C(L2)
20    CONTINUE
      CALL ASGNMT(I3,P,KK,NCOS)
      LENGTH=LENGTH+NCOS
C
C ... STEP 4
C
      DO 30 I=1,I3
      I1=X(I)
      I2=KK(I)
      I2=X(I2)
      J=I2
35    K=J
      J=IND(I1,K,N)
      J=D(J)
      L=IND(J,K,N)
      F(L)=F(L)+1
      I4=KK(I)
      IF(KK(I4) .EQ. I) GO TO 40
      L=IND(K,J,N)
      F(L)=F(L)+1
40    IF(J .NE. I1) GO TO 35
30    CONTINUE
C
C ... STEP 5
C
1     CALL EULER(N,F,NUMB,KK)
      RETURN
      END
```

```
C ... *** PROGRAM FOR SOLVING ASSIGNMENT PROBLEMS
C ... ***
C
C ... INPUT
C
C ... N      NUMBER OF ITEMS TO BE ASSIGNED TO EACH OTHER
C ... P(L)   COSTS OF ASSIGNING ITEM I=(L-1)/N+1 TO
C            ITEM J=L-(I-1)*N, I .NE. J, L=1,..,N*N
C
C ... OUTPUT
C
C ... KK(I)  NUMBER OF ITEM TO WHICH ITEM I IS ASSIGNED TO
C ... COST   MINIMUM ASSIGNMENT COSTS
C
      SUBROUTINE ASGNMT(N,P,KK,COST)
      INTEGER N,P(1),KK(1),COST,C(2704),Q(2704),F(2704)
C
C ... DEFINING THE NETWORK FOR THE MINIMUM COST FLOW PROGRAM
C
      NN=2*(N+1)
      NN1=NN*NN
      DO 10 I=1,NN1
      C(I)=0
10    Q(I)=0
      DO 5 I=1,N
      I1=I+1
      I2=I1+N
      L1=IND(1,I1,NN)
      L2=IND(I2,NN,NN)
      C(L1)=1
      Q(L1)=1
      C(L2)=1
5     Q(L2)=1
      DO 15 I=1,N
      DO 20 J=1,N
      IF(I .EQ. J) GO TO 20
      I1=I+1
      I2=J+N+1
      L=IND(I1,I2,NN)
      L1=IND(I,J,N)
      C(L)=P(L1)
      Q(L)=1
20    CONTINUE
15    CONTINUE
      NV=N
      NS=1
      NT=NN
C
C ... SOLVING THE MINIMUM COST FLOW PROBLEM
C
      CALL MINCOS(NN,C,Q,NV,NS,NT,F,COST)
C
C ... PREPARATION OF OUTPUT
C
      COST=COST-2*N
      DO 25 I=1,N
```

```
      DO 30 J=1,N
      I1=I+1
      I2=J+N+1
      L=IND(I1,I2,NN)
      IF(F(L) .EQ. 0) GO TO 30
      KK(I)=J
      GO TO 25
30    CONTINUE
25    CONTINUE
      RETURN
      END
```

6.1.2. Street cleaning with limited route length

In practice the assumption that all the streets are cleaned by just one vehicle is too simple. Usually on each vehicle, a constraint is imposed that restricts the length of the tour, because, for example, cleaning can only be performed a certain time per day or because a certain area has to be cleaned within a given time to guarantee that the total urban area can be cleaned, say , at least once a week. Talking about garbage collection, the constraint on the tour length can also mean that the total volume of garbage that can be collected during one tour is limited.

There are two possible ways in handling this problem. One can first solve the Chinese postman problem on the given graph and then break this tour into parts such that each part is a feasible tour. Or, one can first partition the given graph into smaller ones such that each subgraph can now be served by one vehicle and then solve the Chinese postman problem on each subgraph.

The first approach might not suit too well because the public administration might prefer the region split into seperated subregions for organisatorial reasons, which is certainly not a result of the first approach.

The second approach, however, has some methodological problems:

- The Euler tour formed for a subregion may not be feasible in the sense that the time capacity constraints of the vehicle may be violated.
- If all the Euler tours are feasible, then the total travel time over all tours may not be minimized.

Liebling (197o) stated heuristic algorithms for both approaches, while Beltrami & Bodin (1975) gave a brief description of an algorithm for the first approach. Here we shall only present an algorithm for the first approach which is a simplified version of the algorithm given by Liebling (197o).

Algorithm for finding the optimal tours for street cleaning vehicles

Step 1:

Let the original graph be G=(X,A). Solve the Chinese postman problem on G. Let $H=[a_1,a_2,\ldots,a_l]$ denote the sequence of arcs of the founded tour.

Step 2:

Let $B \subset X$ be the set of vertices in X from where vehicles start or stop (the garages of the vehicles). Find the shortest of all paths between any $x_i \varepsilon X$ and all $x_j \varepsilon B$. Denote the length by p_i.

Step 3:

Start at some vertex $x_j \varepsilon B$. Find the longest sequence of arcs in H, such that the sum of the p_i of the starting point, plus the length of the sequence of arcs, plus p_j of the final vertex of the cleaning tour, is less or equal to some fixed number L. Assign a vehicle to this tour (including the shortest path from and to a garage, to and from the initial and final vertex). Do this again until each arc in H is assigned to a tour.

Of course the result will only be suboptimal, as this heuristic algorithm depends strongly on the tour that has been found in Step 1 (usually the solution of Step 1 is not unique) and on the choice of the initial vertex of the first tour. So, if the result is not satisfying, one has to repeat the algorithm (which is very fast) with another route and a different initial vertex.

6.2. Municipal waste collection

Most refuse collection activities in a city center around the pickup of household refuse in small bins (problems of this type can be handled in the same way as street cleaning). However, large institutional sites such as schools, hospitals, and apartment complexes usually have their refuse stored in large containers. Thus, in many cities such sites will be serviced by different trucks than those collecting the garbage from normal households. Each such truck can service several such sites before going to a dump to unload. The problem to be considered is, then, how to route the trucks to minimize the total travel time of the vehicles and to determine the minimum number of trucks needed each day. This last condition is important from a point of view of minimizing the capital expenditure needed to outfit a fleet of trucks. If we are dealing with an unlimited tour length, then the problem to be solved is the travelling salesman problem which we shall discuss in chapter 6.2.1. If the tour length is restricted, only heuristic algorithms are applicable which we shall discuss in chapter 6.2.2.

6.2.1. Refuse collection with unlimited route length - the travelling salesman problem

The travelling salesman problem already has been studied extensively and various algorithms exist to solve it. However, we do not want to present them all. A very good review of some of them can be found in Christofides (1975). The approach we shall present here is based on the similarity between travelling salesman and assignment problem, such that for solving the travelling salesman problem the minimum cost flow algorithm of chapter 3.3.1 can be used. From the definition of a Hamiltonian circuit in a graph in chapter 2. it becomes clear that the travelling salesman problem is one of finding a Hamiltonian circuit (i.e. an elementary circuit which passes through all vertices of a given graph) with minimum length.

The linear assignment problem (which was already discussed in chapter 6.1.1.) for a graph with a cost matrix $C=[c_{ij}]$ can be stated as follows:

Let k_{ij} be an n x n matrix of 0-1 variables,so that $k_{ij}=1$ if vertex x_i is assigned to x_j and $k_{ij}=0$ otherwise. In the travelling salesman problem we could use a similar scheme,where $k_{ij}=1$ would mean that the truck travels from x_i to x_j directly and $k_{ij}=0$ would indicate that the truck does not. For this last problem we can assume $c_{ii}=\infty (i=1,\ldots,n)$ to eliminate non-sensical solutions with $k_{ii}=1$.

The assignment problem now becomes:
Find 0-1 variables k_{ij} so as to minimize

$$\text{min: } z = \sum_{j=1}^{n} \sum_{i=1}^{n} c_{ij}\, k_{ij} \qquad (6.3)$$

subject to

$$\sum_{i=1}^{n} k_{ij} = \sum_{j=1}^{n} k_{ij} = 1 \qquad (6.4)$$

(for all i and j = 1,2,...,n)

and

$$k_{ij} = 0 \text{ or } 1. \qquad (6.5)$$

Equations (6.4) simply insure that to each vertex x_i exactly one vertex x_j is assigned,or in terms of the travelling salesman, that each truck entering a vertex by an arc is also leaving this vertex. (6.3), (6.4) and (6.5), together form the assignment problem which can be solved by the minimum cost flow algorithm.

Together with the additional constraint that the solution must form a single (Hamiltonian) circuit and not just a number of disjoint circuits, the equations (6.3) - (6.5) represent a formulation of the travelling salesman problem. Since the addition of any constraint to the assignment problem can only increase or leave unchanged the minimum value of z as calculated from equations (6.3) - (6.5), this value of z is a valid lower bound to the cost of the solution to the travelling salesman problem for a graph with a cost matrix $[c_{ij}]$. Using therefore the objective of the assignment problem as a lower bound to the objective of the associated travelling salesman problem, a branch-and-bound algorithm can be stated to find the optimal solution of the travelling salesman problem.

Algorithm for the travelling salesman problem

Let X be the set of all sites to be serviced for garbage collection. Let A be the set of all arcs connecting the sites in X, which represent the shortest path between two sites using the road network. Therefore the graph G=(X,A) is complete (i.e. each pair of vertices is connected by an arc) with the cost of the arc c_{ij} being the length of the shortest path between x_i, $x_j \varepsilon X$. Set $c_{ii}=\infty$ and $C= [c_{ij}]$. Let Z(C) denote the value of the objective of the assignment problem solved for vertices in X and the cost matrix C. Let U(C) denote a sequence of arcs which form a circuit according to the optimal solution of the assignment problem on X with cost matrix C. Let k(C) denote the number of vertices which are incident to the arcs in U(C) and n denote the number of vertices in X.

Step 1:

Compute $Z(C)$
If $k(C) = n$, Stop.
Set $M=\infty$ and $D=(d_{ij})=C=(c_{ij})$.
Store $Z(C)$ into the set N, denoting the set of assignment problems to be analyzed further.

Step 2:

Delete $Z(D)$ from N.

Step 3:

Choose an arc $(x_i,x_j)\varepsilon U(D)$ and set its corresponding cost to $\bar{c}_{ij}=\infty$. Set $\bar{C} = D$, but instead of d_{ij} place $\bar{c}_{ij}$.
Solve $Z(\bar{C})$.
If $k(\bar{C}) = n$, compute $M=\min(M,Z(\bar{C}))$.
If $k(\bar{C}) < n$, store $Z(\bar{C})$ into N.
Set $U(D) = U(D) - \{(x_i,x_j)\}$.

Step 4:

If $U(D)\neq\emptyset$, go to Step 3.
If $U(D)=\emptyset$, go to Step 5.

Step 5:

Find the minimum value $Z(D)$ for all solved assignment problems stored in N.
If N is empty, set $Z(D) = \infty$.
If $Z(D) \geq M$, the travelling salesman problem is solved and is the solution of the assignment problem associated with M.
If $Z(D) < M$, go to Step 2.

The idea of the above stated algorithm is to exclude circuits that do not contain all vertices of X by setting the cost of one arc of the circuit to infinity (Step 3). Then the assignment problem associated to this new cost matrix will produce another circuit. This procedure is done until only one circuit is found with the shortest length (the smallest objective) of all assignment problems under consideration (Step 5).

```
C ... *** TRAVELLING SALESMAN PROGRAM
C ... ***
C
C ... INPUT
C
C ... N       NUMBER OF VERTICES
C ... C(L)    LENGTH OF ARC(I,J), WHERE L=IND(I,J,N). IF C(L)=0
C             THEN THIS ARC DOES NOT EXIST.
C
C ... OUTPUT
C
C ... LENGTH TOTAL LENGTH OF THE TRAVELLING SALESMAN TOUR
C ... NUMB    NUMBER OF VERTICES PASSED ON THE TOUR
C ... KK(I)   NUMBER OF THE I-TH VERTEX TO BE PASSED ON THE TOUR
C             I=1,...,NUMB
C
      SUBROUTINE TRAVSL(N,C,LENGTH,KK,NUMB)
      INTEGER N,C(1),LENGTH,KK(1),D(625),IG(625),U(10000,3)
      INTEGER K(25),K2(25)
      LOGICAL LOG
C
C ... STEP 1
C
      CALL SPII(N,C,D,LOG)
      CALL ASGNMT(N,C,K,LENGTH)
      J=1
      DO 10 I=1,N
      J=K(J)
      IF(J .NE. 1) GO TO 10
      JX=I
      GO TO 15
10    CONTINUE
15    IF(JX .EQ. N) GO TO 22
      M=2**30
      KU=1
      LU=1
      NN=N*N
C
C ... STEP 2
C
2     U(KU,2)=2**30
C
C ... STEPS 3 AND 4
C ... FINDING THE SHORTEST CIRCUIT
C
      LIY=N
      DO 95 I=1,N
      IH=0
      J=I
100   J=K(J)
      IH=IH+1
      IF(J .NE. I) GO TO 100
      IF(IH .GE. LIY) GO TO 95
      IY=I
      LIY=IH
95    CONTINUE
```

```
      I=IY
C
C ... COMPUTING THE NEW COSTS FOR THE ASSIGNMENT PROBLEM
C
3     JJ=K(I)
      L=IND(I,JJ,N)
      DO 25 II=1,NN
25    IG(II)=C(II)
      IG(L)=2**30
      II=KU
30    IF(II .EQ. 1) GO TO 35
      J=U(II,3)
      IG(J)=2**30
      II=U(II,1)
      GO TO 30
35    CALL ASGNMT(N,IG,K2,NCOS)
      IF(NCOS .GE. M) GO TO 20
      J=1
      DO 40 II=1,N
      J=K2(J)
      IF(J .NE. 1) GO TO 40
      JX=II
      GO TO 45
40    CONTINUE
45    IF(JX .LT. N) GO TO 50
      M=NCOS
      MLU=LU+1
50    LU=LU+1
      U(LU,2)=NCOS
      U(LU,1)=KU
      U(LU,3)=L
20    I=JJ
      IF(JJ .NE. IY) GO TO 3
C
C ... STEP 5
C
5     NX=M
      KU=MLU
      DO 55 J=1,LU
      I=LU-J+1
      IF(U(I,2) .GE. NX) GO TO 55
      NX=U(I,2)
      KU=I
55    CONTINUE
      DO 60 I=1,NN
60    IG(I)=C(I)
      I=KU
65    J=U(I,3)
      IG(J)=2**30
      I=U(I,1)
      IF(I .NE. 1) GO TO 65
      CALL ASGNMT(N,IG,K,LENGTH)
      IF(NX .LT. M) GO TO 2
C
C ... PREPARATION OF OUTPUT
C
22    NUMB=1
```

```
      KK(NUMB)=K(N)
      I2=K(N)
      DO 70 I=1,N
      DO 80 II=1,N
      IF(K(II) .NE. I2) GO TO 80
      I1=II
      GO TO 85
80    CONTINUE
85    J=I2
75    KX=J
      J=IND(I1,KX,N)
      J=D(J)
      NUMB=NUMB+1
      KK(NUMB)=J
      IF(J .NE. I1) GO TO 75
      I2=I1
70    CONTINUE
      NX=NUMB/2
      DO 90 I=1,NX
      J=NUMB-I+1
      KH=KK(I)
      KK(I)=KK(J)
90    KK(J)=KH
      NUMB=NUMB-1
      RETURN
      END
```

6.2.2. Refuse collection with limited route length

Dealing with practical problems, the question is not simply one of finding the travelling salesman circuit since there are a number of complicating factors. First, one must be mindful of capacity and time constraints. Each pickup point can have a different quantity to be picked up and, since the capacity of the truck is limited, the route must be interrupted for travel between pickup points to dumps. Moreover, there are several dump sites. Having saturated the truck, the problem asked is which dump site should be used? Finally, some locations require daily service while others do not. Since there are typically many points to be serviced, the problem is not only to arrange the routes feasible but to assign each pickup point to days of the week to minimize the number of trucks.

Here we shall only be dealing with the simpler problem, where the pickup points are already assigned to days of the week and only the tours for the day have to be found. We shall also restrict ourselves to the problem with only one dump site. For a more detailed discussion of the problem see Beltrami & Bodin (1974). The algorithm presented here is heuristic by nature .
The idea of the algorithm is to combine vertices to lie on the same route, such that the savings in terms of route length is maximized, compared to the two separated tours for each vertex and that the time and capacity constraints are not validated. Let T denote the set of tours yet found. For each tour $t \in T$ let $p(t)$ denote the initial and the final vertex (site) x_i and $x_j \in X$ (the set of all vertices) of the tour (excluding the dump site). Let the costs associated to the arcs $(x_i, x_j) \in A$ be c_{ij}.

Algorithm for finding refuse collection tours

Step 1:

For all vertices $x_i \varepsilon X$ let the initial tours consist of one vertex only, therefore for each $t \varepsilon T$ $p(t)=\{x_i, x_i\}$, $x_i \varepsilon X$. Compute shortest paths between all pairs of vertices in X. Let the shortest route from p(t) to the dump site x_o be denoted by $s_1(t)$ (from dump site to initial vertex of tour t) and $s_2(t)$ (from final vertex of tour to dump site x_o).

Step 2:

Let n(T) be the maximum number of vertices in a tour $t \varepsilon T$. Let the set of such tours be $T_n \subset T$. Set l=n(T).

Step 3:

For all tours $t_1 \varepsilon T_l$ and all tours $t_2 \varepsilon T_q$, $q \leq l$, let $S(t_1,t_2)$ denote the savings that would result if these two tours would be combined to one. This is done by eliminating two paths $s_i(t)$ from the dump site x_o to one initial vertex of t_1 or t_2 and one final vertex of t_2 or t_1 and adding the path length to combine the final vertex of t_1 or t_2 with the initial vertex of t_2 or t_1. Find the pair of tours (t_i,t_j) with the largest savings $S(t_i,i_j)$, such that the time and capacity constraint is not validated.

Step 4:

If such a pair exists, eliminate t_i and t_j from T, add to T the new tour (t_i,t_j) and go to Step 2.
If no such pair exists, set l=l-1.
If $l \geq 1$, go to Step 3.
If l=0, the tours have been found, Stop.

Note that all vertices $x_i \varepsilon X$, except the dump site x_o, always belong to one, and only one, tour in T. This algorithm can easily be expanded to the case with more than one dump site.

```
C ... *** ALGORITHM FOR SOLVING TRAVELLING SALESMAN PROBLEM
C ... *** WITH RESTRICTED TOUR LENGTH
C ... ***
C
C ... INPUT
C
C ... N       NUMBER OF VERTICES
C ... C(L)    LENGTH OF ARC(I,J), WHERE L=IND(I,J,N)
C ... LT      MAXIMAL ALLOWED LENGTH OF A TOUR
C ... IT IS ASSUMED THAT EACH TOUR STARTS AND ENDS AT VERTEX 1
C
C ... OUTPUT
C
C ... NT      NUMBER OF TOURS
C ... NV(I)   NUMBER OF VERTICES BELONGING TO TOUR I, I=1,...,NT
C ... NVX(I,J) J-TH VERTEX OF TOUR I, I=1,...,NT, J=1,...,NV(I)
C
      SUBROUTINE RECOTO(N,C,LT,NT,NV,NVX)
      INTEGER N,C(1),LT,NT,NV(1),NVX(30,30)
      INTEGER D(900),S(30,2)
      LOGICAL LOG
C
C ... STEP 1
C
      CALL SPII(N,C,D,LOG)
      NT=N-1
      DO 5 I=1,NT
      NVX(I,1)=I+1
      NV(I)=1
      I1=I+1
      L1=IND(1,I1,N)
      L2=IND(I1,1,N)
      S(I,1)=C(L1)
5     S(I,2)=C(L2)
      NTMAX=1
C
C ... STEP 2
C
2     L=NTMAX
C
C ... STEP 3
C
3     MAXSAV=0
      DO 10 I=1,NT
      IF(NV(I) .NE. L) GO TO 10
      DO 15 J=1,NT
      IF(I.EQ.J .OR. NV(J).GT.L) GO TO 15
      I1=NV(I)
      I1=NVX(I,I1)
      I2=NVX(J,1)
      L1=IND(I1,I2,N)
      LL1=S(I,2)+S(J,1)-C(L1)
      LENG=C(L1)+S(I,1)+S(J,2)
      IF(LL1 .LE. MAXSAV) GO TO 20
      IF(NV(I) .EQ. 1) GO TO 25
      NZ=NV(I)
```

```
      DO 30 JL=2,NZ
      JL1=NVX(I,JL-1)
      JL2=NVX(I,JL)
      L2=IND(JL1,JL2,N)
30    LENG=LENG+C(L2)
25    IF(NV(J) .EQ. 1) GO TO 35
      NZ=NV(J)
      DO 40 JL=2,NZ
      JL1=NVX(J,JL-1)
      JL2=NVX(J,JL)
      L2=IND(JL1,JL2,N)
40    LENG=LENG+C(L2)
35    IF(LENG .GT. LT) GO TO 20
      MAXSAV=LL1
      II=I
      JJ=J
20    I1=NV(J)
      LENG=LENG-C(L1)-S(I,1)-S(J,2)
      I1=NVX(J,I1)
      I2=NVX(I,1)
      L1=IND(I1,I2,N)
      LENG=LENG+C(L1)+S(J,1)+S(I,2)
      IF(LENG .GT. LT) GO TO 15
      LL1=S(J,2)+S(I,1)-C(L1)
      IF(LL1 .LE. MAXSAV) GO TO 15
      MAXSAV=LL1
      II=J
      JJ=I
15    CONTINUE
10    CONTINUE
C
C ... STEP 4
C
      IF(MAXSAV .EQ. 0) GO TO 45
      I1=1+NV(II)
      I2=I1+NV(JJ)-1
      DO 55 I=I1,I2
55    NVX(II,I)=NVX(JJ,I-I1+1)
      NV(II)=I2
      S(II,2)=S(JJ,2)
      NV(JJ)=NV(NT)
      S(JJ,1)=S(NT,1)
      S(JJ,2)=S(NT,2)
      NZ=NV(JJ)
      DO 50 I=1,NZ
50    NVX(JJ,I)=NVX(NT,I)
      NT=NT-1
      NTMAX=I2
      GO TO 2
45    L=L-1
      IF(L .GE. 1) GO TO 3
      RETURN
      END
```

6.3. School bus routing

With the growing need for better education in all countries, there seems to be a great tendency towards building larger schools which often results in reducing the total number of schools, thus applying pupils with better facilities which can be of full usage only if the number of pupils is big enough. Especially in areas where the population density is low, this causes a lot of problems because somehow the pupils have to go to school and the supply of transportation facilities via public transportation systems is in many places not sufficient. Therefore in many countries special buses are used to pick up pupils at some points that can easily be reached from their homes and carry them to school and vice versa. Of course, the expenses for this transportation facility are high and efforts for reducing the total amount of buses needed to meet the demands are undertaken. The problem can again be considered as a routing problem, similar to the one we described in chapter 6.2.2. The most general approach to this particular problem was published by Newton & Thomas (1974), which we shall state-with some modifications - in the following. Like in all papers on this subject, it is assumed that only one school is served at a time, i.e. all buses transport pupils only to one school. If more than one school must be served by the same buses, then the idea is that those schools are served one after the other, which means that schools do not start (and end) at the same time, but after some interval. Then we are confronted with the following problem: Given a network $G=(X,A)$, where $s \varepsilon X$ is the location of the school and $O \subset X$ is the set of vertices where buses are located, then from each vertex $x_i \varepsilon X$ a given number of pupils needs to be taken to the school $s \varepsilon X$. The shortest transportation time between each pair of vertices is given as the cost c_{ij} of the arc connecting them (G is a complete graph, i.e. there is an arc between each pair of vertices in X). Therefore it holds that $c_{ij} \leq c_{ik} + c_{kj}$. For each vertex $x_i \varepsilon 0 \subset X$, the number of buses that are located there is given. All buses are assumed to have the same capacity C of people to be taken with. Finally, the maximal length of a route taken by any bus from its origin $x_i \varepsilon 0$ to the school $s \varepsilon X$ is also restricted by R, being the same for all buses. That the tour must not exceed R has two reasons: First, all pupils have to be transported within some time interval to have the buses available for the next school to be served and secondly, pupils should not have to sit in buses for hours. Then the problem is to find the minimum amount of buses needed to meet all constraints, and to assign a route to each bus actually used with shortest length. Note that this problem formulation includes the possibility that all buses start from the school, meaning that $0=\{s\}$.

In this case, the problem would be exactly the same than the one of finding refuse collection tours with restricted length (see chapter 6.2.2.). Obviously, as this problem is even more complex than the one of chapter 6.2.2.,only a heuristic method can be used, the idea of which being the following: Assign buses to vertices in 0. Find a shortest path from the vertex in 0, to which the most buses have been assigned, to the school by passing through all vertices. Partition this path into subpathssuch that these routes do not validate the capacity and time constraint of the buses. Iterate this procedure to find routes such that their total length is reduced and, finally, assign the buses located at the vertices in 0 to the routes.

Algorithm for solving the school bus routing problem

Step 1: (Preliminary computations)

Find a lower bound for the number of buses necessary to transport all pupils to school, being

$$R_{min} = \left[\frac{\text{number of pupils to be transported}}{\text{bus capacity } C} \right] ,$$

where [.] denotes the next largest integer.
Denote by R_{max} the number of buses actually used and set $R_{max} = R_{min}$.

Calculate a lower bound for the average length of a route from any bus origin $x_i \varepsilon O$ to school $s \varepsilon X$. As each bus has to pass by at least one vertex on its way from x_i to s, the average minimum route length $\bar{p}_i$ from x_i to s is then

$$\bar{p}_i = \frac{L_i}{v-2} ,$$

where L_i is the sum of the length of all arcs

$$L_i = \sum_{\substack{x_j \varepsilon X \\ x_j \neq x_i, s}} (c_{ij} + c_{js})$$

and v is the number of vertices in X.

Let k_i be the number of buses available at origin $x_i \varepsilon O$. It is assumed that there are not less buses available in the total than there are needed. Order the vertices $x_i \varepsilon O$ such that $\bar{p}_1 \leqslant \bar{p}_2 \leqslant \ldots \leqslant \bar{p}_l$, where l is the number of vertices in O.

Let the sum of the length over all bus routes, for the best solution yet found, be B and set $B = \infty$.

<u>Step 2</u>:

Set $R_{max} = R_{max} + 1$.
Assign k_1 buses to x_1, k_2 to x_2 until all needed buses R_{max} are assigned to some origin. Of course, if $R_{max} < k_1$, then only assign R_{max} buses to x_1.

Step 3:

Find the origin to which the most buses have been assigned and order the vertices $x_i \varepsilon 0$, such that this origin is x_1.
Set r=2 and $F_1 = \infty$.

Step 4:

Find a path from x_1 to s by choosing first to go from x_1 to x_r ($x_r \neq s$) and then always choosing a vertex x_i, which does not yet belong to the path, such that the travel time to this vertex is shortest among the possible ones. If all vertices in X except s belong to the path, then go to s. Determine the length of this path and denote it by F_r.
Set $F_r = \min(F_r, F_{r-1})$.

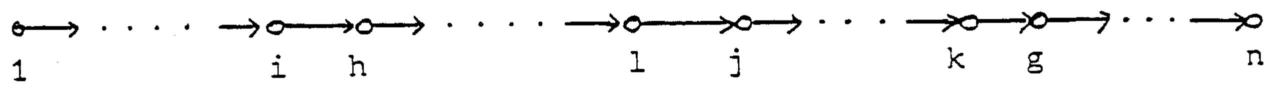
Step 5:

Generate a set of bus routes each of which starts at x_1, going along the path under consideration until either the bus capacity is reached (by loading at each vertex all pupils assigned to this vertex) or the time constraint (by adding up the arc lengths), proceeding to s then. The next route again starts at x_1 and proceeds directly to the first vertex of the path not yet assigned to a bus route. All individual bus routes are determined in the same manner until all vertices in X are assigned to some bus route. Let the sum of the length of these bus routes be S and the number of bus routes be t.
If $t \leq R_{max}$, then go to Step 6.
If $t > R_{max}$, then go to Step 7.

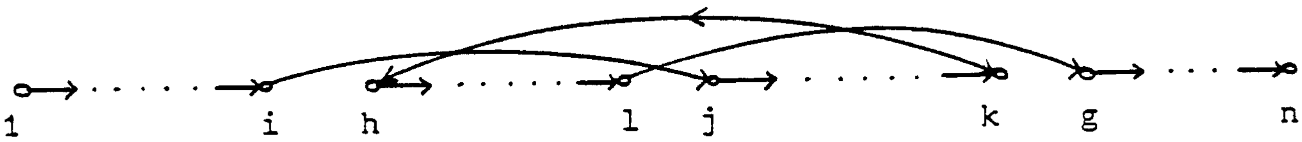
Step 6:

Try to improve each bus route found in Step 5 individually in the following way:
Let (1,2,...,i,...,j,...,k,...,n) denote the sequence of

vertices in the route. Then for each triple of vertices i,j,k such that $1 \leq i < h < j \leq k < n$ delete three arcs and build in three new arcs such that:



original



new

Note that the new route has the same direction, the same initial and final vertices as the original one. If this new route is shorter than the original one, i.e. if

$$c_{ih} + c_{lj} + c_{kg} > c_{ij} + c_{lg} + c_{kh} \quad ,$$

then use the new route instead of the old one. The number of possible new routes for a route with $n \geq 4$ vertices is growing with $O(n^3)$.

Let the length over all bus routes be again S, then set B=min(B,S). Go to Step 7.

Step_7:

If $F_r < F_{r-1}$, then try to improve the path associated to F_r with the algorithm described in Step 6. If an improvement is found, set the length of this new path F_r and go to Step 5. If no improvement can be found or if $F_r \geq F_{r-1}$, set r=r+1. If $r \leq v$, then go to Step 4. If $r > v$, and $B=\infty$, then go to Step 2, otherwise go to Step 8.

Step 8:

Assign the buses from the vertices $x_i \varepsilon O$ to the optimal routes that are associated to B in the following way:

Let P be the set of vertices that are the first ones on each bus route. Construct a network N=(Y,B) where

$$Y = P \cup O \cup \{o\} \cup \{d\}$$

and B consists of

arc (o,x_i), $x_i \varepsilon O$ with no arc cost and arc capacity k_i (the number of buses at x_i)

arc (x_i,x_j), $x_i \varepsilon O$, $x_j \varepsilon P$ with arc cost c_{ij} (the travelling cost from x_i to x_j) and arc capacity =1. If $x_i = x_j$, set $c_{ij}=0$.

arc(x_j,d), $x_j \varepsilon P$ with no arc cost and arc capacity =1.

Now send a flow of the amount R_{max} from o to d and find a minimum cost flow pattern with the minimum cost flow algorithm.

From each $x_i \varepsilon O$ to each $x_j \varepsilon P$, where there is a non zero flow in the optimal solution, a bus is assigned to the route starting at x_j.

This algorithm avoids the time consuming solution of a travelling salesman problem. Instead, the less expensive heuristic search presented in Step 6 is performed, although for large networks this also can cause problems. In this case one has to reduce this search and not examine all possible combinations of i,j,k.

Notice that although for O= {s} the problem of chapter 6.2.2. and the school bus problem are equal, the algorithms proposed for each problem are not. Because of their heuristic nature one cannot say a priori anything about the relation between their results.

```
C ... *** PROGRAM FOR SOLVING THE SCHOOL BUS ROUTING PROBLEM ***
C ... ***
C
C ... INPUT
C
C ... N           NUMBER OF VERTICES
C ... C(L)        LENGTH OF ARC(I,J), WHERE L=IND(I,J,N)
C ... PUP(I)      NUMBER OF PUPILS AT VERTEX I WHO HAVE TO BE
C                 TRANSPORTED TO THE SCHOOL AT VERTEX 1, I=2,...,N
C ... O(I)        NUMBER OF BUSES LOCATED AT VERTEX I, I=1,..,N
C ... CAP         CAPACITY OF A BUS
C ... ML          MAXIMAL LENGTH OF A BUS TOUR
C
C ... OUTPUT
C
C ... RMAX        NUMBER OF BUSES NECESSARY FOR TRANSPORTATION
C ... NV(I)       NUMBER OF VERTICES BELONGING TO TOUR I, I=1,...,RMAX
C ... NVX(I,J)    J-TH VERTEX OF TOUR I, J=1,...,NV(I). BETWEEN THE
C                 J-TH AND THE (J+1)-ST VERTEX THE SHORTEST PATH HAS TO
C                 BE USED.
C
      SUBROUTINE SCHOOL(N,C,PUP,O,CAP,ML,RMAX,NV,NVX)
      INTEGER N,C(1),PUP(1),O(1),CAP,ML,RMAX,NV(1),NVX(30,30)
      INTEGER RMIN,P(30),B,K(30),F(30),R,S,T,D(900),PX(30),PT(30)
      INTEGER TT,NVY(30),NVXY(30,30)
      LOGICAL LOG
C
C ... STEP 1 (PRELIMINARY COMPUTATIONS)
C
      CALL SPII(N,C,D,LOG)
      RMIN=0
      DO 10 I=2,N
10    RMIN=RMIN+PUP(I)
      RMIN=RMIN/CAP+1
      RMAX=RMIN-1
      NBUS=0
      DO 15 I=1,N
15    NBUS=NBUS+O(I)
      DO 20 I=1,N
      P(I)=0
      IF(O(I) .EQ. 0) GO TO 20
      DO 25 J=2,N
      IF(J .EQ. I) GO TO 25
      L1=IND(I,J,N)
      L2=IND(J,1,N)
      P(I)=P(I)+C(L1)+C(L2)
25    CONTINUE
20    P(I)=P(I)/(N-2)
      I=1
      J=1
35    PX(I)=J
      J=2**30
      DO 40 IX=1,N
      IF(P(IX).LT.PX(I) .OR. P(IX).GE.J) GO TO 40
      J=P(IX)
      P(IX)=0
```

```
        JJ=IX
 40     CONTINUE
        IF(J .EQ. 2**30) GO TO 45
        PX(I)=JJ
        I=I+1
        IF(I .LE. N) GO TO 35
 45     B=2**30
        NK=I-1
C
C ... STEP 2
C
 2      RMAX=RMAX+1
        IF(RMAX .LE. NBUS) GO TO 30
        PRINT *,' THERE ARE NOT ENOUGH BUSES AVAILABLE'
        RETURN
 30     MH=RMAX
        DO 50 I=1,N
 50     K(I)=0
        DO 55 I=1,NK
        J=PX(I)
        K(J)=MIN0(MH,O(J))
        MH=MH-K(J)
        IF(MH .EQ. 0) GO TO 3
 55     CONTINUE
C
C ... STEP 3
C
 3      MY=0
        DO 60 I=1,NK
        J=PX(I)
        IF(J .EQ. 1) GO TO 60
        IF(K(J) .LE. MY) GO TO 60
        MY=K(J)
        JZ=J
 60     CONTINUE
        R=2
        F(1)=2**30
C
C ... STEP 4
C
 4      IF(R .EQ. JZ) F(R)=F(R-1)
        IF(R .EQ. JZ) R=R+1
        IF(R.GT.N .AND. B.EQ.2**30) GO TO 2
        IF(R .GT. N) GO TO 8
        DO 65 I=1,N
 65     P(I)=I
        PT(1)=JZ
        P(JZ)=0
        PT(2)=R
        P(R)=0
        I=3
        L=IND(JZ,R,N)
        F(R)=C(L)
 70     MH=2**30
        DO 75 J=2,N
        IF(P(J) .EQ. 0) GO TO 75
        L=IND(PT(I-1),J,N)
```

```
      IF(C(L) .GE. MH) GO TO 75
      MH=C(L)
      JA=J
75    CONTINUE
      PT(I)=JA
      P(JA)=0
      F(R)=F(R)+MH
      I=I+1
      IF(I .LT. N) GO TO 70
      PT(I)=1
      L=IND(PT(I-1),1,N)
      F(R)=F(R)+C(L)
      F(R)=MINO(F(R),F(R-1))
      LOG=.FALSE.
C
C ... STEP 5
C
5     T=0
      I=2
80    T=T+1
      NVY(T)=1
      NVXY(T,1)=JZ
      MCAP=0
      IF(T .EQ. 1) MCAP=PUP(JZ)
      MML=0
      J=2
85    L1=IND(NVXY(T,J-1),PT(I),N)
      L2=IND(PT(I),1,N)
      MH=MML+C(L1)+C(L2)
      IF(MH .GT. ML) GO TO 90
      L3=PT(I)
      MH=MCAP+PUP(L3)
      IF(MH .GT. CAP) GO TO 90
      NVY(T)=NVY(T)+1
      NVXY(T,J)=PT(I)
      I=I+1
      J=J+1
      MCAP=MH
      MML=MML+C(L1)
      IF(I .EQ. N) GO TO 90
      IF(I .GT. N) GO TO 95
      GO TO 85
90    NVY(T)=NVY(T)+1
      NVXY(T,J)=1
      IF(I .EQ. N) GO TO 95
      GO TO 80
95    IF(T .GT. RMAX) GO TO 7
C
C ... STEP 6
C
6     S=0
      DO 100 I=1,T
      DO 105 J=1,NVY(I)
105   P(J)=NVXY(I,J)
      CALL IMPR(N,NVY(I),P,C,M)
      DO 106 J=1,NVY(I)
106   NVXY(I,J)=P(J)
```

```
100     S=S+M
        IF(B .LE. S) GO TO 7
        TT=T
        DO 120 I=1,TT
        NV(I)=NVY(I)
        DO 125 J=1,NV(I)
125     NVX(I,J)=NVXY(I,J)
120     CONTINUE
        B=S
C
C ... STEP 7
C
7       IF(F(R) .GE. F(R-1)) GO TO 110
        IF(LOG) GO TO 110
        CALL IMPR(N,N,PT,C,M)
        IF(M .GE. F(R)) GO TO 110
        LOG=.TRUE.
        GO TO 5
110     R=R+1
        IF(R .LE. N) GO TO 4
        IF(B .EQ. 2**30) GO TO 2
C
C ... STEP 8
C
8       RMAX=TT
        DO 130 I=1,N
        DO 135 J=1,RMAX
        J1=NVX(J,2)
        IF(J .EQ. 1) J1=NVX(1,1)
        L1=IND(I,J1,N)
        L2=IND(I,J,N)
        D(L2)=C(L1)
135     IF(I .EQ. J1) D(L2)=0
130     CONTINUE
        CALL ASGNMX(RMAX,D,N,O,P,M)
        DO 140 I=1,N
        IF(P(I) .EQ. 0) GO TO 140
        J=P(I)
        NVX(J,1)=I
140     CONTINUE
        RETURN
        END
```

```
C ... *** PROGRAM FOR REDUCING THE LENGTH OF A SCHOOL BUS ROUTE ***
C ... ***
C
C ... INPUT
C
C ... N         NUMBER OF VERTICES
C ... C(L)      LENGTH OF ARC(I,J), WHERE L=IND(I,J,N)
C ... NP        NUMBER OF VERTICES IN THE TOUR
C ... P(I)      I-TH VERTEX IN THE TOUR
C
C ... OUTPUT
C
C ... P(I)      I-TH VERTEX IN THE IMPROVED TOUR
C ... M         LENGTH OF THE IMPROVED TOUR
C
      SUBROUTINE IMPR(N,NP,P,C,M)
      INTEGER N,NP,P(1),C(1),M,H,L,G,II,HH,LL,JJ,KK,GG,PX(30)
      IF(NP .LE. 3) GO TO 20
      NP1=NP-3
      MSPAR=0
      NP2=NP-1
      DO 5 I=1,NP1
      H=I+1
      MH=H+1
      II=P(I)
      HH=P(H)
      DO 10 J=MH,NP2
      L=J-1
      JJ=P(J)
      LL=P(L)
      DO 15 K=J,NP2
      G=K+1
      KK=P(K)
      GG=P(G)
      L1=IND(II,HH,N)
      L2=IND(LL,JJ,N)
      L3=IND(KK,GG,N)
      L4=IND(II,JJ,N)
      L5=IND(LL,GG,N)
      L6=IND(KK,HH,N)
      L7=C(L1)+C(L2)+C(L3)-C(L4)-C(L5)-C(L6)
      IF(MSPAR .GE. L7) GO TO 15
      MSPAR=L7
      I1=I
      J1=J
      K1=K
15    CONTINUE
10    CONTINUE
5     CONTINUE
      IF(MSPAR .EQ. 0) GO TO 20
      DO 25 I=1,NP
25    PX(I)=P(I)
      H=I1+1
      L=J1-1
      DO 30 I=J1,K1
30    P(H+I-J1)=PX(I)
```

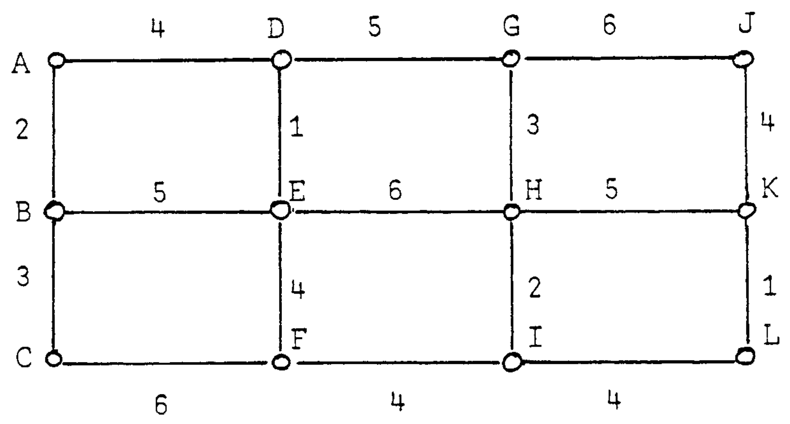
```
      DO 35 I=H,L
35    P(H+K1-J1+I+1-H)=PX(I)
20    M=0
      DO 40 I=2,NP
      I1=P(I-1)
      I2=P(I)
      L1=IND(I1,I2,N)
40    M=M+C(L1)
      RETURN
      END
```

```
C ... *** PROGRAM FOR SOLVING ASSIGNMENT PROBLEMS
C ... ***
C
C ... INPUT
C
C ... N       NUMBER OF ITEMS TO WHICH AN ASSIGNMENT IS MADE
C ... P(L)    COSTS OF ASSIGNING ITEM I TO ITEM J, WHERE L=IND(I,J,N)
C ... K(I)    NUMBER OF ITEMS I WHICH CAN BE ASSIGNED, I=1,..,M
C ... M       NUMBER OF DIFFERENT ITEMS WHICH CAN BE ASSIGNED
C
C ... OUTPUT
C
C ... KK(I)   NUMBER OF ITEM TO WHICH ITEM I IS ASSIGNED TO
C ... COST    MINIMUM ASSIGNMENT COSTS
C
       SUBROUTINE ASGNMX(N,P,M,K,KK,COST)
       INTEGER N,P(1),KK(1),COST,C(2704),Q(2704),F(2704),K(1)
C
C ... DEFINING THE NETWORK FOR THE MINIMUM COST FLOW PROGRAM
C
       NN=2+N+M
       NN1=NN*NN
       DO 10 I=1,NN1
       C(I)=0
10     Q(I)=0
       DO 5 I=1,M
       I1=I+1
       L1=IND(1,I1,NN)
       C(L1)=1
5      Q(L1)=K(I)
       DO 35 J=1,N
       I2=M+1+J
       L2=IND(I2,NN,NN)
       C(L2)=1
35     Q(L2)=1
       DO 15 I=1,M
       DO 20 J=1,N
       I1=I+1
       I2=J+M+1
       L=IND(I1,I2,NN)
       L1=IND(I,J,N)
       C(L)=P(L1)
       Q(L)=1
20     CONTINUE
15     CONTINUE
       NV=N
       NS=1
       NT=NN
C
C ... SOLVING THE MINIMUM COST FLOW PROBLEM
C
       CALL MINCOS(NN,C,Q,NV,NS,NT,F,COST)
C
C ... PREPARATION OF OUTPUT
C
       COST=COST-2*N
```

```
      DO 25 I=1,M
      KK(I)=0
      DO 30 J=1,N
      I1=I+1
      I2=J+M+1
      L=IND(I1,I2,NN)
      IF(F(L) .EQ. 0) GO TO 30
      KK(I)=J
      GO TO 25
30    CONTINUE
25    CONTINUE
      RETURN
      END
```

6.4. Exercises

1) Given the following undirected network (the numbers on the arcs denote their length in hundred meters), representing a road network in a town (the vertices represent intersections of the streets).



Find a shortest cycle for a vehicle situated at vertex D to clean all streets, with the Chinese postman algorithm .

2) Find an Eulerian circuit in the nondirected graph given by the following incidence matrix:

	a	b	c	d	e	f	g	h	i
a	0	0	0	1	1	0	1	1	0
b		1	1	1	0	1	0	0	1
c			0	1	0	1	0	0	1
d				0	1	1	1	1	1
e					1	0	1	1	0
f						0	0	0	1
g							0	1	0
h								0	0
i									1

7. Route planning for urban public transportation systems

7.1. The problem and the need for solving it

Transportation has become one of the urgent problems of urban areas. In many cities the gap between the need for and the possibilities of transportation seems still to be growing, resulting in increasing travel times for citizens. Apart, air pollution, noise and accidents are quite unwanted side effects of this development. Therefore new solutions and planning in urban transportation systems are needed. Although on the political scene it is still an open question if public transportation systems, should be given priority to individual car traffic, it has become quite clear that individual transportation systems like the present one by cars, will not be able to solve the urban transportation problem. Since presently there exists no other alternative to cars than buses, trams and railways (above and under the earth), strong efforts to improve these mass transportation systems should be undertaken.

Planning transportation systems can hardly be separated into various subproblems, because all the aspects of such a system depend on each other.

Transportation planning tries to fulfill the forecasted demand, but rarely takes into account that future demand also depends on the results of todays transportation planning. People choose their jobs and homes depending on transportation facilities, firms, shops and offices try to find good locations that also depend on transportation facilities. Therefore a good transportation system tends to create new transportation demands. This important effect, in the long run, can hardly be forecasted yet - too little is known about behaviour of people in this aspect.

Given a certain transportation demand, public and individual transportation are in a competitive situation. Although it is not too difficult to evaluate the number of people that use one or the other transportation possibility, only weak models exist that try to forecast the splitting of the demand between the two possibilities, if the transportation systems are changed, e.g. if a better road or a new bus - or underground line is built. Forecast models of this type, called modal-split models,are presented in P.Mäcke & H.Hensel (1975). So far,modal-split models are only of the forecast-type and do not try to optimize certain criterions of transportation planning.

Quantitative models that do optimize or suboptimize come into consideration, if one is willing to neglect

- long run effects that means, one accepts the assumption of independence between demand forecast and transportation planning and

- the competitive situation between public and individual transportation,implying that the demand for a public transportation system does not change if this system is changed. Therefore it is assumed that people either go by car or by a public transportation system, but do not move from one to the other.

Thus, being quite aware of the assumptions we have to state for an optimization problem, we do not yet see a way out of this dilemma. Although the need for better planning in urban mass transportation systems seems to be obvious, surprisingly little has been published in this area. It was not until 1967, when W.Lampkin & P.D.Saalmans (1967) reported on an attempt to reorganize the bus routes and frequencies of a public transportation system in an English town with the help of an Operational Research approach. Besides this work only two other case studies on this subject can be found in the literature, being published by Silman, Barzily & Passy (1974), who worked on the same problem for Haifa, Israel and Hoidn (1977), who studied the problem for a Swiss town. Uebe (197o) and especially Friedman (1976) reported on models for optimal scheduling of mass transportation vehicles.

In this chapter we shall discuss the problem of network optimization for urban mass transportation systems in detail, also stating not yet published algorithms.

Because the network of an urban public transportation system has some special characteristics, we shall develop a shortest path algorithm which will perform better on such special networks than the ones of chapter 3.1. Chapter 7.3. will then be dealing with the problem of finding a descriptive approach of assigning passengers to the routes of the transportation vehicles.Finally, we shall treat the problem of finding optimal routes for transportation vehicles in a given network to meet some objectives of transportation planning.

7.2. Shortest paths in public transportation networks

Looking at an urban public transportation network that is built by bus or tram lines,we can note some special characteristics. The vertices of this network usually indicate a stop or, more generally,an area of a city that has to be served via this stop. - This area should be small enough to reach every point within it from the stop by foot. - The arcs connecting the vertices will then either denote the street (in case of a bus), along which the bus drives or denote the rails (in case of a tram), along which the tram proceeds. Of course, not each vertex (and therefore not each arc) will be served by all bus or tram lines that are running within the city. By a line we mean the route of a vehicle (tram or bus) along the specified network consisting of streets or rails. Usually lines have two final points (vertices), called terminals, where they turn and change the direction, but there also exist ring lines, where the route goes along a cycle of the network, thus no terminals exist. Because in practice most of the lines use the same arcs in both directions - this is especially true for trams, sometimes it is not true for buses, because of one-way-streets, but even then the routes are very close to each other (i.e. the next street), we shall assume that the transportation network (consisting of stops and streets or rails) is not directed. Formally speaking we are dealing with a nondirected network of streets or rails $G=(X,A)$ and a set of lines L defined on it, such that each element $l \in L$ denotes a chain (a nondirected path) which can be a cycle. Because each stop (vertex in X) must be served by at least one line, each $x \in X$ must belong to at least one $l \in L$. Certainly each passenger of a public transportation system wants to be able to reach stop $x_i \in X$ from any other $x_j \in X$ by using a sequence of lines $l_i \in L$, changing the lines at vertices $x_k \in X$ which belong at least to those two lines between which the passenger changes. Let us denote by $V(l) \in X, l \in L$, the set of vertices that belong to line l and $S(l) \in A$, $l \in L$, the set of arcs that belong to line l. Then the following must hold

$$\bigcup_{l_i \in L} V(l_i) = X \qquad (7.1)$$

If we call the set of arcs that belong to at least one line F,

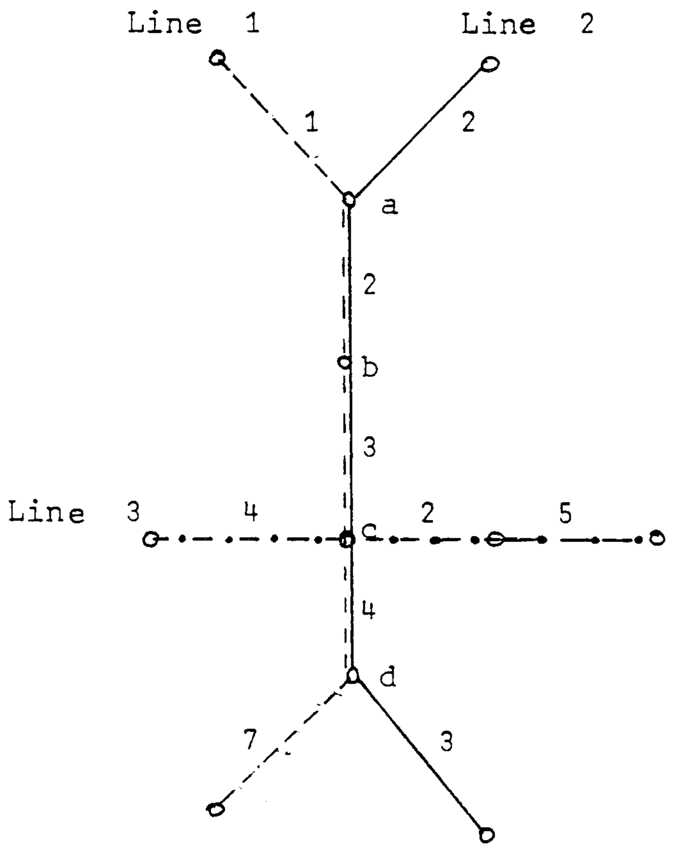
$$F = \bigcup_{l_i \in L} S(l_i) \subset A \qquad (7.2)$$

then the network $\bar{G} = (X,F)$ must be strongly connected (or strong), i.e. any two vertices are mutually reachable. G must be strong too, of course.

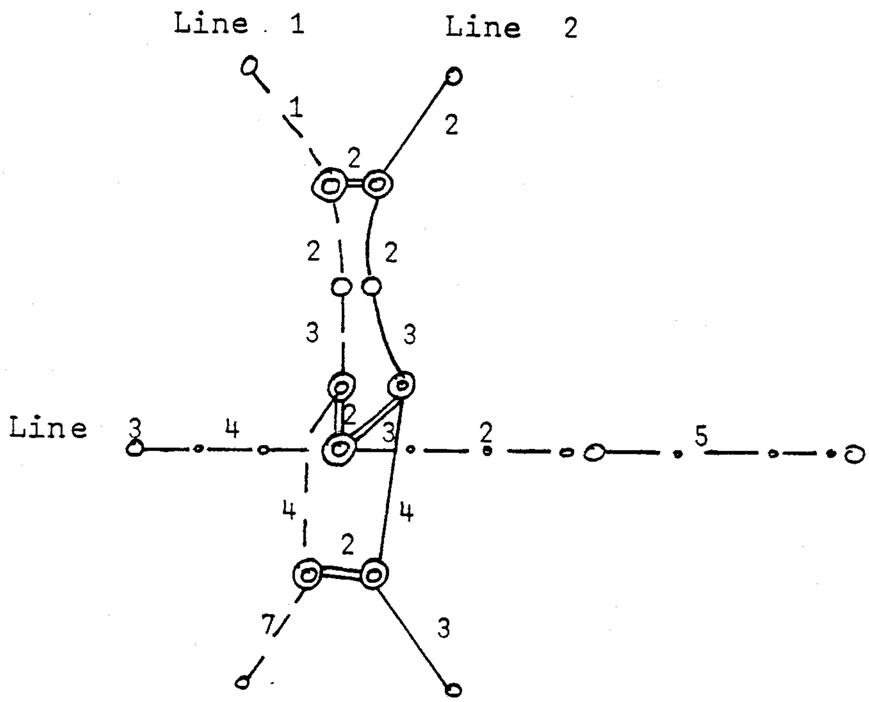
Let us now call the line-degree of a vertex the number of lines $l_i \varepsilon L$ to which the vertex belongs. Then we can state that if the line-degree of $x \varepsilon X$ is one, the degree of $x \varepsilon X$ must be one (if it is a terminal vertex) or two. This is obvious because each line is a chain, which means that each vertex of the chain is incident to one arc (if it is an initial or final vertex of the chain) or to two arcs. Because any community wants to keep the number of lines small (for operational and financial reasons, as we shall see in chapter 7.4) and also wants the lines to proceed on near to shortest paths in the network G (which will again be argued for in chapter 7.4), many vertices in the network $\bar{G}$ will only have line-degree one, therefore having degree one or two. This now is a special characteristic of $\bar{G}$, which we were talking about at the beginning of this chapter and which we shall make use of in our further considerations. Another special class of vertices are those with degree two and line-degree of two or more. This case occurs when two or more lines proceed along the same sequence of vertices and arcs for some stops. For example, in main streets quite a few lines will pass through usually with more than one stop. If people want to change lines they can then do this at any stop which belongs to both lines, but for simplicity we shall assume that people change either at the stop where the two lines meet or at the stop were they separate, but not at a stop between those two. This assumption will not influence any analysis of the following chapters.

We can now divide the vertices in X into two subsets:
The set I of vertices where interchanges (from one line to another) occur and the set Q of vertices that do not belong to I.

Like in other transportation networks, it is now interesting to know the shortest path between two vertices in the network $\bar{G} = (X,F)$. But, although all vertices are reachable from any other vertex, in many cases people will have to change lines to reach their destination. Because on each line vehicles run in some frequency (usually between 5 to 15 minutes) to change a line also means to have to wait for the next vehicle of the other line. Therefore the travel time in an urban public transportation system is the sum of the transportation time along the used lines plus the sum of all waiting times that occur when waiting for a vehicle of a line. When talking about railway systems in chapter 4.5.,we could ignore the waiting time because the transportation time usually dominates the total travel time and therefore the waiting time can be neglected. But in an urban transportation network where distances are small,the waiting time might be even greater than the transportation time and cannot be neglected. To include waiting time in our network $\bar{G}$ means that a vertex is split into as many vertices as there are lines passing through this vertex and if the original vertex belongs to I,these new vertices are connected by arcs denoting the possible changes and the length of each arc denoting the average waiting time. This transformation is shown in the example of Fig.7.1.



a) Original network without waiting time. The number on each arc denotes the travel time



b) Network with waiting time on the double-lined arcs

Fig. 7.1.

Some remarks on Fig.7.1. are necessary: First, it is easy to see that, in order to compute the shortest paths in $\bar{G}$, the number of vertices and arcs has to be increased substantially. Secondly, the waiting time can be chosen different at all interchanges. Finally at the intersection of line 3 with line 2 and 1, only interchanges between 3 and 2 or 3 and 1 are possible and not between 2 and 1, because we assume that people change from 2 to 1 or vice versa at vertices a and d in Fig.7.1. a). Although the shortest paths can be computed in Fig.7.1. b) in principle (with Dijkstra's or Floyd's algorithm), we are seeking now for a procedure to decrease the number of vertices and arcs again, thus reducing computation time. Let us look to the vertices belonging to the set Q, where no interchanges occur . Let us assume that the shortest paths between all vertices in I in Fig.7.1.b), where interchanges occur, are known. Then, in order to find the shortest paths from a vertex q in Q to any other vertex x in X;one only has to find the nearest vertices to q that belong to I and also belong to the same line than does q - for each line to which q belongs there are two such vertices (except if q is a terminal,then there is only one such vertex). If the same is done for x,then the shortest path can be computed as the minimum over all possible paths from q to all nearest vertices in I on the same line, plus the shortest paths between those vertices and the nearest vertices in I on the same line as x; plus the length from these vertices to x. If x and q belong only to one line,then only four possible paths must be considered. If x and q belong to the same line,then the path length without interchange must be considered as well. Although this procedure sounds rather complicated,Ajzen & Rokeach (1974) report on good results with a slightly less general procedure than the one stated here, although they did not use Floyd's or Dijkstra's algorithm for the computation of the shortest paths. Therefore to solve the shortest paths problem on Fig. 7.1. b),only the shortest paths of the graph in Fig.7.2. (including only the vertices in I) have to be computed by Floyd's or Dijkstras's algorithm, out of which all other shortest paths can be found directly.

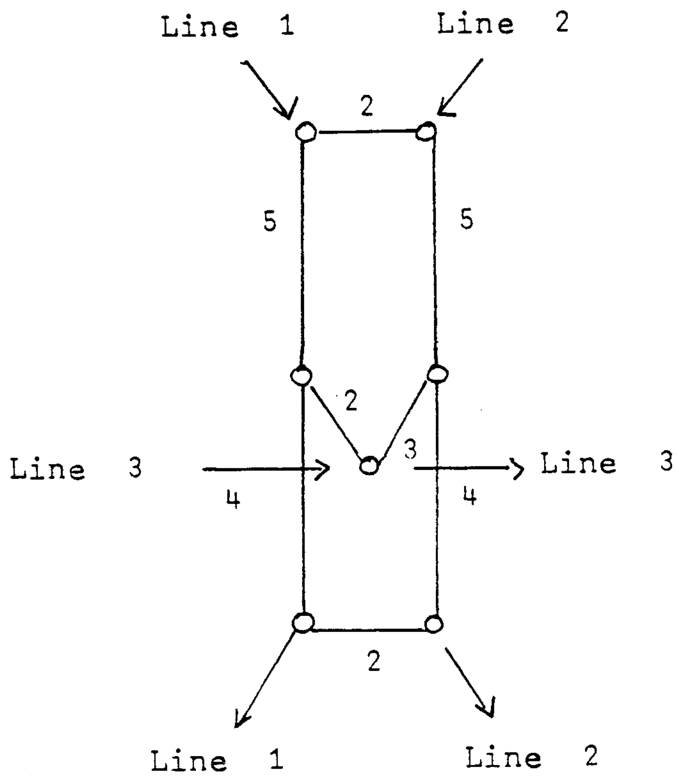


Fig.7.2.

Because changing lines is not only time consuming but also inconvenient - one might have to wait while raining or one might have to give up a seat and perhaps has to stand in the next vehicle - many people do not want to find the shortest path but the one with the least changes necessary. This minimum - change paths can be found completely the same way than the shortest paths if, instead of assigning the real waiting time to each arc denoting a possible change, one assigns the same high value a to those arcs, such that a is much greater than the transportation time. Then the shortest path will be one where changes occur as seldom as possible because any such change would increase the path length by the value a. Also, by dividing the path length by a and taking the integer part of it, it would immediatly give the number of necessary changes. We can therefore conclude that both problems, the shortest path and the minimum change in urban public transportation systems, can be found efficiently with the following algorithm.

<u>Algorithm for finding shortest paths or minimum changes in urban public transportation networks</u>

Let the nondirected network $\bar{G}$ = (X,F) denote the transportation network that is built up by the set of lines L. For each arc in F the travel time on this arc is given (it is assumed to be the same for any line $l \epsilon L$ using this arc) and the average waiting time for a vehicle for each line $l \epsilon L$.

<u>Step 1</u>:

Find the set of vertices $I \subset X$ where people change lines. These are all vertices where lines cross, meet or separate.

Let $Q = X-I$ be the set of vertices where people do not change.

Step 2:

Construct a new nondirected network $H=(Y,E)$ in the following way. For each $x_i \varepsilon I$ define a vertex y_{ij} if x_i belongs to line $j \varepsilon L$. The set of all y_{ij} being Y. Let E consist of the following arcs:

arc (y_{ij}, y_{ik}) if at vertex x_i people change from line j to line k and vice versa. The length of this arc is either the average waiting time for a vehicle of line j or k (if these are not equal then H must be directed) in case of shortest path problem or a large number a in case of minimum change problem.

arc (y_{ij}, y_{lj}) connecting vertices that belong to the same line $j \varepsilon L$. The length of this arc is the transportation time between x_i and x_l along line j.

For each vertex $q \varepsilon Q$ find the "nearest" vertices εY in the following way: If q belongs to line j, then find the closest vertex $x_i \varepsilon I$ that also belongs to line j in both possible directions to go along from q along line j. (If q is a terminal vertex of line j then, of course, one can only go along one direction). By this procedure each $q \varepsilon Q$ is assigned to one or two vertices y_{ij} and y_{lj} for each line j to which q belongs. Doing this for all lines to which q belongs, let the set of all vertices y_{ij} to which q is assigned to be denoted by $n(q) \subset Y$ and the transportation time from q to some $n(q)$ being the transportation time along line j to which both vertices belong.

For each vertex $x_i \varepsilon I$ let $n(x_i) \subset Y$ consist of all vertices y_{ij} and the transportation time between x_i and $n(x_i)$ be zero.

Step 3:

Find the shortest paths between all pairs of vertices on the network $H=(Y,E)$ with Floyd's algorithm.

Step 4:

For any pair of vertices x_i and $x_j \varepsilon X$ find the shortest path as the path with minimum length among all paths

$$x_i - n(x_i) - n(x_j) - x_j \qquad (7.3)$$

The lengths of all paths (7.3) can easily be computed because the lengths of the paths $n(x_i) - n(x_j)$ have been computed in Step 3 and the lengths of the paths $x_i - n(x_i)$ were stored in Step 2.

Note that in order to find the shortest path itself (and not only its length), only the shortest paths in network H have to be computed because the rest of the path for any pair of vertices x_i, x_j can be readily found out of (7.3).

```
C ... *** PROGRAM FOR FINDING SHORTEST PATHS IN PUBLIC
C ... *** TRANSPORTATION NETWORKS
C ... ***
C
C ... INPUT
C
C ... N       NUMBER OF VERTICES IN THE NETWORK
C ... C(L)    TRAVEL TIME ON ARC(I.J), L=IND(I.J.N)
C ... NL      NUMBER OF LINES
C ... NV(I)   NUMBER OF VERTICES BELONGING TO LINE I
C ... NVX(I.J) J-TH VERTEX OF LINE I, J=1,...NV(I), I=1,...NL
C ... WAIT    WAITING TIME IF A LINE HAS TO BE CHANGED
C
C ... OUTPUT
C
C ... NNX        SUM OVER ALL VERTICES WHERE PEOPLE CAN CHANGE
C                LINES MULTIPLIED BY THE NUMBER OF LINES TO WHICH
C                EACH SUCH VERTEX BELONGS TO
C ... H(L)       LENGTHS OF SHORTEST PATHS BETWEEN I AND J,
C                L=IND(I.J.NNX), WHERE NX(I.1) AND NX(I.2) DENOTES
C                THE VERTEX NUMBER AND THE LINE NUMBER, RESPECTIVELY
C ... NX(I.K)    SEE ABOVE, I=1,...NNX, K=1.2
C ... T(L)       SHORTEST PATHS BETWEEN I AND J, L=IND(I.J.NNX)
C                WHERE NX(I,1) AND NX(I.2) DENOTES THE VERTEX
C                NUMBER AND THE LINE NUMBER, RESPECTIVELY
C ... NQ(I,J,K) NQ(I.J,2) DENOTES A VERTEX NUMBER OF H(L) TO WHICH
C                VERTEX I IS ASSIGNED TO AND NQ(I,J,1) THE ASSOCIATED
C                TRAVEL TIME, SUCH THAT NX(NQ(I,J,2),1) DENOTE THE REAL
C                VERTEX NUMBER AND NX(NQ(I,J,2),2) THE LINE, I=1,...,N,
C                J=2,..,NQ(I,1,1)
C
      SUBROUTINE SHOPAT(N,C.NL,NV,NVX,WAIT,NNX,H,NX,T.NQ)
      INTEGER N,C(1),NL,NV(1),NVX(30,30).WAIT,T(1)
      INTEGER I(30).Q(30),NX(30,2),H(1).QX,NQ(30,20,2),NNX
      LOGICAL LOG
C
C ... STEP 1
C
      J=0
      DO 5 K=1,N
      JJ1=0
      JJ2=0
      JJ3=0
      DO 10 L=1,NL
      DO 15 M=1,NV(L)
      IF(NVX(L,M) .NE. K) GO TO 15
      IF(JJ2 .NE. 0) GO TO 20
      JJ2=K
      IF(M .GT. 1) JJ1=NVX(L,M-1)
      IF(M .LT. NV(L)) JJ3=NVX(L,M+1)
      GO TO 15
20    JJ4=0
      JJ5=0
      IF(M .GT. 1) JJ4=NVX(L,M-1)
      IF(M .LT. NV(L)) JJ5=NVX(L,M+1)
      IF((JJ1.EQ.JJ4 .OR. JJ1.EQ.JJ5) .AND. (JJ3.EQ.JJ4 .OR.
```

```
     X JJ3.EQ.JJ5)) GO TO 15
       J=J+1
       I(J)=K
       GO TO 5
15     CONTINUE
10     CONTINUE
5      CONTINUE
       IX=J
       J=0
       L=1
       DO 25 K=1,N
       IF(K .EQ. I(L)) GO TO 30
       J=J+1
       Q(J)=K
       GO TO 25
30     L=L+1
25     CONTINUE
       QX=J
C
C ... STEP 2
C
       J=0
       DO 35 K1=1,IX
       K=I(K1)
       DO 40 L=1,NL
       DO 45 M=1,NV(L)
       IF(NVX(L,M) .NE. K) GO TO 45
       J=J+1
       NX(J,1)=K
       NX(J,2)=L
       GO TO 40
45     CONTINUE
40     CONTINUE
35     CONTINUE
       NNX=J
       L=0
       DO 50 K=1,NNX
       DO 55 M=1,NNX
       L=L+1
       H(L)=2**34
       IF(K .EQ. M) GO TO 55
       IF(NX(K,1) .NE. NX(M,1)) GO TO 60
       H(L)=WAIT
       GO TO 55
60     IF(NX(K,2) .NE. NX(M,2)) GO TO 55
       H(L)=0
       MY=2**34
       LL=NX(K,2)
       KZ=0
65     KZ=KZ+1
       IF(NVX(LL,KZ).EQ.NX(K,1) .OR. NVX(LL,KZ).EQ.NX(M,1)) GO TO 70
       GO TO 65
70     KZ=KZ+1
       IF(KZ .GT. NV(LL)) GO TO 66
       LR=IND(NVX(LL,KZ),NVX(LL,KZ-1),N)
       H(L)=H(L)+C(LR)
       IF(NVX(LL,KZ).NE.NX(K,1) .AND. NVX(LL,KZ).NE.NX(M,1)) GO TO 70
```

```
      MY=MIN0(MY,H(L))
      H(L)=0
      GO TO 70
66    H(L)=MY
55    CONTINUE
50    CONTINUE
      DO 75 K=1,QX
      K1=Q(K)
      NQ(K1,1,1)=1
      DO 80 L=1,NL
      DO 85 M=1,NV(L)
      IF(K1 .NE. NVX(L,M)) GO TO 85
      LS=1
90    LT=NQ(K1,1,1)+1
      NQ(K1,LT,1)=0
      MX=M
95    MX=MX+LS
      IF(MX.LT.1 .OR. MX.GT.NV(L)) GO TO 100
      LR=IND(NVX(L,MX-LS),NVX(L,MX),N)
      NQ(K1,LT,1)=NQ(K1,LT,1)+C(LR)
      DO 105 K2=1,IX
      IF(NVX(L,MX) .LT. I(K2)) GO TO 95
      IF(NVX(L,MX) .NE. I(K2)) GO TO 105
      NQ(K1,1,1)=LT
      GO TO 101
105   CONTINUE
101   DO 102 KB=1,NNX
      IF(NX(KB,1).NE.NVX(L,MX) .OR. NX(KB,2).NE.L) GO TO 102
      NQ(K1,LT,2)=KB
      GO TO 100
102   CONTINUE
100   IF(LS .EQ. -1) GO TO 85
      LS=-1
      GO TO 90
85    CONTINUE
80    CONTINUE
75    CONTINUE
      DO 110 K=1,IX
      K1=I(K)
      NQ(K1,1,1)=1
      DO 115 L=1,NL
      DO 120 M=1,NV(L)
      IF(K1 .NE. NVX(L,M)) GO TO 120
      NQ(K1,1,1)=NQ(K1,1,1)+1
      LR=NQ(K1,1,1)
      NQ(K1,LR,1)=0
      DO 103 KB=1,NNX
      IF(NX(KB,1).NE.K1 .OR. NX(KB,2).NE.L) GO TO 103
      NQ(K1,LR,2)=KB
      GO TO 115
103   CONTINUE
120   CONTINUE
115   CONTINUE
110   CONTINUE
C
C ... STEP 3
C
```

```
      CALL SPII(NNX,H,T,LOG)
      DO 145 K=1,NNX
      L=IND(K,K,NNX)
      T(L)=K
145   H(L)=0
      RETURN
      END
```

7.3. Traffic assignment

Traffic assignment in this context can mean two things: Assigning people to lines and/or arcs or assigning lines to the underlying transportation network. The latter we shall discuss in chapter 7.4. So, our problem is to find the number of people travelling along a specific line or arc, given the network $\bar{G} = (X,F)$ of lines and given the trip matrix $T=(t_{ij})$, stating the number of people travelling between pairs of vertices x_i and x_j. So far, very little attention has been paid to this problem and only Chriqui & Robillard (1975) have treated it in more detail.

Although Wardrop's principle on the normative assignment can be accepted here (including, of course, waiting time), the descriptive assignment principle will not remain true. As already mentioned, not all people really minimize travel time, but rather try to minimize changes, and some behave according to a weighted sum of transportation and waiting time. Unfortunately, no empirical results seem to exist on the behaviour of people in urban public transportation networks. Therefore, any descriptive assignment principle must be tested on its validity. Nothing is reported on this matter in Chriqii & Robillard (1975).

As a first approach to this problem, we suggest to assume that some fraction, say one third, of all passengers going from any vertex to another (i.e. $t_{ij}/3$) behaves according to time minimization, another one third behaves according to change minimization and one third gives weight to each change as being two or three times the average waiting time and minimizes travel time according to this weighted waiting time.

Although we shall make use of the just stated descriptive assignment "principle", we are quite aware of the fact that a lot of empirical investigations need to be undertaken to find the correct descriptive assignment principle.

Using this descriptive assignment approach, there are five different models that could be used. First, where no arc capacities exist and the arc costs are constant on the network $\bar{G} = (X,F)$. Of course, in contrary to car traffic assignment, normative and descriptive assignment is not equal, because some people do not minimize travel time in the descriptive assignment. The next models would be with constant arc costs again but with arc capacities and, finally, with no arc capacities but with costs increasing with arc flow.

The last two models can also be applied if the arcs have no capacity constraints but a specific line on an arc, therefore each line can have a different capacity on the same arc. Instead of an arc cost depending on the flow, one can also assume different arc costs for each line depending on the flow on an arc belonging to a specific line.

a) No arc capacities and constant arc costs

In this case all people travel along the path they wish to take according to their objective. Therefore the algorithm of chapter 7.2. can be used directly for the normative assignment. In case of the descriptive assignment the shortest path problem has to be solved for three different waiting times. Assuming that $t_{ij}/3$ people use each of the three different found shortest paths, the flow on each arc can be computed as the sum of all people using the arc by one of the three paths (which may be, of course, equal). The flow on each arc of each line is not completely defined, because if more than one line proceedsalong the same arcs, people might use both of them along these arcs, thus the flow assignment to lines is not unique.

```
C ... *** DESCRIPTIVE ASSIGNMENT IN PUBLIC TRANSPORTATION
C ... *** NETWORKS WITH CONSTANT ARC COSTS AND WITHOUT ARC CAPACITIES
C ... ***
C
C ... INPUT
C
C ... N      NUMBER OF VERTICES IN THE NETWORK
C ... C(L)   TRAVEL TIME ON ARC(I,J), L=IND(I,J,N). IT IS ASSUMED
C            THAT TRAVEL TIME ON ARC(I,J) IS EQUAL TO THE ONE
C            ON ARC(J,I). IF C(L)=0 NO ARC EXISTS.
C ... NL     NUMBER OF LINES
C ... NV(I) NUMBER OF VERTICES BELONGING TO LINE I
C ... NVX(I,J) J-TH VERTEX OF LINE I, J=1,..,NV(I), I=1,..,NL
C ... WAIT   AVERAGE WAITING TIME FOR A BUS (TRAM) WHICH IS HALF
C            THE TIME INTERVAL BETWEEN BUSES (TRAMS)
C ... G(L)   NUMBER OF PEOPLE WHO WANT TO TRAVEL FROM VERTEX I
C            TO J, L=IND(I,J,N). IT IS ASSUMED THAT G(L)=G(K),
C            K=IND(J,I,N).
C
C ... OUTPUT
C
C ... TOT    TOTAL TRANSPORTATION TIME OF ALL PASSENGERS
C            INCLUDING THE WAITING TIME FOR A BUS (TRAM) AT THE
C            BEGINNING OF THE JOURNEY AND WHEN CHANGING LINES
C ... FL(L) FLOW FROM VERTEX NX(I,1) BELONGING TO LINE NX(I,2)
C            TO VERTEX NX(J,1) BELONGING TO LINE NX(J,2), WHERE
C            L=IND(I,J,NNX) AND I,J=1,...,NNX
C            FL(K), K=IND(I,I,NNX), DENOTES THE FLOW THROUGH K
C            WITHOUT CHANGING LINE.
C ... NX(I,K) SEE ABOVE
C ... NNX    SEE ABOVE
C ... T(L)    LENGTH OF SHORTEST PATH BETWEEN VERTEX I AND J,
C             L=IND(I,J,N)
C
      SUBROUTINE DESCRI(N,C,NL,NV,NVX,WAIT,G,TOT,FL,NX,NNX,T)
      INTEGER N,C(1),NL,NV(1),NVX(30,30),WAIT,G(1),TOT,FL(1),NX(30,2)
      INTEGER NNX,H(900),T(1),NQ(30,20,2),D(900)
C
C ... COMPUTING THE LENGTHS OF THE SHORTEST PATHS
C
      CALL SHOPAT(N,C,NL,NV,NVX,WAIT,NNX,H,NX,D,NQ)
      M=NNX*NNX
      TOT=0
      DO 5 I=1,M
5     FL(I)=0
      DO 125 K=1,N
      L=IND(K,K,N)
      T(L)=0
      DO 130 M=1,N
      IF(K .EQ. M) GO TO 130
      L=IND(K,M,N)
      T(L)=2**34
      KY=NQ(K,1,1)
      MY=NQ(M,1,1)
      DO 135 I1=2,KY
      DO 140 I2=2,MY
```

```
      J1=IND(NQ(K,I1,2),NQ(M,I2,2),NNX)
      J2=NQ(K,I1,1)+NQ(M,I2,1)+H(J1)+WAIT
      IF(J2 .GE. T(L)) GO TO 140
      T(L)=J2
      JB=NQ(K,I1,2)
      JC=NQ(M,I2,2)
140   CONTINUE
135   CONTINUE
      TOT=TOT+T(L)*G(L)
      LZ=IND(JC,JC,NNX)
      IF(JB .EQ. JC) FL(LZ)=FL(LZ)+G(L)
      IF(JB .EQ. JC) GO TO 130
      JZ=JC
      IZ=JC
10    KZ=JZ
      JZ=IZ
      IZ=IND(JB,JZ,NNX)
      IZ=D(IZ)
      LZ=IND(JZ,JZ,NNX)
      IF(NX(IZ,2).EQ.NX(JZ,2) .AND. NX(JZ,2).EQ.NX(KZ,2))
     XFL(LZ)=FL(LZ)+G(L)
      LZ=IND(IZ,JZ,NNX)
      FL(LZ)=FL(LZ)+G(L)
      IF(IZ .NE. JB) GO TO 10
      LZ=IND(IZ,IZ,NNX)
      IF(NX(IZ,2) .EQ. NX(JZ,2)) FL(LZ)=FL(LZ)+G(L)
130   CONTINUE
125   CONTINUE
C
C ... COMPUTING THE LENGTHS OF THE PATHS WITH MINIMUM CHANGES
C
      MWAIT=2**30
      CALL SHOPAT(N,C,NL,NV,NVX,MWAIT,NNX,H,NX,D,NQ)
      NNX2=NNX*NNX
      DO 15 I=1,NNX2
      J=H(I)/MWAIT
15    H(I)=H(I)-J*MWAIT+J*WAIT
      DO 225 K=1,N
      DO 230 M=1,N
      IF(K .EQ. M) GO TO 230
      L=IND(K,M,N)
      MTL=2**34
      KY=NQ(K,1,1)
      MY=NQ(M,1,1)
      DO 235 I1=2,KY
      DO 240 I2=2,MY
      J1=IND(NQ(K,I1,2),NQ(M,I2,2),NNX)
      J2=NQ(K,I1,1)+NQ(M,I2,1)+H(J1)+WAIT
      IF(J2 .GE. MTL) GO TO 240
      MTL=J2
      JB=NQ(K,I1,2)
      JC=NQ(M,I2,2)
240   CONTINUE
235   CONTINUE
      TOT=TOT+MTL*G(L)
      LZ=IND(JC,JC,NNX)
      IF(JB .EQ. JC) FL(LZ)=FL(LZ)+G(L)
```

```
      IF(JB .EQ. JC) GO TO 230
      JZ=JC
      IZ=JC
20    KZ=JZ
      JZ=IZ
      IZ=IND(JB,JZ,NNX)
      IZ=D(IZ)
      LZ=IND(JZ,JZ,NNX)
      IF(NX(IZ,2).EQ.NX(JZ,2) .AND. NX(JZ,2).EQ.NX(KZ,2))
     XFL(LZ)=FL(LZ)+G(L)
      LZ=IND(IZ,JZ,NNX)
      FL(LZ)=FL(LZ)+G(L)
      IF(IZ .NE. JB) GO TO 20
      LZ=IND(IZ,IZ,NNX)
      IF(NX(IZ,2) .EQ. NX(JZ,2)) FL(LZ)=FL(LZ)+G(L)
230   CONTINUE
225   CONTINUE
      TOT=TOT/2.
      DO 25 I=1,NNX2
25    FL(I)=FL(I)/2.
      RETURN
      END
```

b) Arc capacities or non constant arc costs:

Given the network $\bar{G} = (X,F)$ and the set of lines L. In order to include waiting costs at vertices, the network $\bar{G}$ has to be expanded in the following way (which is similar to Step 2 of the algorithm of chapter 7.2.):

Let the new network be $B=(Z,K)$. For each $x_i \varepsilon X$ define a vertex $z_{ij} \varepsilon Z$, if x_i belongs to line $j \varepsilon L$. The set of all z_{ij} being Z. Let K consist of the following arcs:

arc (z_{ij}, z_{lj}), if arc $(x_i, x_l) \varepsilon F$ with the same arc costs.

arc (z_{ij}, z_{il}), if people may change from line j to line l at vertex x_i. The arc cost is the waiting time or some value greater than the waiting time.

Because each arc of $\bar{G}$ appears now more than once in B, arc capacities d_{il} in $\bar{G}$ are transferred into capacity constraints over the sum of flows, being

$$\sum_j \text{arcflow } (z_{ij}, z_{lj}) \leq a_{il}$$

No specific algorithm is known for this problem, but the general simplex-algorithm can be used in case of normative assignment.

If the arc costs depend on the flow in $\bar{G}$, this is transferred in B into a problem where the arc costs on arc (z_{ij}, z_{lj}) depend on the sum of flows $\sum_j \text{arcflow}(z_{ij}, z_{lj})$. Again no specific algorithm is known, but an algorithm for solving optimization problems with linear constraints and a convex objective could be used, in principle, for the normative assignment (although only on small problems). For larger normative assignment problems the algorithm of chapter 3.3.2. can be adapted. Again the assignment to lines will not be unique. For the descriptive assignment problem no algorithm exists so far.

c) Line capacities or non constant line-arc costs:

In this case each arc in B=(Z.K) has his own capacity or cost and the algorithms of chapter 3. can be applied directly. In case of the descriptive assignment no algorithm exists so far.

Concluding this chapter,we remark that descriptive assignment can only be found for the simplest model yet. Which of the models fit best to reality can hardly be answered in general.

7.4. Route planning

In the last two chapters we assumed that the set of lines L is given. However, as this set of lines only has to satisfy the feasibility conditions (i.e. all vertices of the network G=(X,A) have to belong to at least one line $l \varepsilon L$ and the network $\bar{G}$ = (X,F) built by the set of lines L is strongly connected), a number of feasible sets of lines will exist.

Thus, one may introduce some objective according to which the best set of lines is chosen. Generally spoken, there are two meaningful approaches to the problem of choosing a suitable set of lines: Either the service level offered to the passengers is given and the objective is to minimize the operating costs or, vice versa, the operating costs are restricted and the service level is to be maximized. Here we shall concentrate on the latter problem, because it seems to be the usual way in practice to deal with the problem. Service level can easily be measured by the total transportation time of the passengers as found by a suitable descriptive assignment: service level is good if total transportation time is low. For measuring the operating costs,we shall adapt the approach suggested by Silman et al. (1974), namely the number of buses/or trams used at the same time to travel along the lines. This is suggestive, because the fixed and variable costs of all the buses and/or trams together represent the largest part of the total operating costs (of course the first costs not only include the expenditure for buying a vehicle,but also the salaries for the drivers).For a given set of lines the number of buses (trams) determines the frequencies and thus the waiting time. Therefore, the more buses used on a given set of lines, the less the total transportation time will be.

Our problem can now be formulated as follows:

Given the transportation network G=(X,A), where X represents urban areas (or stops) that have to be served and A represents possible streets (or rails) that can be used. To each arc in A the transportation time on this arc is given. The transportation demand matrix T (i.e. the number of passengers) from vertex x_i to x_j ($x_i,x_j \varepsilon X$) is assumed to be known and constant. The number of buses or trams is restricted by some number N. Then a feasible set of lines L is to be found,such that the total transportation time according to some descriptive assignment (with waiting times defined by the number of buses N) is minimized.

Note that this model does not include the possibility that not every vehicle can use all arcs in A. This can occur if the transportation system consists of both, trams running along rails and buses running along streets. Then the model only applies if the set of streets and the set of rails are the same (i.e. every vertex can be served by bus and by tram) or if the set of bus lines is chosen independently from the set of tram lines.

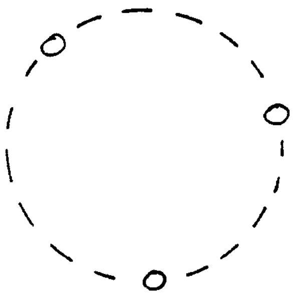
Complex as the stated problem is, only a heuristic algorithm seems appropriate. The algorithm we present here completely differs from those presented by Lampkin et al. (1967), Silman et al.(1974) and Hoidn (1977). Its advantages to the already published approaches are:

- The algorithm is independent of the particular descriptive assignment procedure chosen. Any descriptive assignment algorithm may be used.
- The algorithm proceeds to find first a feasible set of lines and then iteratively changes this set while reducing the total transportation time in every step. Therefore the algorithm produces a number of feasible sets of lines to be compared by the transportation planner.
- The transportation planner can decide if the set of terminal vertices chosen for initialization of the algorithm is fixed or may be altered by the algorithm.
- Ring lines (i.e. lines that form a cycle) can be considered by the algorithm as well.
- Existing lines can easily be taken into acount.

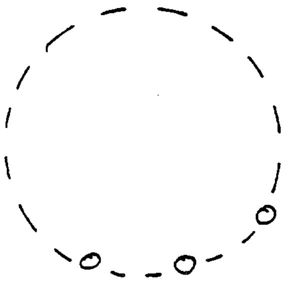
As the size of the network G=(X,A) determines the size of the problem and therefore the costs (especially the computer time) to find the solution, the construction of this network is of great importance. From the view of minimizing the solution costs it should be as small as possible, from the view of the transportation planner who wants a detailed answer, it should be as large as possible. In practice, not every stop that should be served will be included into the set of vertices X. Rather, the urban region should be divided into areas, each of which should be served by at least one line. The size of these areas usually varies and will be larger where the population density is low (i.e. in suburbian areas) and smaller nearby the center of the city. As already mentioned in chapter 7.1.,the assumption that the demands $T=(t_{ij})$ are constant will only remain true over a rather short time period. The only way of handling expected changes in T is by performing sensitivity analysis and to find the actual demand T every year. In fact, it is one of the main handicaps of every urban public transportation system that estimating T is rather expensive and therefore not done frequently. So the recent changes in transportation demands cannot be considered and individual car traffic becomes more and more attractive. Another simplification of the model is the fact that T is assumed constant during day. In reality, the demand is quite different at every hour of the day. To deal with this problem an average demand has to be used or the maximum demand that occurs during rush-hours. It's no use to find optimal sets of lines for different hours of the day, because the organizational problems would become enourmous and also no passenger would be interested in having different lines at different times. Finally, the model does not deal with varying travel and waiting times, which also change during time because of congestions due to individual car traffic. Thus the transportation times along arcs are supposed to be an average transportation time.

As already mentioned the algorithm is divided into two parts. First a good feasible, initial set of tours is created and second, the set of lines is changed to reduce total transportation time. To find such a good set of lines, only heuristic rules can apply because the quality of a set of lines cannot be measured by descriptive assignment as long as this set is not feasible. Let us denote the total travel time found by a descriptive assignment for a given network G=(X,A), transportation demand T and number of buses N with D(L) thus being a function of the set of lines. L. In order to run the initialization algorithm, some additional data is required, namely

- an even number of vertices in X to become terminals of some line. Each vertex that has to remain a terminal is marked as fixed terminal, while the other terminals may become non-terminals in the course of the algorithm.
- for each ring line that should be introduced, three vertices that should belong to a particular ring line. Again these vertices may be permanent or temporary members of the ring line. Preferably, these three vertices should approximately mark the size of the cycle, as shown in Fig.7.3.a). A choice like the one shown in Fig.7.3. b) should be avoided.



a)



b)

Fig.7.3.

Algorithm to find a good, feasible set of lines

Step 1:

Compute the shortest paths between all pairs of vertices in X. Let g_{ij} denote the length of the shortest path between x_i and x_j.

Let $Y \subset X$ denote the set of vertices which are not yet assigned to a line $l_i \varepsilon L$. Let v_{ij} denote the number of vertices which belong to the shortest path between x_i and x_j. Let Q be the set of terminals not yet used. Let $G=(X,A)$ be the given network.

Step 2:

In order to create lines that combine terminals along shortest paths and that include as many vertices as possible (to avoid line changing), choose $x_i, x_j \varepsilon Q$ such that

$$v_{ij} = \max_{x_k, x_l \varepsilon Q} v_{kl} \quad . \tag{7.4}$$

If more than one such pair exists choose the one with minimum distance g_{ij}. Mark the shortest path from x_i to x_j as a new line $l \varepsilon L$ and delete all vertices belonging to l from the set Y and delete x_i and x_j from Q.
Repeat Step 2 until $Q=\emptyset$.

Step 3:

Combine the three vertices belonging to the same ring line by a cycle that is equal to the shortest path between every pair of the three vertices. Mark this cycle as a new line $l \varepsilon L$ and delete all vertices that belong to l from Y. Repeat Step 3 for all ring lines given.

Step 4:

If Y is empty, go to Step 6.
Otherwise go to Step 5.

Step 5:

Compute

$$c = \min_{i,j,k} (g_{ij}+g_{ik}-g_{jk}) \qquad (7.5)$$

where $x_i \varepsilon L$, $x_j, x_k \varepsilon l \varepsilon L$
and arc (i,j), arc (j,k), arc$(k,i) \varepsilon A$.
If a feasible (and therefore an optimal) solution of (7.5) exists, include x_i into line l between vertices x_j and x_k, delete x_i from Y and go to Step 4.
If no feasible solution to (7.5) exists, set Q=Y and find a new line as stated in Step 1 in case Y contains at least two vertices. If Y contains only one vertex, create a new line between this vertex and the nearest vertex belonging to some line $l \varepsilon L$.
Delete the vertices now belonging to a line from Y and go to Step 4.

Step_6:

Prove, if the network $\bar{G} = (X,F)$ created by the set of lines L is strongly conncted. If it is, then a feasible set of lines L has been found. Stop.
If not, identify the set of vertices $V_i \subset X$ ($\bigcup_i V_i = X$, $V_i \cap V_j = \emptyset$), the members of one set being mutually reachable . Set the members of some V_i into the set Y and combine V_i with $X-V_i$ in the same way as stated in Step 5.
Repeat Step 6 until $\bar{G}$ is strongly connected.

Note that the algorithm never fails to find a feasible set of lines, but eventually creates lines by itself with terminals not stated in Q.

```
C ... *** PROGRAM FOR FINDING A "GOOD" AND FEASIBLE SET OF
C ... *** LINES FOR AN URBAN PUBLIC TRANSPORTATION SYSTEM
C ... ***
C
C ... INPUT
C
C ... N       NUMBER OF VERTICES
C ... C(L)    TRAVEL TIME ON ARC(I,J), L=IND(I,J,N). C(L)=0 DENOTES
C             THAT THIS ARC DOES NOT EXIST. IT IS ASSUMED THAT
C             C(L)=C(M), M=IND(J,I,N). IT IS FURTHER ASSUMED THAT EVERY
C             EXISTING ARC(I,J) IS THE SHORTEST PATH BETWEEN VERTICES
C             I AND J.
C ... QX      NUMBER OF PREFIXED TERMINALS. QX MUST BE EVEN.
C ... Q(I)    TERMINAL VERTICES, I=1,..,QX<N
C ... RX      NUMBER OF WANTED RING LINES (CYCLES)
C ... R(I,K)  R(I,1),..,R(I,3) DENOTE 3 VERTICES BELONGING TO CYCLE I,
C             I=1,..,RX, K=1,..,3
C
C ... OUTPUT
C
C ... NL      NUMBER OF LINES
C ... NV(I)   NUMBER OF VERTICES BELONGING TO LINE I, I=1,..,NL
C ... NVX(I,J) J-TH VERTEX OF LINE I, J=1,..,NV(I), I=1,..,NL
C
      SUBROUTINE FEASIB(N,C,QX,Q,RX,R,NL,NV,NVX)
      INTEGER N,C(1),QX,Q(1),RX,R(10,3),NL,NV(1),NVX(30,30)
      INTEGER Y(40),V(1600),VV(1600),G(1600),D(1600)
      LOGICAL LOG,ICAL
C
C ... STEP 1
C
      N2=N*N
      DO 20 I=1,N2
      G(I)=C(I)
      V(I)=0
20    IF(C(I) .GT. 0) V(I)=1
      DO 10 I=1,N
10    Y(I)=I
      CALL SPII(N,G,D,LOG)
      N1=N-1
      DO 15 I=1,N1
      I1=I+1
      DO 16 J=I1,N
      L=IND(I,J,N)
      V(L)=1
      IB=J
17    IB=IND(I,IB,N)
      IB=D(IB)
      V(L)=V(L)+1
      IF(IB .NE. I) GO TO 17
      L1=IND(J,I,N)
16    V(L1)=V(L)
15    CONTINUE
      NL=0
      NQX=QX
      LOG=.FALSE.
```

```
      ICAL=.FALSE.
C
C ... STEP 2
C
2     IF(NQX .LE. 1) GO TO 3
      M=0
      MG=2**34
      MQX=QX-1
      DO 25 I=1,MQX
      I1=I+1
      IA=Q(I)
      IF(IA .EQ. 0) GO TO 25
      DO 30 J=I1,QX
      JA=Q(J)
      IF(JA .EQ. 0) GO TO 30
      L=IND(IA,JA,N)
      IF(V(L) .LT. M) GO TO 30
      IF(V(L).EQ.M .AND. MG.LE.G(L)) GO TO 30
      MG=G(L)
      M=V(L)
      IB=I
      JB=J
30    CONTINUE
25    CONTINUE
      IA=Q(IB)
      Q(IB)=0
      JA=Q(JB)
      Q(JB)=0
      NQX=NQX-2
      NL=NL+1
      NV(NL)=1
      NVX(NL,1)=JA
      I=JA
40    I=IND(IA,I,N)
      I=D(I)
      NV(NL)=NV(NL)+1
      NVX(NL,NV(NL))=I
      IF(I .NE. IA) GO TO 40
      DO 70 I=1,NV(NL)
      J=NVX(NL,I)
70    Y(J)=0
      IF(LOG) GO TO 4
      GO TO 2
C
C ... STEP 3
C
3     IF(RX .EQ. 0) GO TO 4
      DO 80 I=1,RX
      NL=NL+1
      NV(NL)=1
      NVX(NL,1)=R(I,1)
      Y(R(I,1))=0
      IIA=1
      JJA=0
85    IIA=IIA+1
      JJA=JJA+1
      IF(IIA .GT. 3) IIA=1
```

```
      IA=R(I,IIA)
      JA=R(I,JJA)
      IB=JA
90    IB=IND(IA,IB,N)
      IB=D(IB)
      NV(NL)=NV(NL)+1
      NVX(NL,NV(NL))=IB
      Y(IB)=0
      IF(IB .NE. IA) GO TO 90
      IF(IIA .NE. 1) GO TO 85
80    CONTINUE
C
C ... STEP 4
C
4     IF(ICAL) GO TO 6
      DO 95 I=1,N
      IF(Y(I) .GT. 0) GO TO 5
95    CONTINUE
      GO TO 6
C
C ... STEP 5
C
5     MC=2**34
      DO 100 I=1,N
      IF(Y(I) .EQ. 0) GO TO 100
      DO 105 JJ=1,N
      IF(Y(JJ) .NE. 0) GO TO 105
      L=IND(I,JJ,N)
      IF(C(L) .EQ. 0) GO TO 105
      MG=G(L)
      JA=JJ
      DO 110 J=1,NL
      DO 115 K=1,NV(J)
      IF(NVX(J,K) .NE. JA) GO TO 115
      IF(K .EQ. 1) GO TO 120
      K1=NVX(J,K-1)
      L=IND(I,K1,N)
      IF(C(L) .EQ. 0) GO TO 120
      LA=IND(K1,JA,N)
      LC=MG+G(L)-G(LA)
      IF(MC .LE. LC) GO TO 120
      MC=LC
      MX=K-1
      MY=I
      MZ=J
120   IF(K .EQ. NV(J)) GO TO 110
      K1=NVX(J,K+1)
      L=IND(I,K1,N)
      IF(C(L) .EQ. 0) GO TO 110
      LA=IND(K1,JA,N)
      LC=MG+G(L)-G(LA)
      IF(MC .LE. LC) GO TO 110
      MC=LC
      MX=K
      MY=I
      MZ=J
      GO TO 110
```

```
115     CONTINUE
110     CONTINUE
105     CONTINUE
100     CONTINUE
        IF(MC .EQ. 2**34) GO TO 125
        Y(MY)=0
        NV(MZ)=NV(MZ)+1
        K=MX+2
        DO 130 I=K,NV(MZ)
        J=NV(MZ)-I+K
130     NVX(MZ,J)=NVX(MZ,J-1)
        NVX(MZ,MX+1)=MY
        GO TO 4
125     NQX=0
        DO 135 I=1,N
        IF(Y(I) .EQ. 0) GO TO 135
        NQX=NQX+1
        Q(NQX)=I
135     CONTINUE
        IF(ICAL) GO TO 136
        QX=NQX
        LOG=.TRUE.
        IF(QX .GE. 2) GO TO 2
136     M1=2**34
        M2=2**34
        IM1=0
        DO 140 I=1,N
        IF(Y(I) .NE. 0) GO TO 140
        L=IND(I,Q(NQX),N)
        IF(C(L) .EQ. 0) GO TO 140
        IF(C(L) .GE. M1) GO TO 145
        M2=M1
        M1=C(L)
        IM2=IM1
        IM1=I
        GO TO 140
145     IF(C(L) .GE. M2) GO TO 140
        M2=C(L)
        IM2=I
140     CONTINUE
        IF(IM1 .NE. 0) GO TO 141
        NQX=NQX-1
        GO TO 136
141     NL=NL+1
        NV(NL)=2
        NVX(NL,1)=IM1
        NVX(NL,2)=Q(NQX)
        IF(IM2 .EQ. 0) GO TO 6
        NV(NL)=3
        NVX(NL,3)=IM2
C
C ... STEP 6
C
6       DO 149 I=1,N2
149     V(I)=2**30
        DO 150 I=1,NL
        DO 155 J=2,NV(I)
```

```
      L=IND(NVX(I,J),NVX(I,J-1),N)
      V(L)=1
      L=IND(NVX(I,J-1),NVX(I,J),N)
155   V(L)=1
150   CONTINUE
      CALL SPII(N,V,VV,LOG)
      DO 160 I=2,N
      IF(V(I) .LT. 2**30) GO TO 160
      GO TO 165
160   CONTINUE
      RETURN
165   Y(1)=1
      DO 170 I=2,N
      Y(I)=0
      IF(V(I) .LT. 2**30) Y(I)=I
170   CONTINUE
      ICAL=.TRUE.
      GO TO 5
      END
```

Of course, this algorithm can also be used if part of the lines are already given, in case the transportation system already exists and should be expanded only, including new areas of the city. One only has to exclude all vertices belonging to existing lines from Y in Step 1 and store all existing lines in L.

Having now found a feasible set of lines, another algorithm is applied to improve this set to minimize the total transportation time. On this purpose, not only the transportation time along an arc is needed but also the waiting time for changing. In fact, as long as the bus scheduling for each line has not been done, waiting times are not really defined. We therefore make the assumption that the buses (N in total) are assigned to each line, such that bus frequencies on each line are the same and that the average waiting time is half the time interval between two buses on the same line. Let r(L) be the sum of the travel times over all lines $l \varepsilon L$, then

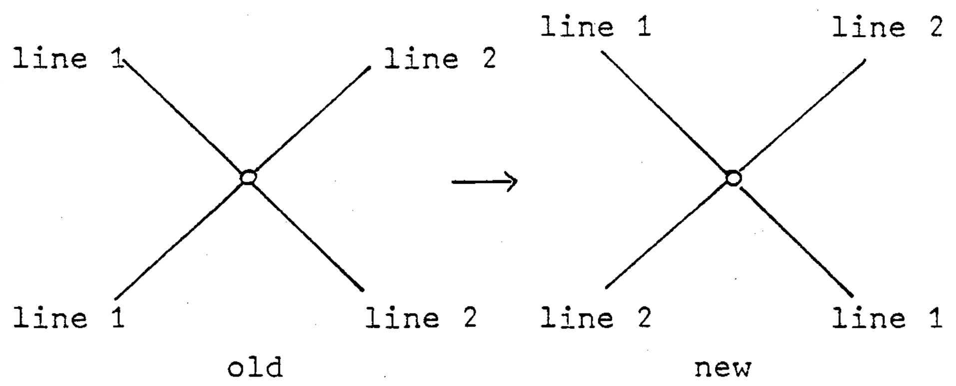
$$\frac{1}{\text{bus frequency}} = \text{bus time interval} = \frac{2.r(L)}{N}$$

$$\text{waiting time} = \frac{r(L)}{N} \qquad (7.6.)$$

Unfortunately, the waiting time as given in (7.6) has two shortcomings. The first being the fact that because only an integer number of buses can be assigned to each line, the frequency on each line cannot be exactly the same. Secondly, if a person can use two lines, because both travel to the same vertex along the same arcs, then the waiting time will of course be shorter, unless buses of different lines appear at the same time. To overcome these problems would involve a much more complicated descriptive assignment procedure than the one given in chapter 7.3., therefore we shall use the waiting time of (7.6) as an approximation of the real one.

The idea of the following algorithm now is to search for changes of lines - such that the set of lines remains feasible - according to heuristic rules that indicate a possible improvement of the descriptive assignment. If a promising change is found, it is performed and the descriptive assignment computed for this new set of lines. If this set of lines turns out to be better than the old one,it is accepted and the search procedure starts again until no improvement can be found any more. The possible changes we are considering are

- New combination of terminals by exchanging parts of lines at an intersection vertex, i.e.



This exchange is performed to reduce the number of people who have to change lines.

- Including a vertex that is close to a line, if transportation demand between this vertex and the vertices on the line is high.
- Excluding a vertex from a line that is already served by another line, if transportation demand between this vertex and the other vertices on the line is low, in order to reduce the length of this line (which results in lower waiting times because $r(L)$ is reduced).
- **Combining one line with part of another line.**

<u>Algorithm for improving a feasible set of lines:</u>

<u>Step 1:</u>

Set $D(\bar{L})=\infty$ and I=2.

<u>Step 2:</u>

Compute the waiting time r(L)/N, where L is the new set of lines. Find descriptive assignment with this waiting time - resulting in the total transportation time D(L).

If $D(L)<D(\bar{L})$ then accept the new set of lines L, set $\bar{L}=L$ and go to Step 3.

If $D(L)\geq D(\bar{L})$ and I=5, Stop.

If $D(L)\geq D(\bar{L})$ and I<5, go to Step (I+1).

<u>Step 3:</u>

Set I=3.

Consider all vertices, where people can change lines (the set I as defined in chapter 7.2).

Let vertex $i\varepsilon I$ belong to line l and k $(l,k\varepsilon L)$. Let i_l and i_k denote vertex i on line l and on line k respectively. Let f_a be the flow from i_l to i_k and f_b the flow from i_k to i_l. Finally let f_l be the flow through vertex i that remains on line l (does not change line at i) and let f_k be the flow through vertex i that remains on line k. The situation is pictured in Fig.7.4.

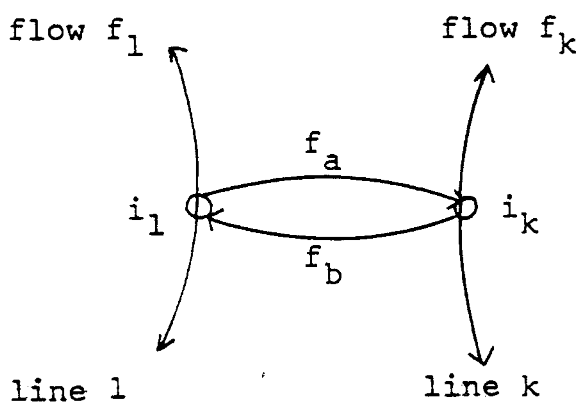


Fig.7.4

Among all vertices in I find the one for which

$$f_a + f_b - f_l - f_k > 0. \qquad (7.7)$$

If more than one such vertex exists, choose the one for which (7.7) is maximum.
If no vertex in I exists for which (7.7) holds, go to Step 4.
Combine the two lines l and k the way shown in Fig.7.5. Out of the two possibilities given in Fig.7.5b) and Fig.7.5c) choose the one with smaller objective value of the descriptive asssignment D(L). Go to Step 2.

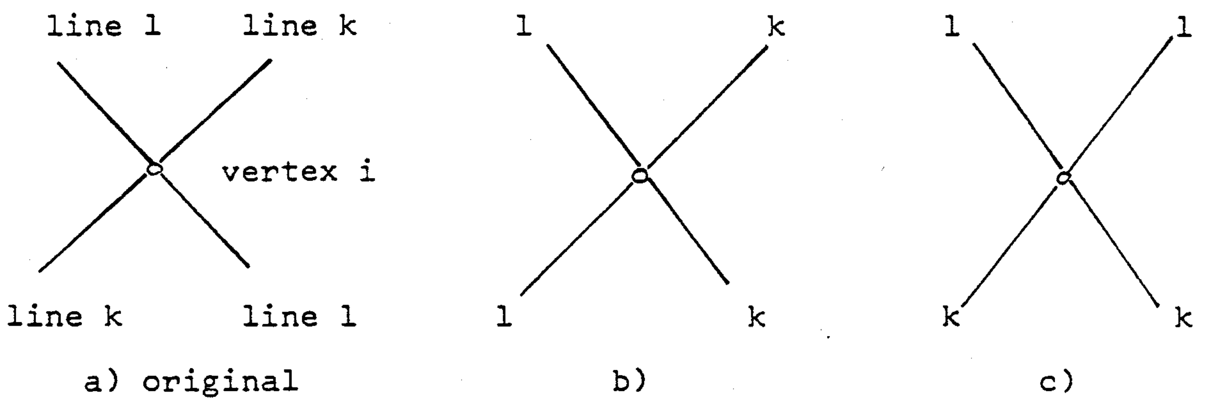


Fig.7.5

Step 4:

Set I=4.
For each pair of vertices $x_i, x_j \varepsilon X$ that do not belong to the same line let b_{ij} be the difference of the length of line l to which x_j belongs, if x_i is included into line l or if it is not (the way x_i is included is stated in Step 5 of the algorithm for finding a feasible set of lines). Let t_i^l be the amount of people travelling between x_i and all vertices on line l.
Find vertices $x_i, x_j \varepsilon X$ for which

$$c = \min_{\substack{x_i \notin l \varepsilon L \\ x_j \varepsilon l \varepsilon L}} (t_i^l / b_{ij}). \qquad (7.8)$$

If no pair of vertices exists such that x_j can be included into some line l go to Step 5. If (7.8) is optimal for vertex x_m to be included into line h, include x_m and go to Step 2.

Step_5:

Set I=5.
Find the set of vertices P that belong to at least two lines and that do not lie on the shortest path in G between the two neighbour vertices of the vertex on line 1. If P is empty, Stop - no further improvement can be made.
If P is not empty find the vertex x in P for which the flow of people changing lines at vertex x plus the flow of people between x and the other vertices of line L (to which x belongs) is minimum. Delete x from line 1 and go to Step 2.

```
C ... *** PROGRAM FOR IMPROVING A FEASIBLE SET OF BUS (TRAM) LINES
C ... ***
C
C ... INPUT
C
C ... N        NUMBER OF VERTICES
C ... C(L)     TRAVEL TIME ON ARC(I,J), L=IND(I,J,N). C(L) DENOTES
C              THAT THIS ARC DOES NOT EXIST. IT ASSUMED THAT C(L)=C(M),
C              M=IND(J,I,N), AND THAT EVERY ARC(I,J) IS THE SHORTEST
C              PATH BETWEEN VERTEX I AND J.
C ... NL       NUMBER OF FEASIBLE LINES
C ... NV(I)    NUMBER OF VERTICES BELONGING TO LINE I, I=1,..,NL
C ... NVX(I,J) J-TH VERTEX OF LINE I, J=1,..,NV(I), I=1,..,NL
C ... G(L)     NUMBER OF PEOPLE WHO WANT TO TRAVEL FROM VERTEX I TO J,
C              L=IND(I,J,N). IT IS ASSUMED THAT G(L)=G(M), M=IND(J,I,N).
C ... NBUS     NUMBER OF OPERATING BUSES (TRAMS)
C
C ... OUTPUT
C
C ... TOTO      TOTAL TRANSPORTATION TIME OF ALL PASSENGERS
C ... WAITO     AVERAGE TIME WAITING ON A BUS (TRAM)
C ... NLO      NUMBER OF OPTIMAL LINES
C ... NVO(I)    NUMBER OF VERTICES BELONGING TO LINE I, I=1,..,NLO
C ... NVXO(I,J) J-TH VERTEX OF LINE I, J=1,..,NVO(I), I=1,..,NLO
C ... T(L)     LENGTH OF SHORTEST PATH BETWEEN VERTICES I AND J,
C              L=IND(I,J,N) USING THE OPTIMAL LINES INCLUDING THE WAITING
C              TIMES
C
      SUBROUTINE BUSOPT(N,C,NL,NV,NVX,G,NBUS,TOTO,WAITO,NLO,NVO,NVXO,T)
      INTEGER N,C(1),NL,NV(1),NVX(30,30),G(1),NBUS,TOT,WAIT,NLO,NVO(1)
      INTEGER NVXO(30,30),T(1),FL(900),NX(30,2),NNX,TOTO,WAITO,U(30,4)
C
C ... STEP 1
C
1     TOTO=2**34
      IIX=2
C
C ... STEP 2
C
2     LENG=0
      DO 10 I=1,NL
      DO 15 J=2,NV(I)
      L=IND(NVX(I,J-1),NVX(I,J),N)
15    LENG=LENG+C(L)
10    CONTINUE
      WAIT=LENG/NBUS+1
      CALL DESCRI(N,C,NL,NV,NVX,WAIT,G,TOT,FL,NX,NNX,T)
      IF(TOT .GE. TOTO) GO TO 20
      CALL PBUSOP(NL,NV,NVX,TOT,WAIT,T,N)
      TOTO=TOT
      NLO=NL
      DO 25 I=1,NL
      NVO(I)=NV(I)
      DO 30 J=1,NVO(I)
30    NVXO(I,J)=NVX(I,J)
25    CONTINUE
```

```
      WAITO=WAIT
      GO TO 3
20    CALL DESCRI(N,C,NLO,NVO,NVXO,WAITO,G,TOTO,FL,NX,NNX,T)
      IF(IIX .EQ. 5) RETURN
      WAIT=WAITO
      IIX=IIX+1
      NL=NLO
      DO 35 I=1,NL
      NV(I)=NVO(I)
      DO 40 J=1,NV(I)
40    NVX(I,J)=NVXO(I,J)
35    CONTINUE
      GO TO (1,2,3,4,5), IIX
C
C ... STEP 3
C
3     MC=0
      IIX=3
C
C ... FIND PAIR OF VERTICES BETWEEN WHICH THE NUMBER OF PEOPLE
C ... WHO CHANGE LINES MINUS THE NUMBER OF PEOPLE WHO DO NOT IS
C ... MAXIMUM
C
      NNX1=NNX-1
      DO 45 I=1,NNX1
      NNX2=I+1
      L1=IND(I,I,NNX)
      DO 50 J=NNX2,NNX
      IF(NX(I,1) .NE. NX(J,1)) GO TO 50
      L2=IND(J,J,NNX)
      L3=IND(I,J,NNX)
      L4=IND(J,I,NNX)
      IA=FL(L3)+FL(L4)-FL(L1)-FL(L2)
      IF(MC .GE. IA) GO TO 50
      MC=IA
      IX=I
      JX=J
50    CONTINUE
45    CONTINUE
      IF(MC .EQ. 0) GO TO 4
      LN1=NX(IX,2)
      LN2=NX(JX,2)
      DO 55 I=1,NV(LN1)
      IF(NVX(LN1,I) .NE. NX(IX,1)) GO TO 55
      I1=I
      GO TO 60
55    CONTINUE
60    DO 65 I=1,NV(LN2)
      IF(NVX(LN2,I) .NE. NX(IX,1)) GO TO 65
      I2=I
      GO TO 70
65    CONTINUE
C
C ... COMBINE FIRST PART OF FIRST LINE WITH SECOND PART OF SECOND LINE
C
70    NU1=NV(LN1)
      NU2=NV(LN2)
```

```
      DO 72 I=1,NU1
72    U(I,1)=NVX(LN1,I)
      DO 73 I=1,NU2
73    U(I,2)=NVX(LN2,I)
      I1X=I1+1
      I2X=I2+1
      NV(LN1)=I1+NU2-I2
      NV(LN2)=I2+NU1-I1
      IF(I1X .GT. NU1) GO TO 85
      DO 90 I=I1X,NU1
      J=I-I1X+I2X
90    NVX(LN2,J)=U(I,1)
85    IF(I2X .GT. NU2) GO TO 75
      DO 80 I=I2X,NU2
      J=I-I2X+I1X
80    NVX(LN1,J)=U(I,2)
75    CALL DESCRI(N,C,NL,NV,NVX,WAIT,G,LT1,FL,NX,NNX,T)
      NU3=NV(LN1)
      NU4=NV(LN2)
      DO 92 I=1,NU3
92    U(I,3)=NVX(LN1,I)
      DO 93 I=1,NU4
93    U(I,4)=NVX(LN2,I)
C
C ... COMBINE FIRST PART OF FIRST LINE WITH FIRST PART OF SECOND LINE
C
      NV(LN1)=I1+I2-1
      NV(LN2)=NU1-I1+1+NU2-I2
      IF(I2 .EQ. 1) GO TO 95
      I2Y=I2-1
      DO 100 I=1,I2Y
      J=I1+I2-I
100   NVX(LN1,J)=U(I,2)
95    IF(I1 .EQ. NU1) GO TO 105
      DO 110 I=I1X,NU1
      J=NU1-I+1
110   NVX(LN2,J)=U(I,1)
105   DO 115 I=I2,NU2
      J=I-I2+NU1-I1+1
115   NVX(LN2,J)=U(I,2)
      CALL DESCRI(N,C,NL,NV,NVX,WAIT,G,LT2,FL,NX,NNX,T)
      IF(LT2 .LE. LT1) GO TO 2
      NV(LN1)=NU3
      NV(LN2)=NU4
      DO 120 I=1,NU3
120   NVX(LN1,I)=U(I,3)
      DO 125 I=1,NU4
125   NVX(LN2,I)=U(I,4)
      GO TO 2
C
C ... STEP 4
C
4     IIX=4
      CM=0.
      DO 130 I=1,NL
      DO 135 J=2,NV(I)
      DO 140 K=1,NL
```

```
      IF(I .EQ. K) GO TO 140
      DO 145 L=1,NV(K)
      L1=IND(NVX(I,J-1),NVX(K,L),N)
      KD=C(L1)
      IF(C(L1) .EQ. 0) GO TO 145
      L1=IND(NVX(I,J),NVX(K,L),N)
      KD=C(L1)+KD
      IF(C(L1) .EQ. 0) GO TO 145
      DO 150 M=1,NV(I)
      IF(NVX(K,L) .EQ. NVX(I,M)) GO TO 145
150   CONTINUE
      L1=IND(NVX(I,J-1),NVX(I,J),N)
      KD=KD-C(L1)
      KB=0
      DO 155 M=1,NV(I)
      L1=IND(NVX(I,M),NVX(K,L),N)
155   KB=KB+G(L1)
      BK=KB/FLOAT(KD)
      IF(CM .GE. BK) GO TO 145
      CM=BK
      IF=I
      JF=J
      KF=K
      LF=L
145   CONTINUE
140   CONTINUE
135   CONTINUE
130   CONTINUE
      IF(CM .EQ. 0) GO TO 5
      DO 160 I=JF,NV(IF)
      J=NV(IF)+1+JF-I
160   NVX(IF,J)=NVX(IF,J-1)
      NV(IF)=NV(IF)+1
      NVX(IF,JF)=NVX(KF,LF)
      GO TO 2
C
C ... STEP 5
C
5     CM=2**34
      DO 165 I=1,NL
      IF(NV(I) .LT. 3) GO TO 165
      DO 170 J=3,NV(I)
      L=IND(NVX(I,J-2),NVX(I,J),N)
      KD=-C(L)
      IF(C(L) .EQ. 0) GO TO 170
      DO 175 K=1,NL
      IF(I .EQ. K) GO TO 175
      DO 180 L=1,NV(K)
      IF(NVX(K,L) .EQ. NVX(I,J-1)) GO TO 185
180   CONTINUE
175   CONTINUE
      GO TO 170
185   L=IND(NVX(I,J-2),NVX(I,J-1),N)
      KD=KD+C(L)
      L=IND(NVX(I,J),NVX(I,J-1),N)
      KD=KD+C(L)
      KB=0
```

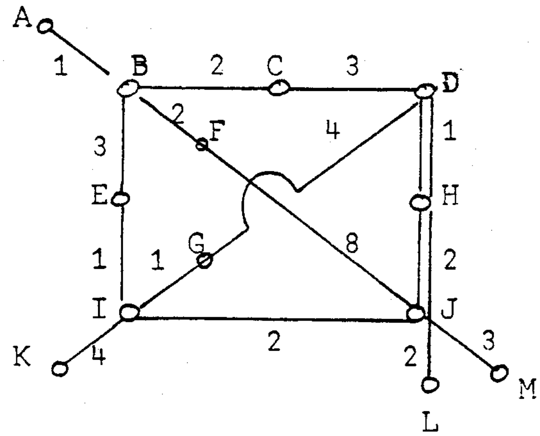
```
      DO 190 K=1,NV(I)
      L=IND(NVX(I,K),NVX(I,J-1),N)
190   KB=KB+2*G(L)
      K1=0
      K2=0
      DO 195 K=1,NNX
      IF(NX(K,1) .NE. NVX(I,J-1)) GO TO 195
      IF(NX(K,2) .NE. I) GO TO 200
      K1=K
      IF(K2 .EQ. 0) GO TO 195
      GO TO 205
200   K2=K
      IF(K1 .EQ. 0) GO TO 195
205   L=IND(K1,K2,NNX)
      KB=KB+FL(L)
      L=IND(K2,K1,NNX)
      KB=KB+FL(L)
195   CONTINUE
      BK=KB/FLOAT(KD)
      IF(CM .LE. BK) GO TO 170
      CM=BK
      IF=I
      JF=J
170   CONTINUE
165   CONTINUE
      IF(CM .EQ. 2**34) RETURN
      DO 210 I=JF,NV(IF)
210   NVX(IF,I-1)=NVX(IF,I)
      NV(IF)=NV(IF)-1
      GO TO 2
      END
```

Having finally found a good set of lines there is one problem left. As already said in chapter 7.1., public and individual transportation facilities are in a competitive situation, especially in urban areas. Assuming that at least some people will make their transportation mode decision depending on the total travel time, it is meaningful to compare the travel time between each pair of vertices of the public transportation network and the equivalent pair of vertices of the road network for cars. If it turns out that for two vertices with high demand between them the travel time on the public transportation system is much greater, because no direct line is connecting the two vertices, then the transportation planner, who wants to convince people rather to use public transportation facilities, should consider the possibility of including a direct line between such vertices. So far, no algorithm exists to perform such considerations automatically.

Although for the purpose of the algorithm we assumed that on each line the buses run in the same frequency, this might not be the best choice. Thus, finding an optimal scheduling for the buses still remains to be solved. This is not a problem of network optimization and therefore is beyond the scope of this book. The interested reader should look at Friedman (1976) and Uebe (1970).

7.5. Exercise

Given a set of bus stations
X = (A,B,C,D,E,F,G,H,I,J,K,L,M)
These bus stations are served by 3 bus lines in the following way:
bus line 1: A - B - F - J - M
bus line 2: B - E - I - J - H - D - C - B
bus line 3: K - I - G - D-- H - J - L



The numbers on the arcs denote travel time in minutes. The waiting time for a bus, if a change is necessary, is 3 minutes. Find the shortest paths between all pairs of bus stations.

8. References

W.Ahrens (1974), Die lösung eines nichtlinearen Investitionsproblems mit Hilfe binärer Optimierung, Zeitschrift für OR 18, B131-B147.

H.Ajzen & L.Rokeach (1974), Routenwahl in Liniennetzen, Zeitschrift für OR 18, B1o1-B12o.

E.Beltrami (1976), Models for public systems analysis, Academic Press.

——————— & L.Bodin (1974), Networks and vehicle routing for municipal waste collection, Network 4, 65-94

J.Byrd (1975), OR models for public administration, Lexington Books

R.Cembrovicz (1972), Ein Modell zur Investitionsoptimierung beim Aufbau von regionalen Abwasserbeseitigungssystemen, Wasser und Abwasser in Forschung und Praxis 5, 227-241

C.Chriqui & P.Robillard (1975), Common bus lines, Transportation Science 9, 115 -121

N.Christofides (1974), Optimal expansion of an existing network, Math.Progr.6, 197-211

——————— (1975), Graph theory an algorithm approach, Academic Press

A.Drake, R.Keeney & P.Morse (1972), Analysis of public systems, MIT Press

M.Florian, S.Nguyen & J.Ferland (1975), On the combined distribution - assignment of traffic, Transportation Science 9, 43-53

M.Florian (1976), Traffic equilibrium methods, Lecture Notes in Economics and Mathematical System 118, Springer Verlag

M.Friedman (1976), A mathematical programming model for optimal scheduling of buses departures under deterministic conditions, Transp.Res.1o, 83-9o.

S.Gass & R.Sisson (1975), A guide to models in governmental planning and operations, Sauger Books

B.Golden & T.Magnanti (1977), Deterministic network optimization: a bibliography, Networks 7, 149-183.

M.Greenberger, M.Crenson & B.Crissey (1976), Models in the policy process:public decision making in the computer era, Russell Sage Foundation

S.Hakimi (1964), Optimal locations of switching centers and the absolute centers and medians of a graph, Opns.Res. 12, 45o-459

———— (1965), Optimun distribution of switching centers in a communication network and some related graph theoretic problems, Opns.Res.13, 462-475

H.Hensel & P.Mäcke (1975), Arbeitsmethode der städtischen Verkehrsplanung, Bauverlag

H.-P.Hoidn (1977), Busnetz von Aarau: Neukonzeption der Linien, IFOR-Studienberichte 6, ETH-Zürich

T.Hu (197o), Integer programming and network flows, Addison Wesley

R.Karp (1975), On the computational complexity of combinatorial problems, Networks 5, 45-68

Th.Knecht (1975), Sequentielle Bauweise eines Abwasserreinigungssystems, IFOR-Studienberichte 3, ETH-Zürich 1-18

W.Knödel (1969), Graphentheoretische Methoden und ihre Anwendungen, Springer Verlag

W.Lampkin & P.D.Saalmans (1967), The design of routes, service frequencies and schedules for a municipal bus undertaking: a case study, OR Quarterly 18, 375-397

L.Leblanc (1975), An algorithm for the discrete network design problem, Trans.Sci.9, 183-199

Th.Liebling (197o), Graphentheorie in Planungs- und Tourenproblemen, Lecture Notes in OR 21, Springer Verlag

R.Mackinnon (1976), Optimization models of transportation network improvement: review and future prospects, IIASA, Austria

R.Miller (1967), An optimization model for transportation planning, Transportation Research 1, 271-287

R.M.Newton & W.H.Thomas (1974), Bus routing in a multischoolsystem, Comput. & Ops.Res.1, 213-222

S.Nguyen (1974), An algorithm for the traffic assignment problem, Transportation Science 8, 2o3-216

R.Oliver & R.Potts (1972), Flows in transportation networks, Academic Press

A.Polyméris (1977) Optimierung und Aufteilung der Kosten regionaler Abwasserverbände, in: Kombinatorische Entscheidungsprobleme, Kursunterlagen des Instituts für OR, ETH-Zürich

Ch.Revelle, D.Marks & L.Liebmann (197o), An analysis of private and public sector location models, Man.Sci.16, 692-7o7

B.Rothfarb, H.Frank, D.Rosenbaum, K.Steiglitz & D.Kleitman (197o), Optimal design of offshore natural-gas pipeline systems, Operations Research 18, 992-1o2o

L.Silman, Z.Barzily & U.Passy (1974), Planning the route system for urban buses, Comput.& Ops.Res.1, 2o1 - 211

P.Steenbrink (1974), Optimization of transport networks, John Wiley & Sons

C.Toregas, R.Swain, Ch.Revelle & L.Bergmann (1971), The location of emergency service facilities, Opns.Res.19, 1363-1373

G.Uebe (197o), Optimale Fahrpläne, Lecture Notes in Economics and Mathematical Systems 2o, Springer Verlag

F.Weinberg et al.(1976), Operations Research im öffentlichen Dienst, Verlag Paul Haupt

N.Zadeh (1973), Construction of efficient tree networks: the pipeline problem, Networks 3, 1-31

——— (1974), On building minimum cost communication networks over time, Networks 4, 19-34